Using lumi, a package processing Illumina Microarray

Pan Du[‡,*], Warren A. Kibbe[‡,†], Simon Lin[‡,‡]

April 25, 2007

[‡]Robert H. Lurie Comprehensive Cancer Center
Northwestern University, Chicago, IL, 60611, USA

Contents

[*]dupan@northwestern.edu
[†]wakibbe@northwestern.edu
[‡]s-lin2@northwestern.edu

1 Overview of lumi

Illumina microarray is becoming a popular microarray platform. The BeadArray technology from Illumina makes its preprocessing and quality control different from other microarray technologies. Unfortunately, until now, most analyses have not taken advantage of the unique properties of the BeadArray system. The *lumi* Bioconductor package especially designed to process the Illumina microarray data. The *lumi* package provides an integrated solution for the bead-level Illumina microarray data analysis. The package covers data input, quality control, variance stabilization, normalization and gene annotation.

The *lumi* package includes a new variance-stabilizing transformation (VST) algorithm that takes advantage of the technical replicates available on every Illumina microarray. A new robust spline normalization (RSN) algorithm, which combines the features of the quantile and loess normalization, is also implemented in this package. Options available in other popular normalization methods are also provided. Multiple quality control plots are provided in the package. To better annotate the Illumina data, a new, vendor independent nucleotide universal identifier (nuID) was devised to identify the probes of Illumina microarray. The nuID indexed Illumina annotation packages is compatible with other Bioconductor annotation packages. Mappings from Illumina Target Id or Probe Id to nuID are also included in the annotation packages. The output of lumi processed results can be easily integrated with other microarray data analysis, like differentially expressed gene identification, gene ontology analysis or clustering analysis.

2 Object models of major classes

The *lumi* package has one major class: **LumiBatch**. **LumiBatch** is inherited from **ExpressionSet** class in Bioconductor for better compatibility. Their relations are shown in Figure 1. **LumiBatch** class includes *se.exprs*, *beadNum* and *detection* in **assayData** slot for additional informations unique to Illumina microarrays. A controlData slot is used to keep the control probe information, and a QC slot is added for keeping the quality control information. The S4 function `plot` supports different kinds of plots by specifying the specific plot type of **LumiBatch** object. See help of `plot-methods` function for details. The *history* slot records all the operations made on the **LumiBatch** object. This provides data provenance. Function `getHistory` is to retrieve the *history* slot. Please see the help files of **LumiBatch** class for more details. A series of functions: `lumiR`, `lumiB`, `lumiT`, `lumiN` and `lumiQ` were designed for data input, preprocessing and quality control. Function `lumiExpresso` encapsulates the preprocessing methods for easier usability.

3 Data preprocessing

The first thing is to load the *lumi* package.

```
> library(lumi)

This is mgcv 1.3-23
```

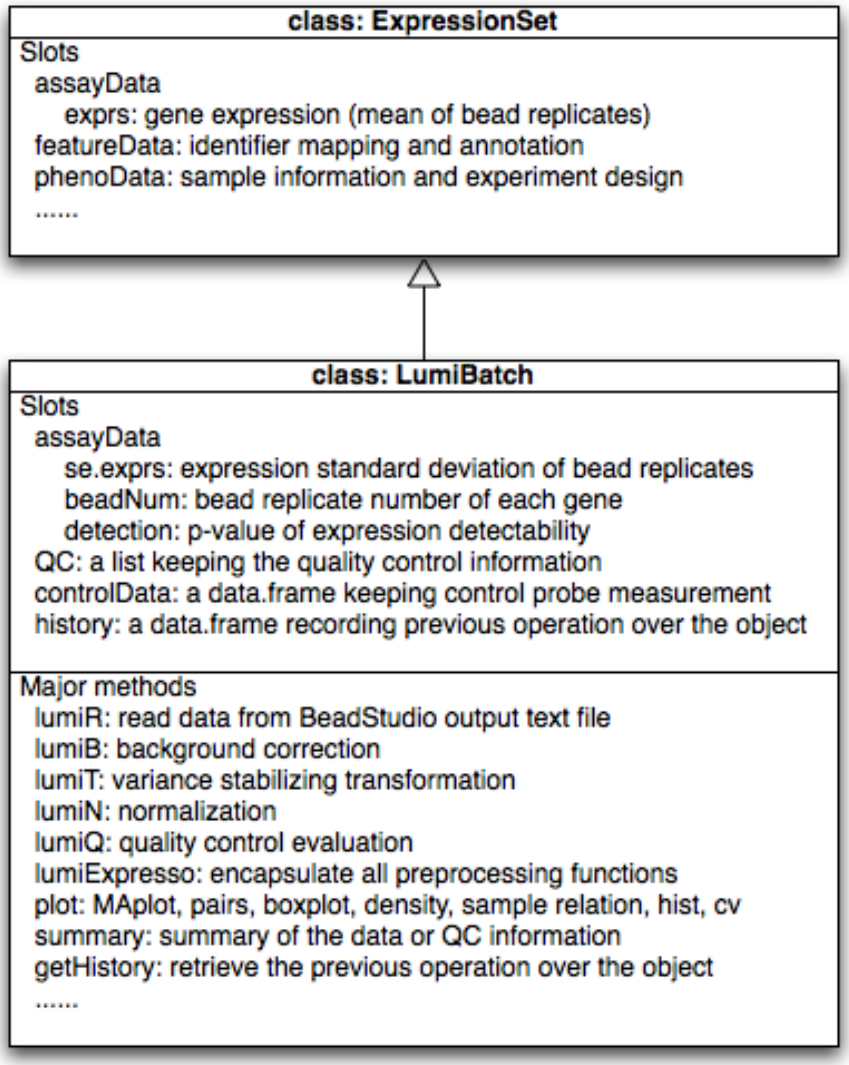


Figure 1: Object models in lumi package

3.1 Input data with "intelligent-load" technology

The `lumiR` function supports directly reading the Illumina raw data output of the Illumina Bead Studio toolkit from version 1 to version 3. It can automatically detect the BeadStudio output version and format and create a new **LumiBatch** object for it. An example of the input data format is shown in in Figure 2. For simplicity, only part of the data of first sample is shown. The data in the highlighted columns are kept in the corresponding slots of **LumiBatch** object, as shown in Figure 2. The `lumiR` function will automatically determine the starting line of the data. The columns with header including `AVG_Signal` and `BEAD_STD` are required for the **LumiBatch** object. The sample IDs and sample labels are extracted from the column names of the data file. For example, based on the column name: `AVG_Signal-1304401001_A`, we will extract `"1304401001"` as the sample ID and `"A"` as the sample label (The function assumes the separation of the sample ID and the sample label is `"_"` if it exists in the column name.). The algorithm will check the uniqueness of sample IDs. If the sample ID is not unique, the entire portion after removing `"AVG_Signal"` will be used as a sample ID. A file format error will be reported if the uniqueness still cannot be satisfied in this way. The `lumiR` will automatically initialize the QC slot of the **LumiBatch** object by calling `lumiQ`.

If BeadStudio outputted the control probe data, their information will be kept in the controlData slot of the **LumiBatch** object. If BeadStudio outputted the sample summary information, which is called [Samples Table] in the output text file, the information will be kept in BeadStudioSummay within the QC slot of the **LumiBatch** object.

The BeadStudio can output the gene profile or the probe profile. As the probe profile provides unique mapping from the probe Id to the expression profile, outputting probe profile is preferred. When the probe profile is outputted, as show in Figure 2(B), the ProbeId column will be used as the identifier of **LumiBatch** object.

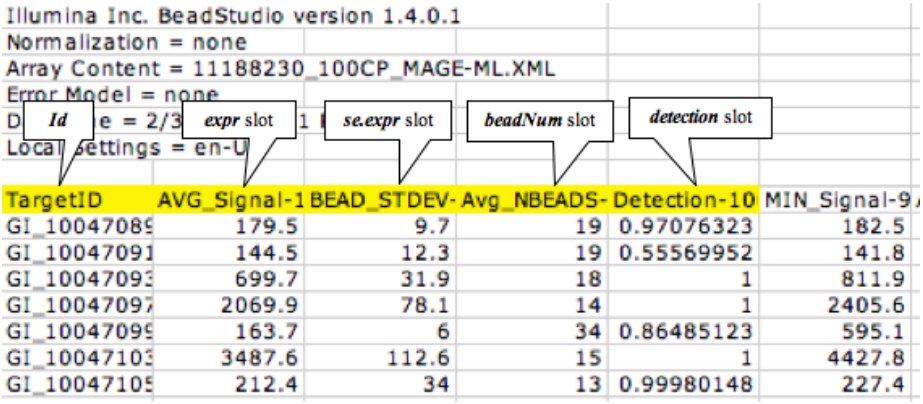
We strongly suggest outputting the header information when using BeadStudio, as shown in Figure 2.

If a lumi annotation library is provided, the `lumiR` function will automatically mapping the ProbeId or TargetID as nuID (see annotation section for more details), and keep the mapping information in the featureData of the **LumiBatch** object.

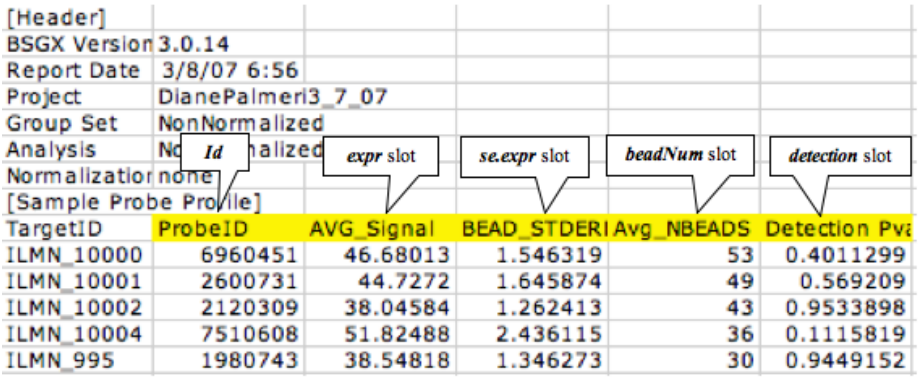
```
> ## specify the file name
> # fileName <- 'Barnes_gene_profile.txt'          # Not Run
> ## load the data
> # x.lumi <- lumiR(fileName, lib='lumiHuamnV1')                # Not Run
```

Here, we just load the pre-saved example data, example.lumi, which is a subset of the experiment data package *lumiBarnes* in the Bioconductor. The example data includes four samples "A01", "A02", "B01" and "B02". "A" and "B" represent different Illumina slides (8 microarrays on each slide), and "01" and "02" represent different samples. That means "A01" and "B01" are technique replicates at different slides, the same for "A02" and "B02".

```
> ## load example data
> data(example.lumi)
```



(A) BeadStudio version 1



(B) BeadStudio version 3

Figure 2: An example of the input data format

```
> ## summary of the example data
> example.lumi

Summary of BeadStudio output:
        Illumina Inc. BeadStudio version 1.4.0.1
        Normalization = none
        Array Content = 11188230_100CP_MAGE-ML.XML
        Error Model = none
        DateTime = 2/3/2005 3:21 PM
        Local Settings = en-US

Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
3 2007-04-22 00:13:06 2007-04-22 00:13:10
4 2007-04-22 00:59:20 2007-04-22 00:59:36
                                          command lumiVersion
1           lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                             lumiQ(x.lumi = x.lumi)       1.1.6
3 addNuId2lumi(x.lumi = x.lumi, lib = "lumiHumanV1")       1.1.6
4            Subsetting 8000 features and 4 samples.       1.1.6

Object Information:
LumiBatch (storageMode: lockedEnvironment)
assayData: 8000 features, 4 samples
  element names: beadNum, detection, exprs, se.exprs
phenoData
  rowNames: A01, A02, B01, B02
  varLabels and varMetadata:
    sampleID: The unique Illumina microarray Id
    label: The label of the sample
featureData
  rowNames: oZsQEQXp9ccVIlwoQo, 9qedFRd_5Cul.ueZeQ, ..., 33KnLHy.RFaieogAF4 (8000 total)
  varLabels and varMetadata:
    TargetID: The Illumina microarray identifier
    presentCount: The number of detectable measurements of the gene
experimentData: use 'experimentData(object)'
Annotation character(0)
```

3.2 Quality control of the raw data

The quality control of a **LumiBatch** object includes a data summary (the mean and standard deviation, sample correlation, detectable probe ratio of each sample (microarray)) and different quality control plots.

In the featureData, it also records the presentCount of each probe, which measures the number of detectable measurements (the detection p-value less than the user provided threshold (0.01 by default)) of each probe.

`LumiQ` function will produce the data summary of a **LumiBatch** object and organize the results in a QC slot of **LumiBatch** object. When creating the

LumiBatch object, the `LumiQ` function will be called to initialize the QC slot of the **LumiBatch** object.

Summary of the quality control information of example.lumi data. If the QC slot of the **LumiBatch** object is empty, function `lumiQ` will be automatically called to estimate the quality control information.

```
> summary(example.lumi, "QC")

Data dimension:  8000 genes x 4 samples

Summary of Samples:
                            A01    A02     B01     B02
mean                     8.3240  8.568  8.2580  8.3470
standard deviation       1.5580  1.686  1.7230  1.6690
detection rate(0.01)     0.5432  0.564  0.5774  0.5758
distance to sample mean 76.9500 65.280 88.3200 49.1100

Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
                                    command lumiVersion
1 lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                   lumiQ(x.lumi = x.lumi)       1.1.6
```

The S4 method `plot` can produce the quality control plots of **LumiBatch** object. The quality control plots includes: the density plot (Figure 3), box plot (Figure 4), pairwise correlation between microarrays (Figure 5), pairwise MAplot between microarrays (Figure 6), density plot of coefficient of varience, (Figure 7), and the sample relations (Figure 8). More details are in the help of `plot,LumiBatch-method` function. Most of these plots can also be plotted by the extended general functions: `hist` (for density plot), `boxplot`, `MAplot`, `pairs`.

Plot the density plot of the **LumiBatch** object. See Figure 3.

```
> ## plot the density
> plot(example.lumi, what='density')
> ## or
> hist(example.lumi)
```

Plot the box plot of the **LumiBatch** object. See Figure 4.

```
> ## plot the box plot
> plot(example.lumi, what='boxplot')
> ## or
> boxplot(example.lumi)
```

Plot the pairwise sample correlation of the **LumiBatch** object. See Figure 5.

```
> ## plot the pair plot
> plot(example.lumi, what='pair')
> ## or
> pairs(example.lumi)
```

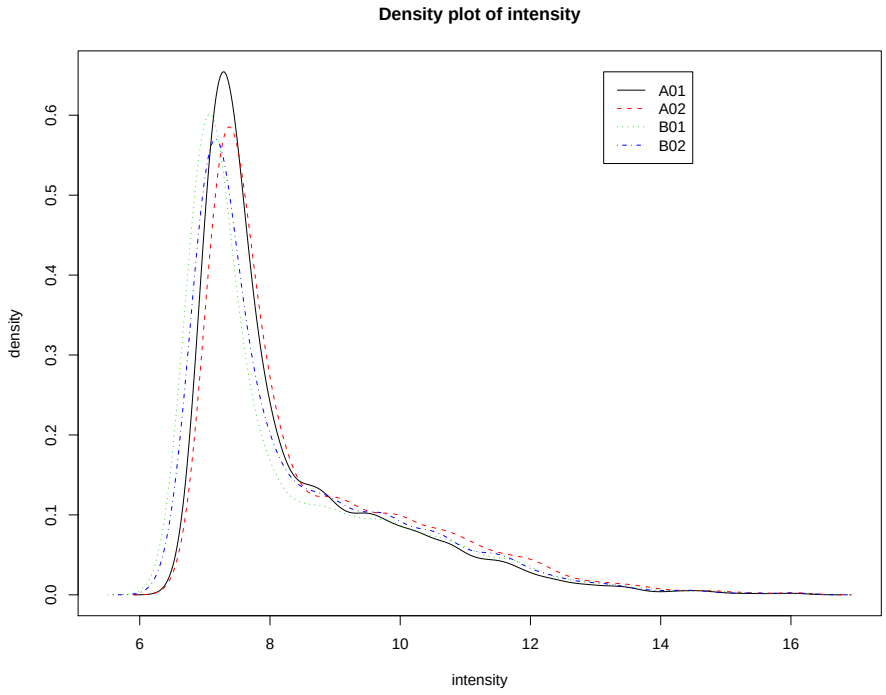


Figure 3: Density plot of Illumina microarrays before normalization

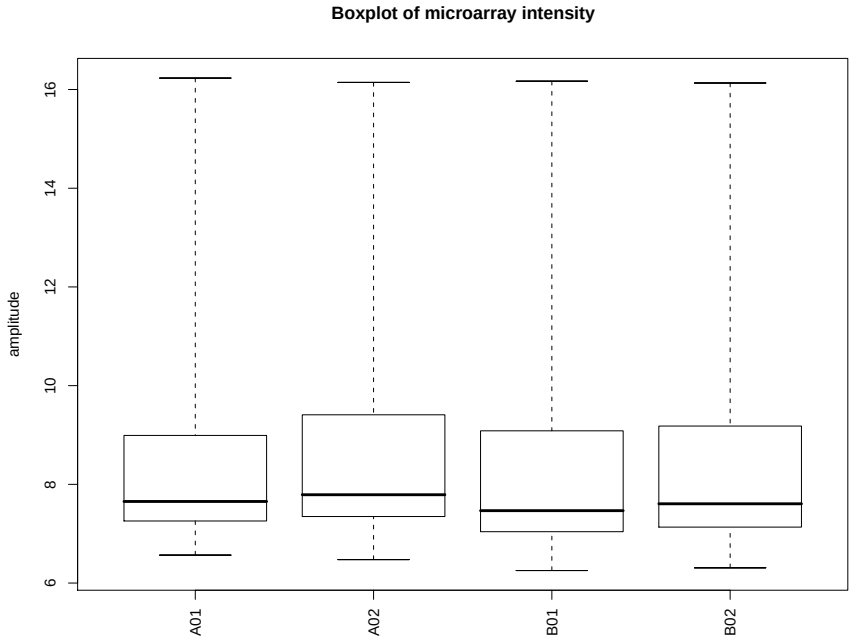


Figure 4: Density plot of Illumina microarrays before normalization

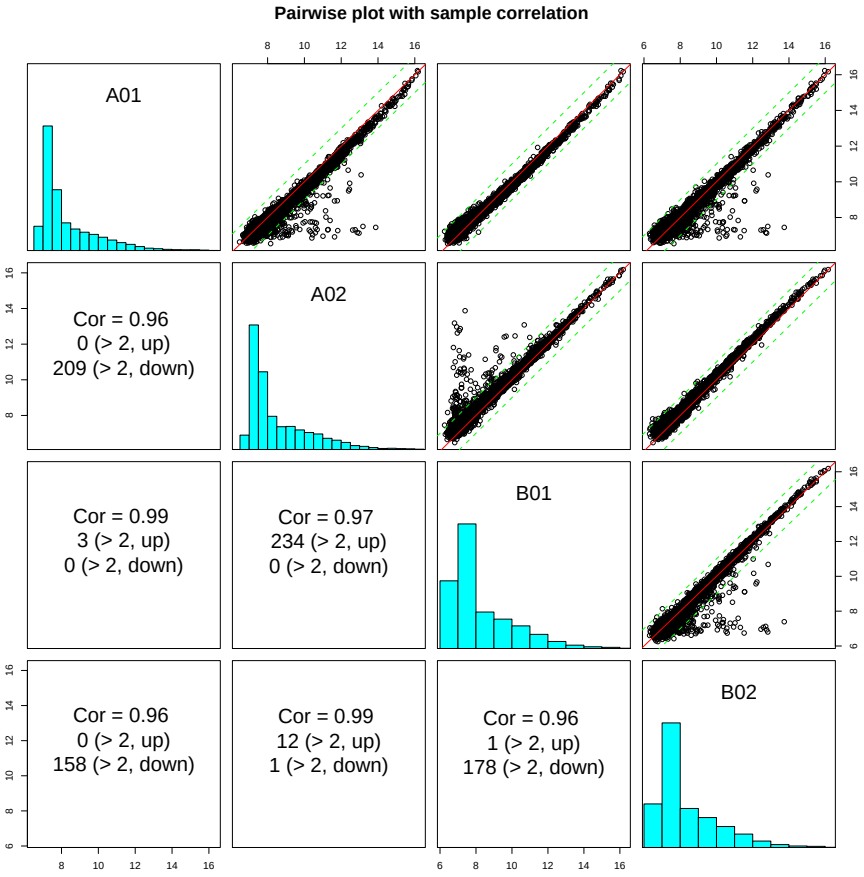


Figure 5: Pairwise plot with microarray correlation before normalization

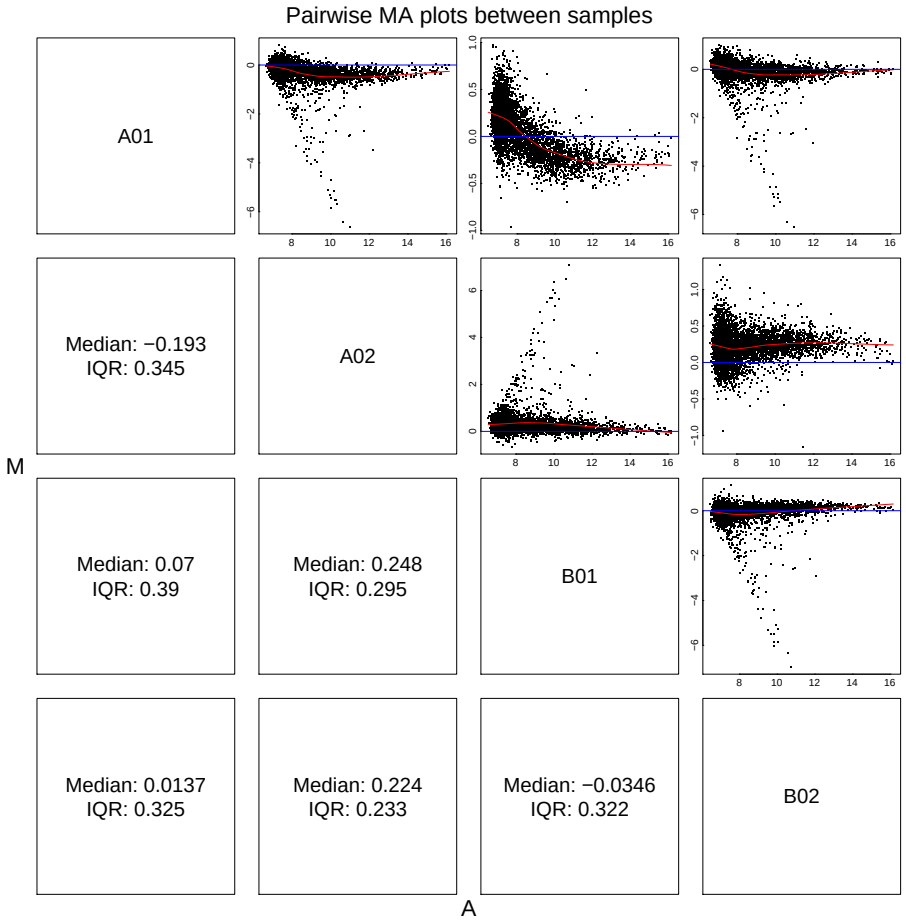


Figure 6: Pairwise MAplot before normalization

The MA plot of the **LumiBatch** object. See Figure 6.

```
> ## plot the MAplot
> plot(example.lumi, what='MAplot')
> ## or
> MAplot(example.lumi)
```

The density plot of the coefficient of variance of the **LumiBatch** object. See Figure 7.

Plot sample relations using hierarchical clustering, see Figure 8

Plot the sampleRelation using MDS, see Figure 9. The color of the sample is based on the sample type, which is `"01", "02", "01", "02"` for the sample data. Please see the help of `getSampleRelation` and `plot-methods` for more details.

```
> ## plot the sample relations
> plot(example.lumi, what='sampleRelation', method='mds', color=c("01", "02", "01", "02"))
> ## or
> plotSampleRelation(example.lumi, method='mds', color=c("01", "02", "01", "02"))
```

```
> plot(example.lumi, what = "cv")
```

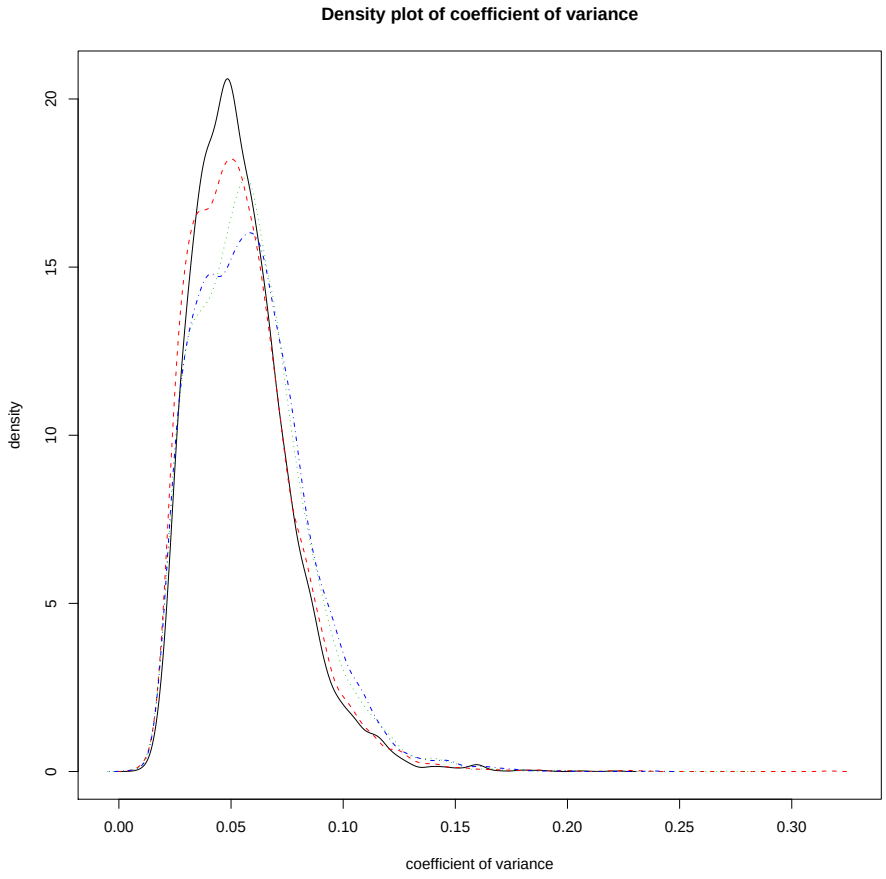


Figure 7: Density Plot of Coefficient of Varience

```
> plot(example.lumi, what = "sampleRelation")
```

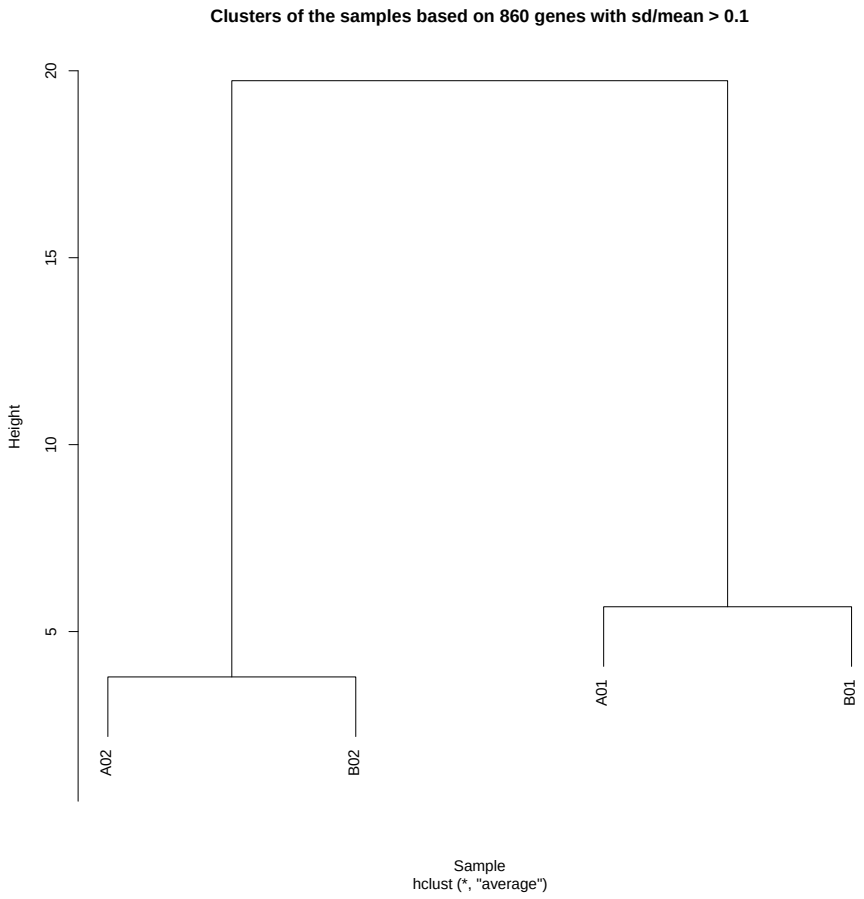


Figure 8: Sample relations before normalization

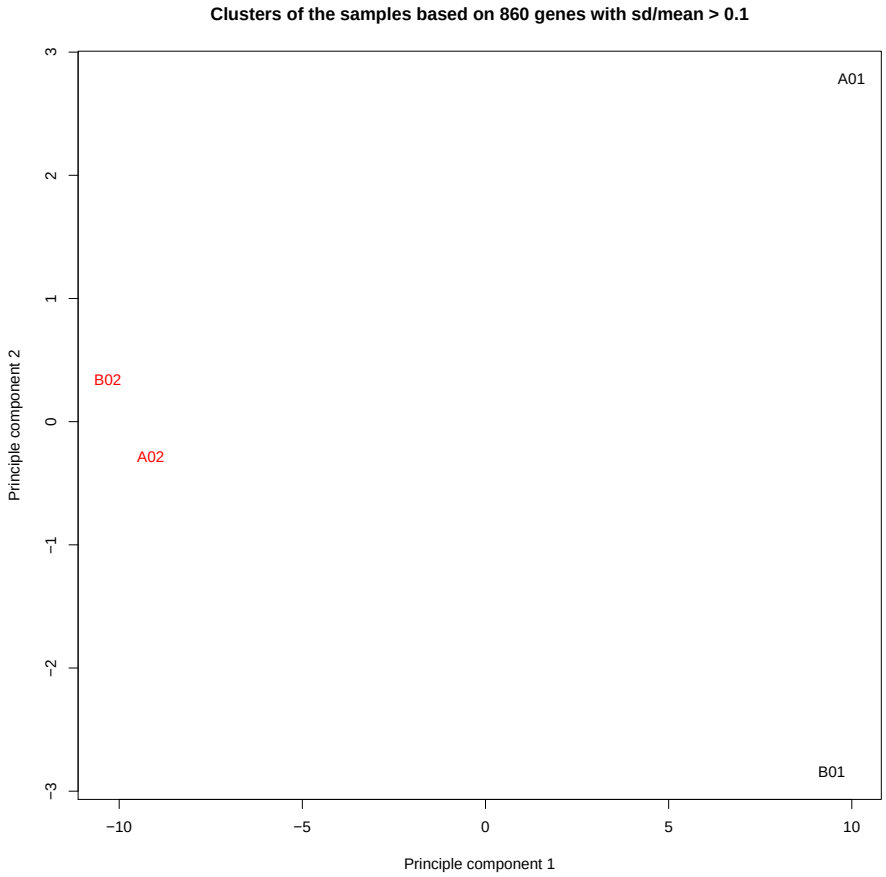


Figure 9: Sample relations before normalization

3.3 Background correction

The *lumi* package provides `lumiB` function for background correction. As both `vst` and `log2` transforms require the expression value to be positive. The default background correction method ('forcePositive') will force all expression values to be positive by adding an offset (minus minimum value plus one). It does nothing if all expression values are positive. Other options of `lumiB` include 'none' and 'bg.adjust'. 'bg.adjust' method is based on the `bg.adjust` function in *affy* package. User can also provide their own background correction function with a LumiBatch Object as the first argument and return a LumiBatch Object with background corrected. See `lumiB` help document for more details.

3.4 Variance stabilizing transform

Variance stabilization is critical for subsequent statistical inference to identify differential genes from microarray data. We devised a variance-stabilizing transformation (VST) by taking advantages of larger number of technical replicates available on the Illumina microarray. Please see [1] for details of the algorithm.

Function `lumiT` performs variance stabilizing transform with both input and output being **LumiBatch** object.

Do default VST variance stabilizing transform

```
> lumi.T <- lumiT(example.lumi)

2007-04-25 02:39:16 , processing array  1
2007-04-25 02:39:16 , processing array  2
2007-04-25 02:39:16 , processing array  3
2007-04-25 02:39:16 , processing array  4
```

The `plotVST` can plot the transformation function of VST, see Figure 10, which is close to log2 at high expression values, see Figure 11. Function `lumiT` also provides options to do `"log2" or "cubicRoot"` transform. See help of `lumiT` for details.

```
> ## plot VST transformation
> trans <- plotVST(lumi.T)
> ## compare the log2 and VST transform
> matplot(log2(trans$untransformed), trans$transformed, main='compare VST and log2 transfo
```

3.5 Data normalization

We proposed a robust spline normalization (RSN) algorithm, which combines the features of quanitle and loess nor-malization, is designed to normalize the variance-stabilized data. Please see [1] for details of the algorithm.

Function `lumiN` performs robust spline normalization (RSN) algorithm with both input and output being **LumiBatch** object. `lumiN` also provides options to do `"loess", "quantile", "vsn"` normalization. See help of `lumiN` for details.

Do default RSN between microarray normaliazation

```
> lumi.N <- lumiN(lumi.T)
```

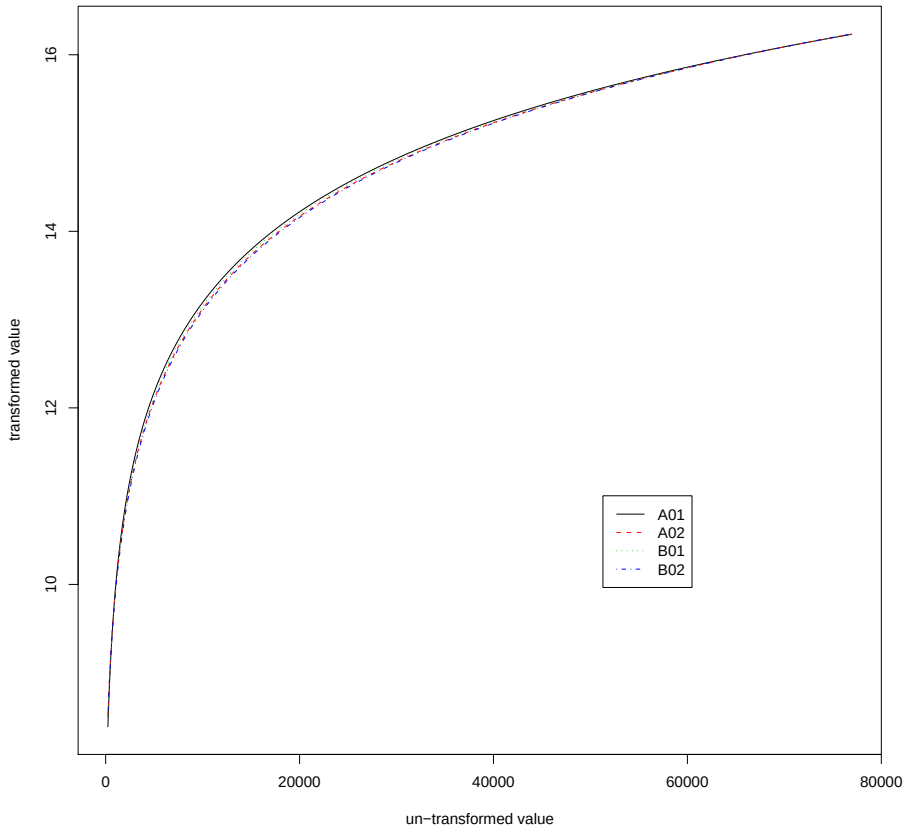


Figure 10: VST transformation

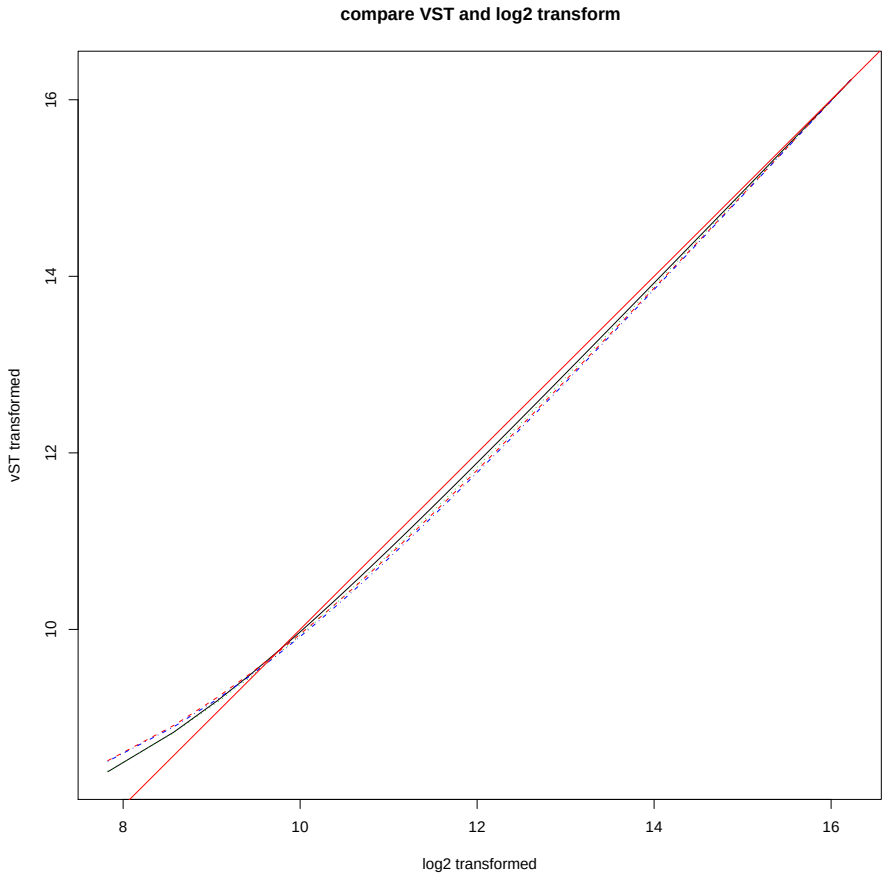


Figure 11: Compare VST and log2 transform

```
2007-04-25 02:39:16 , processing array  1
2007-04-25 02:39:16 , processing array  2
2007-04-25 02:39:16 , processing array  3
2007-04-25 02:39:17 , processing array  4
```

User can also easily select other normalization method. For example, the following command will run quantile between microarray normaliazation.

```
> lumi.N <- lumiN(lumi.T, method = "quantile")
```

3.6 Quality control after normalization

To make sure the data quality meets our requirement, we do a second round of quality control of normalized data with different QC plots. Compare the plots before and after normalization, we can clearly see the improvements.

```
> lumi.N.Q <- lumiQ(lumi.N)
> summary(lumi.N.Q, "QC")

Data dimension:  8000 genes x 4 samples

Summary of Samples:
                            A01    A02     B01     B02
mean                     8.9840  8.984  8.9840  8.9830
standard deviation       1.2200  1.221  1.2210  1.2200
detection rate(0.01)     0.5432  0.564  0.5774  0.5758
distance to sample mean 13.8300 13.580 13.8300 13.9800

Major Operation History:
            submitted            finished
1 2007-04-22 00:08:36 2007-04-22 00:10:36
2 2007-04-22 00:10:36 2007-04-22 00:10:38
3 2007-04-22 00:13:06 2007-04-22 00:13:10
4 2007-04-22 00:59:20 2007-04-22 00:59:36
5 2007-04-25 02:39:16 2007-04-25 02:39:16
6 2007-04-25 02:39:16 2007-04-25 02:39:17
7 2007-04-25 02:39:17 2007-04-25 02:39:17
                                                 command lumiVersion
1           lumiR("../data/Barnes_gene_profile.txt")       1.1.6
2                             lumiQ(x.lumi = x.lumi)       1.1.6
3 addNuId2lumi(x.lumi = x.lumi, lib = "lumiHumanV1")       1.1.6
4            Subsetting 8000 features and 4 samples.       1.1.6
5                       lumiT(x.lumi = example.lumi)       1.2.0
6                             lumiN(x.lumi = lumi.T)       1.2.0
7                             lumiQ(x.lumi = lumi.N)       1.2.0
```

3.7 Encapsulate the processing steps

The `lumiExpresso` function is to encapsulate the major functions of Illumina preprocessing. It is organized in a similar way as the `expresso` function in *affy*

```
> plot(lumi.N.Q, what = "density")
```

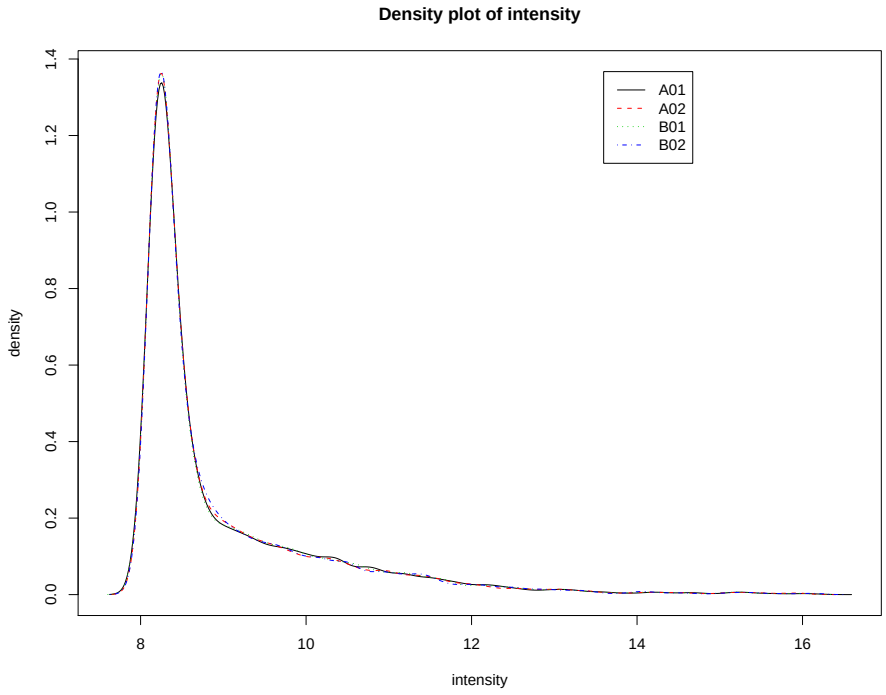


Figure 12: Density plot of Illumina microarrays after normalization

```
> plot(lumi.N.Q, what = "boxplot")
```

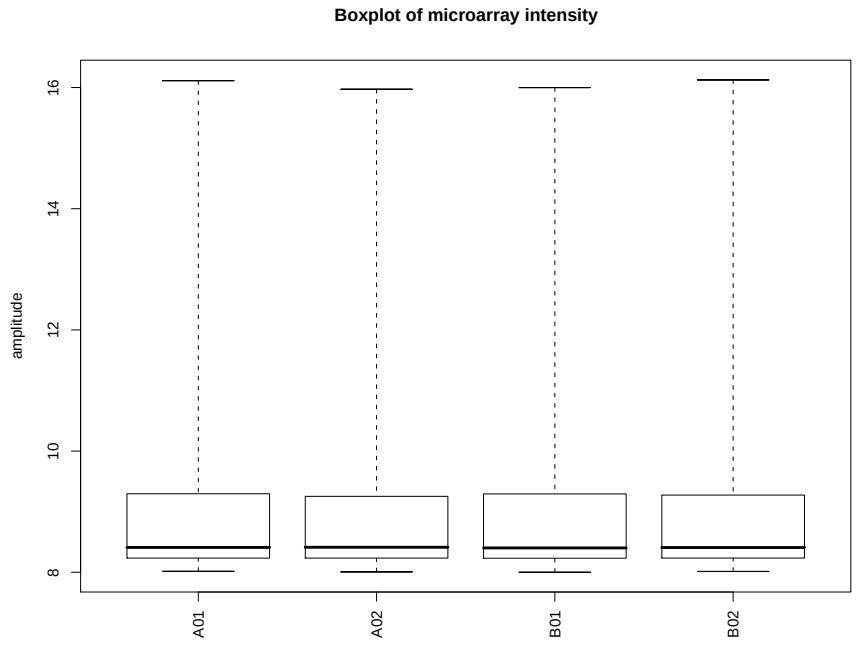


Figure 13: Density plot of Illumina microarrays after normalization

```
> plot(lumi.N.Q, what = "pair")
```

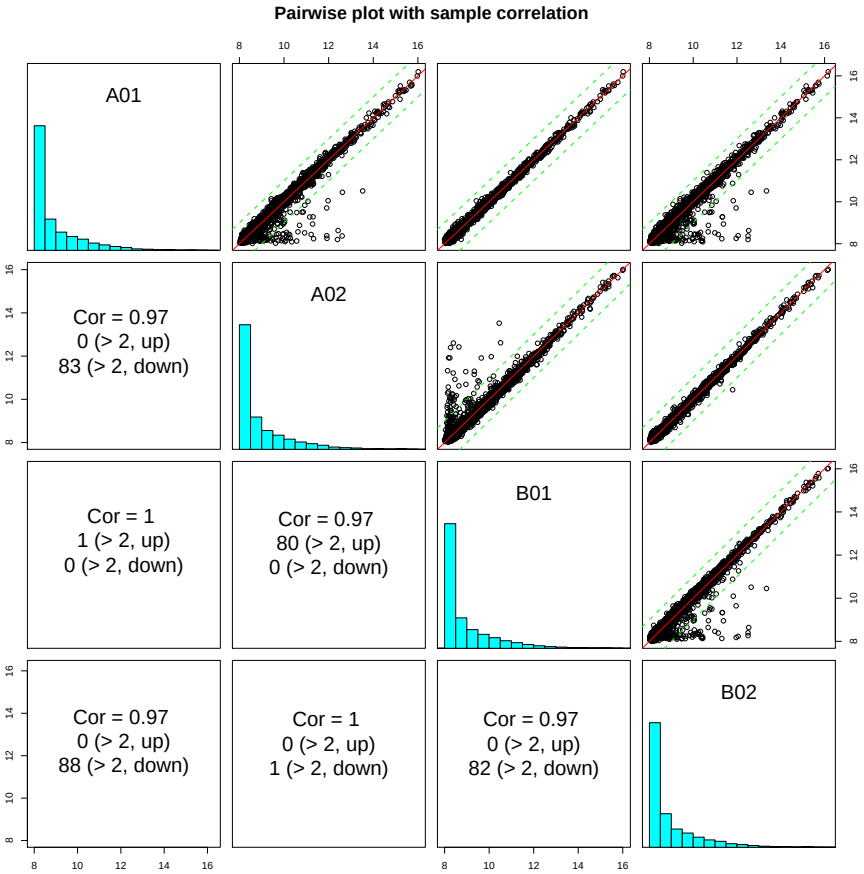


Figure 14: Pairwise plot with microarray correlation after normalization

```
> plot(lumi.N.Q, what = "MAplot")
```

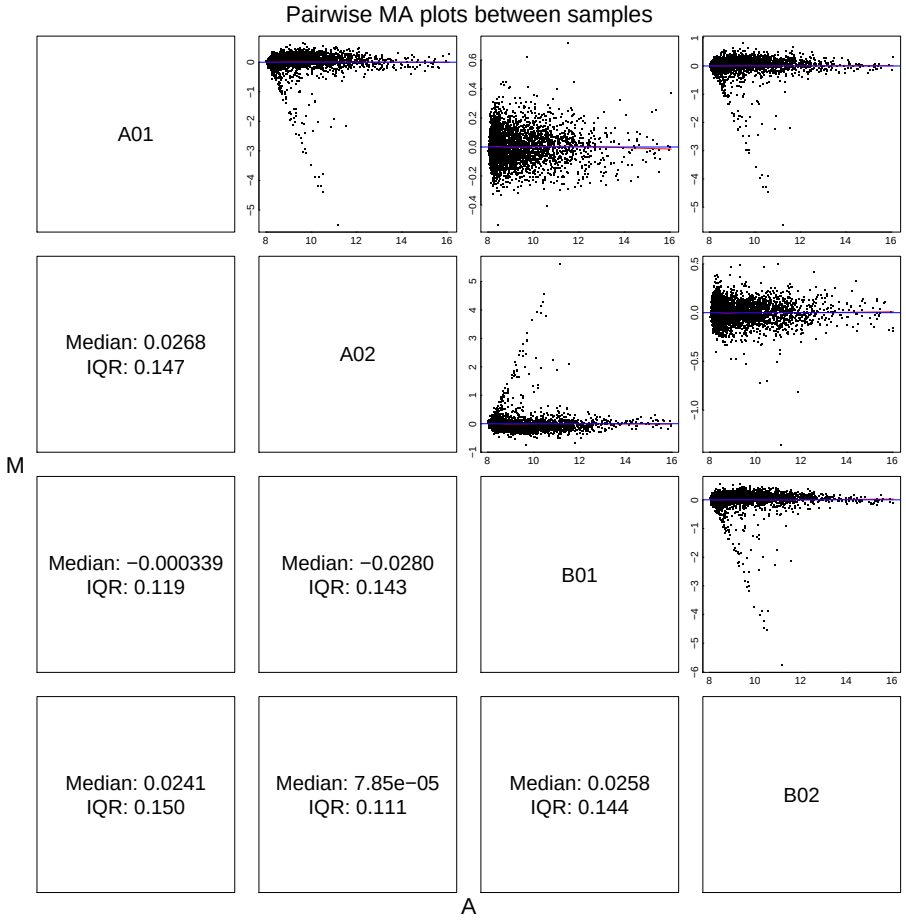


Figure 15: Pairwise MAplot after normalization

```
> plot(lumi.N.Q, what = "sampleRelation")
```

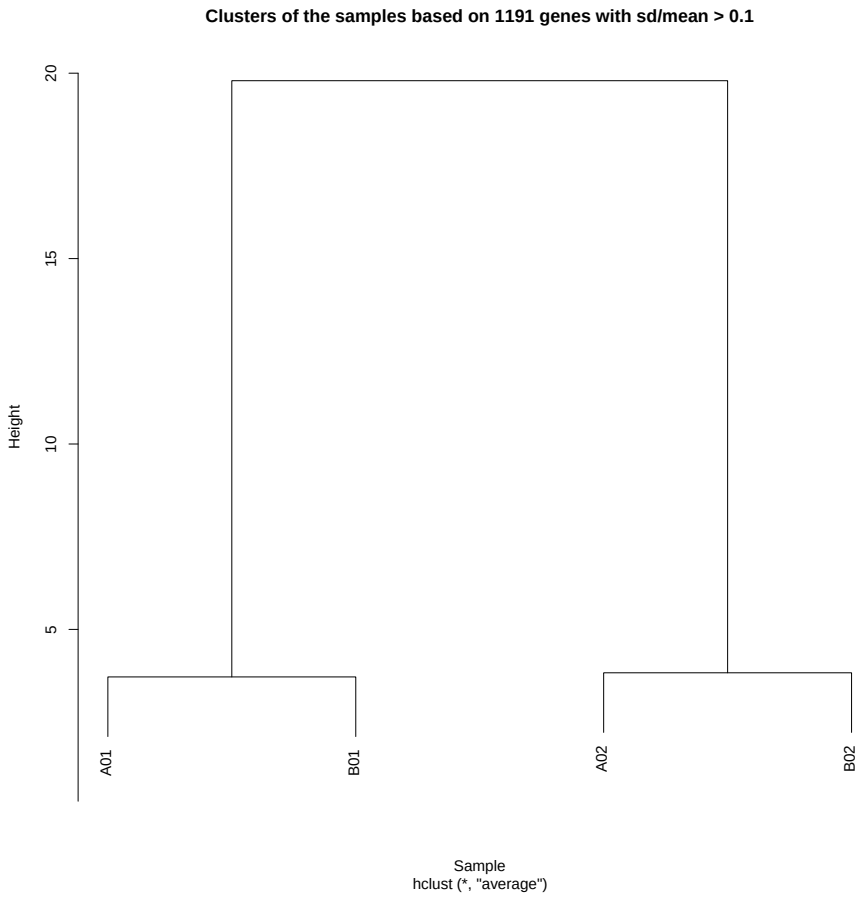


Figure 16: Sample relations after normalization

```
> plot(lumi.N.Q, what = "sampleRelation", method = "mds", color = c("01",
+     "02", "01", "02"))
```

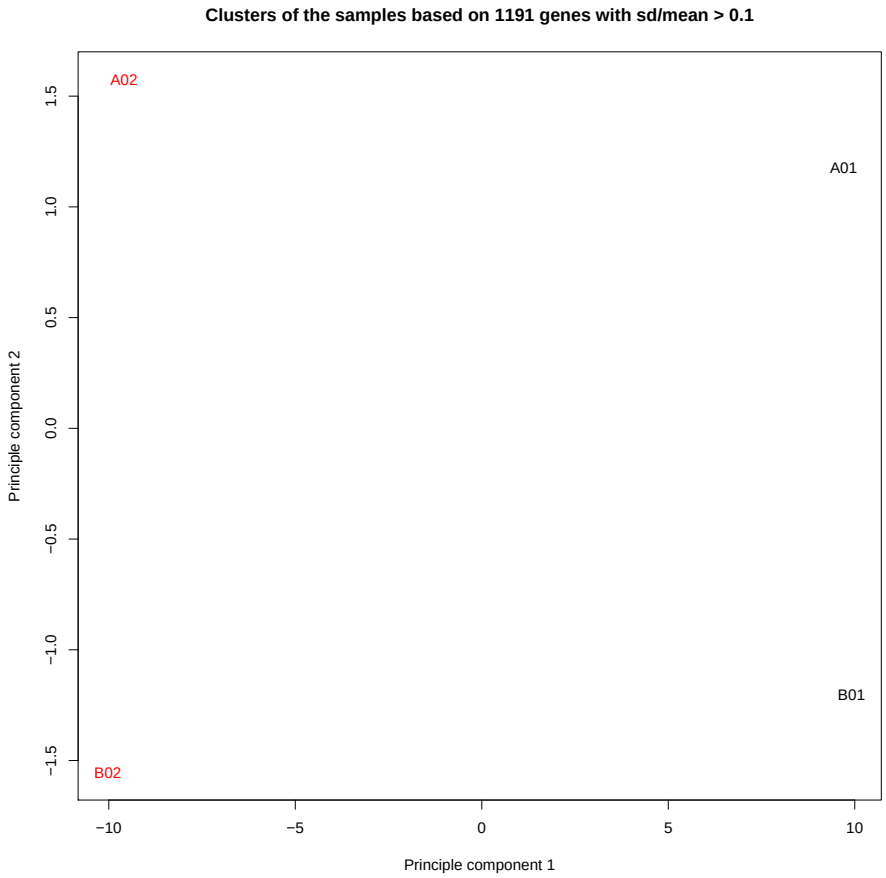


Figure 17: Sample relations after normalization

package. The following code basically did the same processing as the previous multi-steps and produced the same results lumi.N.Q.

```
> lumi.N.Q <- lumiExpresso(example.lumi)

Variance Stabilizing Transform: vst
normalization: rsn

Variance stabilizing ...
2007-04-25 02:39:26 , processing array  1
2007-04-25 02:39:26 , processing array  2
2007-04-25 02:39:26 , processing array  3
2007-04-25 02:39:26 , processing array  4
done.
Normalizing ...
2007-04-25 02:39:26 , processing array  1
2007-04-25 02:39:26 , processing array  2
2007-04-25 02:39:26 , processing array  3
2007-04-25 02:39:27 , processing array  4
done.
Quality control after preprocessing ...
done.
```

Users can easily customize the processing parameters. For example, if the user want to do "quantile" normalization instead of "rsn" normalization, the user can run the following code. For more details, please read the help document of `lumiExpresso` function.

```
> lumi.N.Q <- lumiExpresso(example.lumi, normalize.param = list(method = "quantile"))

Variance Stabilizing Transform: vst
normalization: quantile

Variance stabilizing ...
2007-04-25 02:39:27 , processing array  1
2007-04-25 02:39:27 , processing array  2
2007-04-25 02:39:27 , processing array  3
2007-04-25 02:39:27 , processing array  4
done.
Normalizing ...
done.
Quality control after preprocessing ...
done.
```

4 Gene annotation

Illumina microarray provides the TargetID or the ProbeID to identify the measurements. The TargetID is used as a public identifier by Illumina and is supposed to be stable. The problem of the TargetID is that it can correspond to several different probes, which are supposed to match the same gene. Due to the

binding affinity difference or alternative splicing, the probes corresponding the the sample TargetID may have quite different expression levels and patterns. If we use TargetID to identify the measurements, then we cannot differentiate the difference between these probes. Another problem of using the TargetID is that the mapping between the TargetID and probes could be changed with our better understanding of the gene. Moreover, the TargetID used by Illumina microarray is not consistent among different versions of arrays. For instance, the same 50mer sequence has two different TargetIDs used by Illumina: `"GI_21070949-S"` in the `Mouse_Ref-8_V1` chip and `"scl022190.1_154-S"` in the `Mouse-6_V1` chip. This causes difficulties when combining clinical microarray data collected over time using different versions of the chips.

In order to get unique mapping between microarray measurements and probes, using ProbeID is preferred. However, the ProbeID of Illumina is not stable. It is changing between different versions, even between different batches of Illumina microarrays. To solve these problems, we designed a nucleotide universal identifier (nuID), which encodes the 50mer oligonucleotide sequence and contains error checking and self-identification code. By using nuID, all the problems mentioned above can be easily solved. For details, please read [2].

4.1 Examples of nuID

```
> ## provide an arbitrary nucleotide sequence as an example
> seq <- 'ACGTAAATTTCAGTTTAAAACCCCCCG'
> ## create a nuID for it
> id <- seq2id(seq)
> print(id)

[1] "YGwP0vwBVW"
```

The original nucleotide sequence can be easily recovered by `id2seq`

```
> id2seq(id)

[1] "ACGTAAATTTCAGTTTAAAACCCCCCG"
```

The nuID is self-identifiable. `is.nuID` can check the sequence is nuID or not. A real nuID

```
> is.nuID(id)

[1] TRUE
```

An random sequence

```
> is.nuID("adfqeqe")

[1] FALSE
```

4.2 Illumina microarray annotation package

Because all the Illumina microarrays use 50-mers, by using the nuID universal identifier, we are able to build one annotation database for different versions of the human (or other species) chips. Moreover, the nuID can be directly converted to the probe sequence, and used to get the most updated refSeq matches and annotations. Annotation packages indexed by nuID for different Illumina expression chips can be downloaded from Bioconductor.

The Illumina annotation packages are produced by using *AnnBuilder* with small modification. As a result, the format of the package is the same as Affymetrix annotation package, lots of packages designed for Affymetrix can also be used for Illumina annotation package. The mappings between TargetID to nuID and ProbeID to nuID are also included in the Illumina annotation packages. Thus, we can easily mapping between the nuID and TargetID or ProbeID.

Here is some examples:

```
> ## load lumi annotation package
> lib <- 'example.lumi'                # Huamn lumi annotation package
> if(require(GO) & require(annotate) & require(lib, character.only=TRUE)) {
+         GOId <- 'GO:0004816'                  # asparagine-tRNA ligase activity
+         probe <- lookUp(GOId, lib, 'GO2ALLPROBES')
+         # probes under 'GO:0004816'  category
+         print(probe)
+ }

> # specify a nuID
> nuId <- 'WVUU7XyNw3ucXzwdEk'
> if (require(annotate) & require(lib, character.only=TRUE)) {
+        # get the gene symbol of nuId
+        getSYMBOL(nuId, lib)
+ }
```

Mapping from nuID to TargetID

```
> nuId <- "WVUU7XyNw3ucXzwdEk"
> nuID2targetID(nuId, lib = "lumiHumanV1")

[1] "lumiHumanV1 annotation library is required!"
```

Mapping from TargetID to nuID

```
> targetID <- "GI_7262387-S"
> targetID2nuID(targetID, lib = "lumiHumanV1")

[1] "lumiHumanV1 annotation library is required!"
```

4.3 Transfer Illumina identifier annotated data into nuID annotated

As the annotation packages include the mappings between TargetID to nuID and ProbeID to nuID. We can easily map the targetID (or Probe Id) to nuID. The function can automatically check whether targetID or Probe Id was used

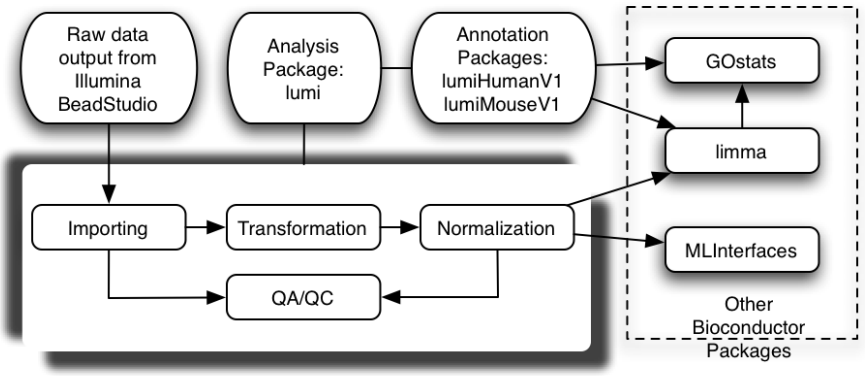


Figure 18: Flow chart of the use case

in the text data file, and convert them as nuID. Function `addNuId2lumi` can transfer a TargetID or Probe Id indexed **LumiBatch** object as an nuID indexed **LumiBatch** object. And the mapping between the nuID and TargetID is kept in the featureData of the **LumiBatch** object. If a **LumiBatch** object has already been nuID indexed, the function will do nothing.

```
> if (require(lumiHumanV1)) {
+     lumi.N <- addNuId2lumi(lumi.N, lib = "lumiHumanV1")
+ }
```

The **LumiBatch** object can also be directly transferred as nuID indexed at the very beginning of inputting data using `lumiR`. For example:

```
> ## load the data
>  example.lumi <- lumiR(fileName, lib='lumiHumanV1')          # Not run
```

5 A use case: from raw data to functional analysis

Figure 18 shows the data processing flow chart of the use case. Since the classes in *lumi* package are inherited from class **ExpressionSet**, packages and functions compatible with class **ExpressionSet** or accepting matrix as input all can be used for *lumi* results. Here we just give two examples: using *limma* to identify differentiated genes and using *GOstats* to annotate the significant genes.

We use the Barnes data set [3] as an example, which has be created as a Bioconductor experiment data package *lumiBarnes*. The Barnes data set measured a dilution series of two human tis-sues, blood and placenta. It includes six samples with the titration ratio of blood and placenta as 100:0, 95:5, 75:25, 50:50, 25:75 and 0:100. The samples were hybridized on HumanRef-8 BeadChip (Illumina, Inc) in duplicate. We select samples with titration ratio, 100:0 and 95:5 (each has two technique replicates) in this data set to evaluate the detection of differential expressions.

5.1 Preprocess the Illumina data

```
> library(lumi)
> ## specify the file name
> # fileName <- 'Barnes_gene_profile.txt'  #  Not run
> ## load the data
> # example.lumi <- lumiR(fileName, lib='lumiHumanV1')         # Not run

> ## load saved data
> load(example.lumi)
> ## sumary of the daa
> example.lumi
> ## summary of quality control information
> summary(example.lumi, 'QC')

> ## preprocessing and quality control after normalization
> lumi.N.Q <- lumiExpresso(example.lumi, QC.evaluation=TRUE)
> ## summary of quality control information after preprocessing
> summary(lumi.N.Q, 'QC')
```

5.2 Identify differentiate genes

Identify the differentiated genes based on moderated t-test using *limma*.

Retrieve the normalized data

```
> dataMatrix <- exprs(lumi.N)
```

To speed up the processing and reduce false positives, remove the unexpressed genes

```
> presentCount <- pData(featureData(lumi.N))$presentCount
> selDataMatrix <- dataMatrix[presentCount > 0, ]
> selProbe <- rownames(selDataMatrix)

> ## Specify the sample type
> sampleType <- c('100:0', '95:5', '100:0', '95:5')
> if (require(limma)) {
+         ## compare '95:5' and '100:0'
+         design <- model.matrix(~ factor(sampleType))
+         colnames(design) <- c('100:0', '95:5-100:0')
+         fit <- lmFit(selDataMatrix, design)
+         fit <- eBayes(fit)
+       ## Add gene symbols to gene properties
+        if (require(lumiHumanV1) & require(annotate)) {
+             geneSymbol <- getSYMBOL(fit$genes$ID, 'lumiHumanV1')
+              fit$genes <- data.frame(fit$genes, geneSymbol=geneSymbol)
+        }
+         ## print the top 10 genes
+         topTable(fit, coef='95:5-100:0', adjust='fdr', number=10)
+
+         ## get significant gene list with FDR adjusted p.values less than 0.01
```

```
+         p.adj <- p.adjust(fit$p.value[,2])
+         sigGene.adj <- selProbe[ p.adj < 0.01]
+         ## without FDR adjustment
+         sigGene <- selProbe[ fit$p.value[,2] < 0.001]
+
+ }
```

```
                    ID    logFC        t      P.Value    adj.P.Val        B
1315 ol_iQkR.siio.kvH6k 5.174566 81.62537 4.760969e-16 1.726889e-12 26.90278
3666 EY761AIG0XSLUfnuyc 5.630769 80.45131 5.541115e-16 1.726889e-12 26.78272
4484 WlCoF7taz2MeYf3l6I 4.335803 73.99741 1.330196e-15 2.763703e-12 26.07156
59   NSjRKdq2eSGf0ur4aQ 4.170903 67.96571 3.239722e-15 3.533437e-12 25.31856
3002 QaYYojcJJvVElV3I98 3.882748 67.81947 3.313596e-15 3.533437e-12 25.29911
1666 6QNThLQLd61eU6IXhI 4.074785 67.65032 3.401351e-15 3.533437e-12 25.27655
4553 TueuSaiCheWBxB6B18 4.217038 66.42233 4.120244e-15 3.668783e-12 25.11031
1207 uioiKiIlzFXx8k5EC4 3.959484 64.30547 5.782757e-15 4.505491e-12 24.81338
3884 rSU1F9I7txuZ3lPQdo 3.725620 62.14171 8.273318e-15 5.329142e-12 24.49557
3859 Q.oCSr13l5wQlRuhS0 3.843102 61.94673 8.549882e-15 5.329142e-12 24.46618
```

Based on the significant genes identified using *limma* or t-test, we can do further analysis, like GO analysis (*GOstats* package) and machine learning (*MLInterface* package). Next, we will use GO analysis as an example.

5.3 Gene Ontology analysis

Based on the significant genes identified using *limma* or t-test, we can further do Gene Ontology annotation. We can use package *GOstats* to do the analysis.

Do Hypergeometric test of Gene Ontology based on the significant gene list (for e. Table 1 shows the significant GO terms of Molecular Function with p-value less than 0.01. Here only show the significant GO terms of BP (Biological Process). For other GO categories MF(Molecular Function) and CC (Cellular Component), it just follows the same procedure.

```
> if (require(GOstats) & require(lumiHumanV1)) {
+
+       ## Get the locuslink Id of the gene
+       sigLL <- unique(unlist(mget(sigGene, env=lumiHumanV1LOCUSID, ifnotfound=NA)))
+       sigLL <- as.character(sigLL[!is.na(sigLL)])
+       params <- new("GOHyperGParams",
+               geneIds= sigLL,
+               annotation="lumiHumanV1",
+               ontology="BP",
+               pvalueCutoff= 0.01,
+               conditional=FALSE,
+               testDirection="over")
+
+       hgOver <- hyperGTest(params)
+
+       ## Get the p-values of the test
+       gGhyp.pv <- pvalues(hgOver)
+
```

```
+       ## select the Go terms with p-value less than 0.001
+       sigGO.ID <- names(gGhyp.pv[gGhyp.pv < 0.001])
+
+       ## Here only show the significant GO terms of BP (Molecular Function)
+       ##         For other categories, just follow the same procedure.
+       sigGO.Term <- getGOTerm(sigGO.ID)[["BP"]]
+ }
```

	GO ID	Term	p-value	Significant Genes No.	Total Genes No.
1	GO:0009611	response to wound...	8.4244e-06	42	443
2	GO:0006955	immune response	8.8296e-06	68	859
3	GO:0006952	defense response	1.7525e-05	72	945
4	GO:0006950	response to stres...	1.9132e-05	81	1103
5	GO:0009607	response to bioti...	5.0811e-05	72	976
6	GO:0009613	response to pest,...	7.2813e-05	45	533
7	GO:0006954	inflammatory resp...	0.00025402	25	250
8	GO:0009605	response to exter...	0.00026005	46	580
9	GO:0051707	response to other...	0.00040553	45	575
10	GO:0051674	localization of c...	0.00082563	30	348
11	GO:0006928	cell motility	0.00082563	30	348
12	GO:0040011	locomotion	0.00099205	30	352

Table 1: GO terms, p-values and counts.

6 Reference

1. Lin, S.M., Du, P., Kibbe, W.A., "Model-based Variance-stabilizing Transformation for Illumina Mi-croarray Data", submitted

2. Du, P., Kibbe, W.A. and Lin, S.M., "nuID: A universal naming schema of oligonucleotides for Illumina, Affymetrix, and other microarrays", submitted.

3. Barnes, M., Freudenberg, J., Thompson, S., Aronow, B. and Pav-lidis, P. (2005) "Experimental comparison and cross-validation of the Affymetrix and Illumina gene expression analysis platforms", Nucleic Acids Res, 33, 5914-5923.