

An end to end workflow for differential gene expression using Affymetrix microarrays

Bernd Klaus¹ and Stefanie Reisenauer²

¹EMBL Heidelberg, Meyerhofstrasse 1, 69117 Heidelberg, Germany, bernd.klaus@embl.de

²EMBL Heidelberg, Meyerhofstrasse 1, 69117 Heidelberg, Germany, stefanie.reisenauer@embl.de

Abstract In this article, we walk through an end-to-end Affymetrix microarray differential expression workflow using Bioconductor packages. This workflow is directly applicable to current “Gene” type arrays, e.g. the HuGene or MoGene arrays, but can easily be adapted to similar platforms. The data re-analyzed is a typical clinical microarray data set that compares inflamed and non-inflamed colon tissue in two disease subtypes. For each disease, the differential gene expression between inflamed- and non-inflamed colon tissue was analyzed. We will start from the raw data CEL files, show how to import them into a Bioconductor ExpressionSet, perform quality control and normalization and finally differential gene expression (DE) analysis, followed by some enrichment analysis.

Keywords

Microarray, Gene Expression

R version: R version 3.5.0 (2018-04-23) **Bioconductor version:** 3.7 **Package version:** 1.0.3

Amendments from Version 1

The individual workflow steps now contain more detailed discussions of the code. We also try to explain the statistics behind the individual steps carefully and in a non-technical way. Where appropriate, we provide suitable references and weblinks.

In general, the new version should be much more accessible to beginners in R / Bioconductor with a working knowledge in statistics.

Furthermore, we want to highlight the following specific changes:

- The workflow has been wrapped into an R package “maEndToEnd” and the package dependencies are now grouped by keywords.
- Relative Log Expression (RLE) analysis has been implemented as another data quality control step. Combined with clustering heatmaps, it can be used as a basic tool to detect low quality arrays. A section on how to interpret these two kinds of plots has also been added.
- The method for intensity-based filtering of genes prior to the differential expression analysis has been changed. We now filter using a more intuitive, visually chosen cutoff instead of fitting a normal distribution and setting the 5 % quantile as a threshold.
- The rather theoretical section on the mathematical background of linear regression models has been removed.
- Instead, we explain the principles of differential expression analysis using a specific gene. We fit the linear model to this gene, explain its rationale and compare it to the standard t-test.
- A section on the use of False Discovery Rate (FDR) control in multiple testing problems has been added: We illustrate the benefits of FDR control using p-values from the data at hand.

Introduction

In this article we introduce a complete workflow for a typical (Affymetrix) microarray analysis. Data import, preprocessing, differential expression and enrichment analysis are discussed. The data set used [1] is from a paper studying the differences in gene expression in inflamed- and non-inflamed tissue. 14 Patients suffering from Ulcerative colitis (UC) and 15 patients with Crohn’s disease (CD) were tested, and from each patient inflamed- and non-inflamed colonic mucosa tissue was obtained via a biopsy. This is a typical clinical data set consisting of 58 arrays in total. Our aim is to analyze differential expression (DE) between the tissues. Our results show a substantial overlap with the results of the original paper.

Required packages and other preparations

The maEndToEnd package can currently be loaded from GitHub and will be submitted to Bioconductor soon.

```
#In order to download the package from GitHub, we need the "install_github"
#function from the "remotes" packages. We download the latest developer
#version of "remotes" from GitHub as well with the devtools::install_github
#function; note that this is necessary as the current "remotes" version on
#CRAN doesn't currently allow us to correctly download the "maEndToEnd" package:

install.packages("devtools")
library(devtools)

devtools::install_github("r-lib/remotes")
library(remotes)
packageVersion("remotes") #has to be 1.1.1.9000 or later

remotes::install_github("Steffireise/maEndToEnd", ref="master")

suppressPackageStartupMessages({
  library("maEndToEnd")
})
```

```

#General Bioconductor packages
library(Biobase)
library(oligoClasses)

#Annotation and data import packages
library(ArrayExpress)
library(pd.hugene.1.0.st.v1)
library(hugene10sttranscriptcluster.db)

#Quality control and pre-processing packages
library(oligo)
library(arrayQualityMetrics)

#Analysis and statistics packages
library(limma)
library(topGO)
library(ReactomePA)
library(clusterProfiler)

#Plotting and color options packages
library(gplots)
library(ggplot2)
library(geneplotter)
library(RColorBrewer)
library(pheatmap)

#Formatting/documentation packages
library(rmarkdown)
library(BiocStyle)
library(dplyr)
library(tidyr)

#Helpers:
library(stringr)
library(matrixStats)
library(genefilter)
library(openxlsx)
library(devtools)

```

Download of the raw data from ArrayExpress

The first step of the analysis is to download the raw data CEL files. These files are produced by the array scanner software and contain the probe intensities measured. The data we use have been deposited at ArrayExpress and have the accession code **E-MTAB-2967**.

We will store these files in the directory **raw_data_dir** which defaults to the subdirectory **rawDataMAWorkflow** of the current working directory:

```

raw_data_dir <- file.path(getwd(), "rawDataMAWorkflow")

if(!dir.exists(raw_data_dir)){
  dir.create(raw_data_dir)
}

```

Each ArrayExpress data set has a landing page summarizing the data set, and we use the *ArrayExpress* Bioconductor package to obtain the ftp links to the raw data files (Data from Palmieri et. al. on ArrayExpress).

Information stored in ArrayExpress

Each dataset at ArrayExpress is stored according to the MAGE-TAB (MicroArray Gene Expression Tabular) specifications as a collection of tables bundled with the raw data. The MAGE-TAB format specifies up to five different types of files, namely the Investigation Description Format (IDF), the Array Design Format (ADF), the Sample and Data Relationship Format (SDRF), the raw data files and the processed data files.

Other than the raw data files, the IDF and the SDRF file are important for us. The IDF file contains top level information about the experiment including title, description, submitter contact details and protocols. The SDRF file contains essential information on the experimental samples, e.g. the experimental group(s) they belong to.

Download of the raw data and the annotation data

With the code below, we download the raw data from ArrayExpress [2] by using the `getAE`-function. The data are saved in the `raw_data_dir` created above. The names of the downloaded files are returned as a list.

```
anno_AE <- getAE("E-MTAB-2967", path=raw_data_dir, type="raw")
```

We now import the SDRF file with the `read.delim` function from the raw data folder in order to obtain the sample annotation.

The raw data consists of one CEL file per sample (see below) and we use the CEL file names, thus the sample identifiers, as row names for the imported SDRF data.

These names are given in a column named `Array.Data.File` in the SDRF table. We turn the SDRF table into an `AnnotatedDataFrame` from the *Biobase* package that we will need later to create an `ExpressionSet` for our data [3].

```
sdrf_location <- file.path(raw_data_dir, "E-MTAB-2967.sdrf.txt")
SDRF <- read.delim(sdrf_location)

rownames(SDRF) <- SDRF$Array.Data.File
SDRF <- AnnotatedDataFrame(SDRF)
```

Before we move on to the actual raw data import, we will briefly introduce the `ExpressionSet` class contained in the *Biobase* package. It is commonly used to store microarray data in Bioconductor.

Bioconductor ExpressionSets

Genomic data can be very complex, usually consisting of a number of different bits and pieces, e.g. information on the experimental samples, annotation of genomic features measured as well as the experimental data itself. In Bioconductor, the approach is taken that these pieces should be stored in a single structure to easily manage the data.

The package *Biobase* contains standardized data structures to represent genomic data. The `ExpressionSet` class is designed to combine several different sources of information (i.e. as contained in the various MAGE-TAB files) into a single convenient structure. An `ExpressionSet` can be manipulated (e.g., subsetted, copied), and is the input to or output of many Bioconductor functions.

The data in an `ExpressionSet` consist of:

- **assayData**: Expression data from microarray experiments.
- **metaData**
 - **phenoData**: A description of the samples in the experiment
 - **featureData**: metadata about the features on the chip or technology used for the experiment
 - further annotations for the features, for example gene annotations from biomedical databases (annotation).
- **experimentData**: A flexible structure to describe the experiment.

The `ExpressionSet` class coordinates all of these data, so that one does not have to worry about the details. However, some constraints have to be met. In particular, the rownames of the `phenoData` (which holds the content of the SDRF file) have to match the column names of the assay data (as they represent the sample identifiers), while the row names of the assay data have to match the row names of the `featureData` (as they represent the feature identifiers). This is illustrated in Figure 1.

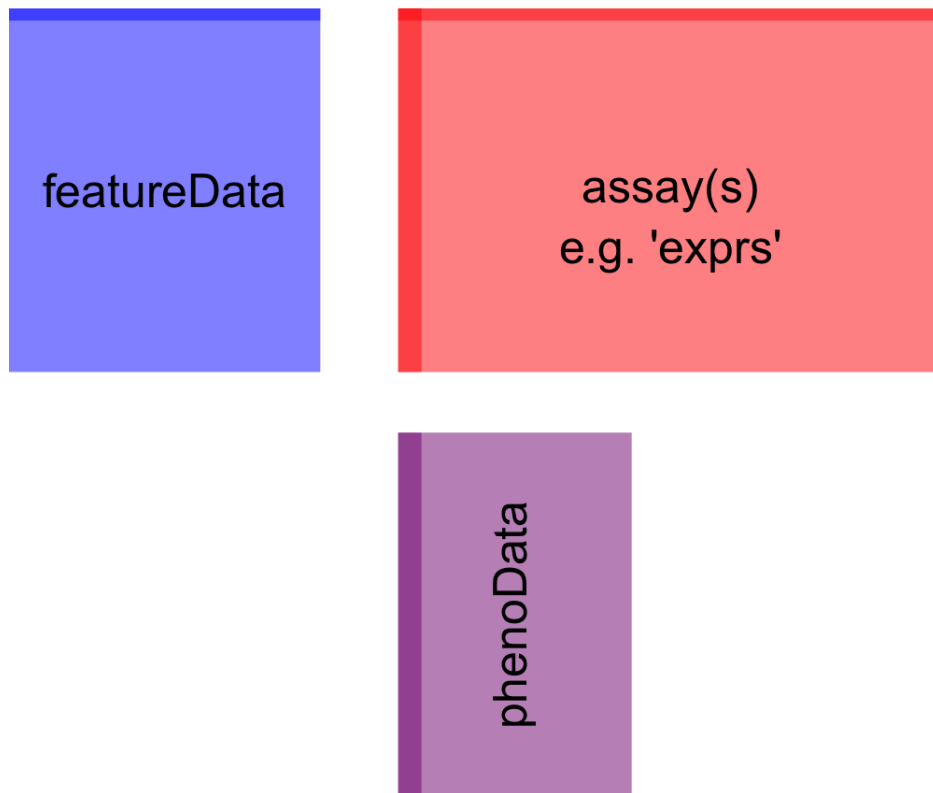


Figure 1. Structure of Bioconductor's ExpressionSet class.

You can use the functions `pData` and `fData` to extract the sample and feature annotation, respectively, from an `ExpressionSet`. The function `exprs` will return the expression data itself as a matrix.

Import of the raw microarray data

In the first step of the analysis, we create the Expression Set object `raw_data`, which contains array data, pheno data (from the SDRF file) as well as the information of the chip annotation package used.

The analysis of Affymetrix arrays starts with CEL files. These are the result of the processing of the raw image files using the Affymetrix software and contain estimated probe intensity values. Each CEL file additionally contains some metadata, such as a chip identifier.

We use the function `read.celfiles` from the *oligo* package [4] to import the files. This automatically creates an `ExpressionSet`, fills the sections “array data” with the data from the CEL files and uses the correct chip annotation package, in this case *pd.hugene.1.0.st.v1* (the chip-type is also stored in the .CEL files).

Furthermore, we specify our `AnnotatedDataFrame` created earlier from the SDRF file as `phenoData`. Thus, we have to make sure that we import the CEL files in the order that corresponds to the SDRF table — to enforce this, we use the column `Array.Data.File` of the SDRF table as the `filenames` argument.

Finally, we check whether the object created is valid (e.g. sample names match between the different tables).

```
raw_data <- oligo::read.celfiles(filenames = file.path(raw_data_dir,
                                                    SDRF$Array.Data.File),
                               verbose = FALSE, phenoData = SDRF)
validObject(raw_data)
```

We now inspect the raw data a bit.

The `pData` function of the Biobase package directly accesses the `phenoData` in the `ExpressionSet` `raw_data`. With the `head()` function, we can view the first six lines of the table. We have a look at the columns included and retain only those columns that are related to the experimental factors of interest.

```
head(Biobase::pData(raw_data))
```

	Source.Name	Characteristics.individual.	Characteristics.organism.		
164_I_.CEL	164_I	164	Homo sapiens		
164_II.CEL	164_II	164	Homo sapiens		
183_I.CEL	183_I	183	Homo sapiens		
183_II.CEL	183_II	183	Homo sapiens		
2114_I.CEL	2114_I	2114	Homo sapiens		
2114_II.CEL	2114_II	2114	Homo sapiens		
	Characteristics.disease.	Characteristics.organism.part.			
164_I_.CEL	Crohn's disease	colon			
164_II.CEL	Crohn's disease	colon			
183_I.CEL	Crohn's disease	colon			
183_II.CEL	Crohn's disease	colon			
2114_I.CEL	Crohn's disease	colon			
2114_II.CEL	Crohn's disease	colon			
	Characteristics.phenotype.	Material.Type	Protocol.REF		
164_I_.CEL	non-inflamed colonic mucosa	organism part	P-MTAB-41361		
164_II.CEL	inflamed colonic mucosa	organism part	P-MTAB-41361		
183_I.CEL	non-inflamed colonic mucosa	organism part	P-MTAB-41361		
183_II.CEL	inflamed colonic mucosa	organism part	P-MTAB-41361		
2114_I.CEL	non-inflamed colonic mucosa	organism part	P-MTAB-41361		
2114_II.CEL	inflamed colonic mucosa	organism part	P-MTAB-41361		
	Protocol.REF.1	Extract.Name	Protocol.REF.2	Labeled.Extract.Name	
164_I_.CEL	P-MTAB-41363	164_I	P-MTAB-41364	164_I:Biotin	
164_II.CEL	P-MTAB-41363	164_II	P-MTAB-41364	164_II:Biotin	
183_I.CEL	P-MTAB-41363	183_I	P-MTAB-41364	183_I:Biotin	
183_II.CEL	P-MTAB-41363	183_II	P-MTAB-41364	183_II:Biotin	
2114_I.CEL	P-MTAB-41363	2114_I	P-MTAB-41364	2114_I:Biotin	
2114_II.CEL	P-MTAB-41363	2114_II	P-MTAB-41364	2114_II:Biotin	
	Label	Protocol.REF.3	Assay.Name	Technology.Type	Array.Design.REF
164_I_.CEL	biotin	P-MTAB-41366	164_I_	array assay	A-AFFY-141
164_II.CEL	biotin	P-MTAB-41366	164_II	array assay	A-AFFY-141
183_I.CEL	biotin	P-MTAB-41366	183_I	array assay	A-AFFY-141
183_II.CEL	biotin	P-MTAB-41366	183_II	array assay	A-AFFY-141
2114_I.CEL	biotin	P-MTAB-41366	2114_I	array assay	A-AFFY-141
2114_II.CEL	biotin	P-MTAB-41366	2114_II	array assay	A-AFFY-141
	Term.Source.REF	Protocol.REF.4	Array.Data.File		
164_I_.CEL	ArrayExpress	P-MTAB-41367	164_I_.CEL		
164_II.CEL	ArrayExpress	P-MTAB-41367	164_II.CEL		
183_I.CEL	ArrayExpress	P-MTAB-41367	183_I.CEL		
183_II.CEL	ArrayExpress	P-MTAB-41367	183_II.CEL		
2114_I.CEL	ArrayExpress	P-MTAB-41367	2114_I.CEL		
2114_II.CEL	ArrayExpress	P-MTAB-41367	2114_II.CEL		
164_I_.CEL	ftp://ftp.ebi.ac.uk/pub/databases/microarray/data/experiment/MTAB/				
164_II.CEL	ftp://ftp.ebi.ac.uk/pub/databases/microarray/data/experiment/MTAB/				
183_I.CEL	ftp://ftp.ebi.ac.uk/pub/databases/microarray/data/experiment/MTAB/				
183_II.CEL	ftp://ftp.ebi.ac.uk/pub/databases/microarray/data/experiment/MTAB/				
2114_I.CEL	ftp://ftp.ebi.ac.uk/pub/databases/microarray/data/experiment/MTAB/				
2114_II.CEL	ftp://ftp.ebi.ac.uk/pub/databases/microarray/data/experiment/MTAB/				
	Factor.Value.disease.	Factor.Value.phenotype.			
164_I_.CEL	Crohn's disease	non-inflamed colonic mucosa			
164_II.CEL	Crohn's disease	inflamed colonic mucosa			
183_I.CEL	Crohn's disease	non-inflamed colonic mucosa			
183_II.CEL	Crohn's disease	inflamed colonic mucosa			
2114_I.CEL	Crohn's disease	non-inflamed colonic mucosa			
2114_II.CEL	Crohn's disease	inflamed colonic mucosa			

```
stopifnot(validObject(raw_data))
```

The columns of interest for us are the following:

- identifiers of the individuals, i.e. columns "Source.Name", "Characteristics.individual."
- disease of the individual, i.e. "Factor.Value.disease."
- mucosa type, i.e. "Factor.Value.phenotype."

We now subselect the corresponding columns:

```
Biobase::pData(raw_data) <- Biobase::pData(raw_data)[, c("Source.Name",
  "Characteristics.individual.",
  "Factor.Value.disease.",
  "Factor.Value.phenotype.")]
```

Quality control of the raw data

The first step after the initial data import is the quality control of the data. Here we check for outliers and try to see whether the data clusters as expected, e.g. by the experimental conditions. The expression intensity values are in the assayData sub-object “exprs” and can be accessed by the `exprs(raw_data)` function. The rows represent the microarray probes, i.e. the single DNA locations on the chip, while the columns represent one microarray, i.e. a sample of inflamed- and non-inflamed tissue of every patient, respectively.

```
Biobase::exprs(raw_data)[1:5, 1:5]
```

	164_I.CEL	164_II.CEL	183_I.CEL	183_II.CEL	2114_I.CEL
1	4496	5310	4492	4511	2872
2	181	280	137	101	91
3	4556	5104	4379	4608	2972
4	167	217	99	79	82
5	89	110	69	58	47

For quality control, we take the `log2` of `Biobase::exprs(raw_data)`, as expression data is commonly analyzed on a logarithmic scale.

We then perform a principal component analysis (PCA) and plot it (Figure 2). Every point in the plot represents one sample, with the colour indicating the mucosa type (inflamed vs non-inflamed) and the shape indicating the disease (UC or CD).

We also represent the probe intensities via a boxplot graph with one box per individual microarray. (Figure 3). Note that the `oligo::boxplot` function, i.e. the boxplot function of the `oligo` package, can take expression sets as argument. It accesses the expression data and performs a `log2`-transformation by default. We therefore can use `raw_data` as argument here.

```
exp_raw <- log2(Biobase::exprs(raw_data))
PCA_raw <- prcomp(t(exp_raw), scale. = FALSE)

percentVar <- round(100*PCA_raw$sdev^2/sum(PCA_raw$sdev^2),1)
sd_ratio <- sqrt(percentVar[2] / percentVar[1])

dataGG <- data.frame(PC1 = PCA_raw$x[,1], PC2 = PCA_raw$x[,2],
  Disease = pData(raw_data)$Factor.Value.disease.,
  Phenotype = pData(raw_data)$Factor.Value.phenotype.,
  Individual = pData(raw_data)$
    Characteristics.individual.)

(qplot(PC1, PC2, data = dataGG,
  color = Phenotype,
  shape = Disease,
  main = "PCA plot of the raw data (log-transformed)",
  size = I(2),
  xlab = paste0("PC1, VarExp: ", percentVar[1], "%"),
  ylab = paste0("PC2, VarExp: ", percentVar[2], "%"),
  asp = sd_ratio)
+ scale_colour_brewer(palette = "Set2"))
```

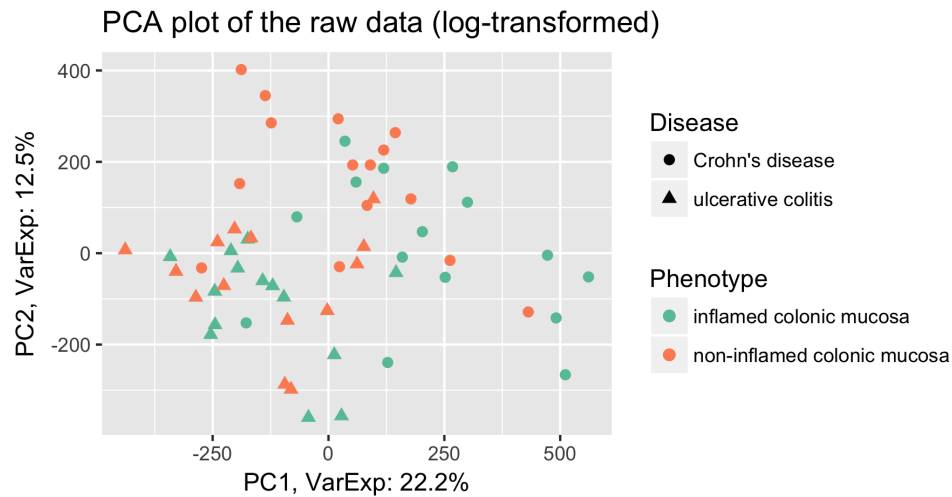


Figure 2. PCA plot of the log-transformed raw data.

```
oligo::boxplot(raw_data, target="core",
               main="Boxplot of log2-intensities for the raw data")
```

Boxplot of log2-intensities for the raw data

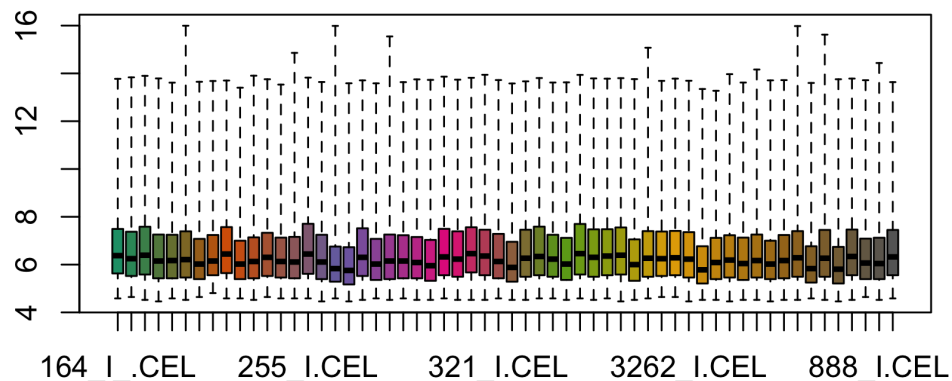


Figure 3. Intensity boxplots of the log2-transformed raw data.

The PCA plot (Figure 2, performed on the log-intensity scale) of the raw data shows that the first principal component differentiates between the diseases. This means that the disease type is a major driver of gene expression differences.

This might hinder our analysis, as we want to analyze the differential expression between inflamed- and non-inflamed tissues, independently of the disease a person suffers from.

When looking at the boxplot (Figure 3), we see that the intensity distributions of the individual arrays are quite different, indicating the need for an appropriate normalization, which we will discuss next.

Until now, we have only performed a very basic quality control; for more elaborate quality control plots, one can use the package *arrayQualityMetrics* [5]. The package produces an html report, containing the quality control plots together with a description of their aims and an identification of possible outliers. We don't discuss this tool in detail here, but simply provide the code below, which creates a report for our raw data.

```
arrayQualityMetrics(expressionset = raw_data,
                    outdir = "Report_for_Palmieri_raw",
                    force = TRUE, do.logtransform = TRUE,
                    intgroup = c("Factor.Value.disease.", "Factor.Value.phenotype."))
```

The report will be written into directory 'Report_for_Palmieri_raw'.

Background adjustment, calibration, summarization and annotation

Background adjustment

After the initial import and quality assessment, the next step in processing of microarray data is background adjustment. This is essential because a part of the measured probe intensities are due to non-specific hybridization and the noise in the optical detection system. Therefore, observed intensities need to be adjusted to give accurate measurements of specific hybridization.

Across-array normalization (calibration)

Without proper normalization across arrays, it is impossible to compare measurements from different array hybridizations due to many obscuring sources of variation. These include different efficiencies of reverse transcription, labeling or hybridization reactions, physical problems with the arrays, reagent batch effects, and laboratory conditions.

Summarization

After normalization, summarization is needed because on the Affymetrix platform, transcripts are represented by multiple probes, that is multiple locations on the array. For each gene, the background-adjusted and normalized intensities of all probes need to be summarized into one quantity that estimates an amount proportional to the amount of RNA transcript.

After the summarization step, the summarized data can be annotated with various information, e.g. gene symbols and ENSEMBL gene identifiers. There is an annotation database available from Bioconductor for our platform, namely the package *hugene10sttranscriptcluster.db*.

You can view its content like this

```
head(ls("package:hugene10sttranscriptcluster.db"))
```

```
[1] "hugene10sttranscriptcluster"
[2] "hugene10sttranscriptcluster_dbconn"
[3] "hugene10sttranscriptcluster_dbfile"
[4] "hugene10sttranscriptcluster_dbInfo"
[5] "hugene10sttranscriptcluster_dbschema"
[6] "hugene10sttranscriptcluster.db"
```

Additional information is available from the reference manual of the package. Essentially, the package provides a mapping from the transcript cluster identifiers to the various annotation data.

Old and new “probesets” of Affymetrix microarrays

Traditionally, Affymetrix arrays (the so-called 3’ IVT arrays) were probeset based: a certain fixed group of probes were part of a probeset which represented a certain gene or transcript (note however, that a gene can be represented by multiple probesets).

The more recent “Gene” and “Exon” Affymetrix arrays are exon based and hence there are two levels of summarization to get to the gene level. The “probeset” summarization leads to the exon level. The gene/transcript level is given by “transcript clusters”. Hence, the appropriate annotation package for our chip type is called *hugene10sttranscriptcluster.db*.

“Gene” arrays were created as affordable versions of the “Exon” arrays, by only taking the “good” probes from the Exon array. Initially on the exon array, at least four probes were part of one “Exon”. With the thinned out “Gene” array, a lot of probesets were made up of three or fewer probes. Thus, a summarization on the probeset / exon level is not recommended for “Gene” arrays but nonetheless possible by using the *hugene10stprobeset.db* annotation package.

Note that furthermore, there are also no longer designated match/mismatch probes present on “Gene” and “Exon” type chips. The mismatch probe was initially intended as base-level for background correction, but didn’t prevail due to more elaborate background correction techniques that didn’t require a mismatch probe.

One-go preprocessing in oligo

The package *oligo* allows us to perform background correction, normalization and summarization in one single step using a deconvolution method for background correction, quantile normalization and the RMA (robust multichip average) algorithm for summarization.

This series of steps as a whole is commonly referred to as RMA algorithm, although strictly speaking RMA is merely a summarization method [6, 7, 8].

Relative Log Expression data quality analysis

Before calibrating and evaluating the data, we want to perform another quality control procedure, namely Relative Log Expression (RLE), as described in the article by Gandolfo et al [9]. To this end, we first perform an RMA without prior normalization:

```
palmieri_eset <- oligo::rma(raw_data, target = "core", normalize = FALSE)
```

```
Background correcting
Calculating Expression
```

Further details on the RMA algorithm will be provided after RLE analysis, when the “full” RMA is carried out, including normalization.

The RLE is performed by calculating the median log2 intensity of every transcript across all arrays.

We do this by calculating the row medians of `exprs(palmieri_eset)`, as the transcripts are represented by the rows and the single microarrays by the columns.

Note that we don't have to apply the log2 manually, as the output data of the RMA function is in log2 scale by default.

We then subtract this transcript median intensity from every transcript intensity via the `sweep` function.

We then reshape the data into a format in which we can use to create a boxplot for each array, as before.

Note that the y-axis now displays for each microarray the deviation of expression intensity from the median expression of the respective single transcripts across arrays.

Boxes with a larger extension therefore indicate an unusually high deviation from the median in a lot of transcripts, suggesting that these arrays are different from most of the others in some way.

Boxes that are shifted in y-direction indicate a systematically higher or lower expression of the majority of transcripts in comparison to most of the other arrays. This could be caused by quality issues or batch effects.

Therefore, if shape and median of the single boxes vary too much from the bulk of the data, they should be inspected and potentially removed.

```
row_medians_assayData <-
  Biobase::rowMedians(as.matrix(Biobase::exprs(palmieri_eset)))

RLE_data <- sweep(Biobase::exprs(palmieri_eset), 1, row_medians_assayData)

RLE_data <- as.data.frame(RLE_data)
RLE_data_gathered <-
  tidyr::gather(RLE_data, patient_array, log2_expression_deviation)

ggplot2::ggplot(RLE_data_gathered, aes(patient_array,
                                       log2_expression_deviation)) +
  geom_boxplot(outlier.shape = NA) +
  ylim(c(-2, 2)) +
  theme(axis.text.x = element_text(colour = "aquamarine4",
                                    angle = 60, size = 6.5, hjust = 1 ,
                                    face = "bold"))
```

```
Warning: Removed 6302 rows containing non-finite values (stat_boxplot).
```

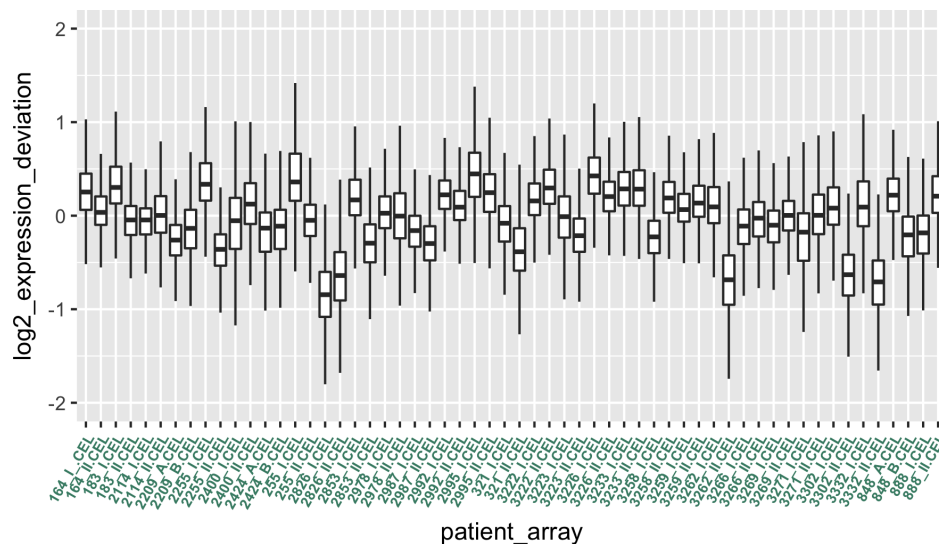


Figure 4. Boxplot for the RLE values

Inspecting the boxplot in Figure 4, 5 arrays could be considered as outliers: 2826_I, 2826_II, 3262_II, 3302_II and 3332_II are negatively y-shifted.

We will keep these samples in mind for heatmap cluster analysis later on in the workflow. Arrays that are confirmed to be outliers by heatmap analysis could be considered as outliers and be removed for subsequent analysis.

RMA calibration of the data

Now, we can apply the full RMA algorithm to our data in order to background-correct, normalize and summarize:

```
palmieri_eset_norm <- oligo::rma(raw_data, target = "core")
```

```
Background correcting
Normalizing
Calculating Expression
```

The parameter `target` defines the degree of summarization, the default option of which is “core”, using transcript clusters containing “safely” annotated genes. Other options for `target` include “extended” and “full”. For summarization on the exon level (not recommended for Gene arrays), one can use “probeset” as the target option. Although other methods for background correction and normalization exist, RMA is usually a good default choice. RMA shares information across arrays and uses the versatile quantile normalization method that will make the array intensity distributions match. However, it is preferable to apply it only after outliers have been removed. The quantile normalization algorithm used by RMA works by replacing values by the average of identically ranked (within a single chip) values across arrays. A more detailed description can be found on the Wikipedia page about it.

An alternative to quantile normalization would be the *vsr* algorithm, that performs background correction and normalization by robustly shifting and scaling intensity values within arrays before log-transforming them. This is less “severe” than quantile normalization [10].

Some mathematical background on normalization (calibration) and background correction

A generic model for the value of the intensity Y of a single probe on a microarray is given by

$$Y = B + \alpha \cdot S$$

where B is a random quantity due to background noise, usually composed of optical effects and non-specific binding, α is a gain factor, and S is the amount of measured specific binding. The signal S is considered a random variable as well and accounts for measurement error and probe effects. The measurement error is typically assumed to be multiplicative so we can write:

$$\log(S) = \theta + \varphi + \varepsilon$$

Here θ represents the logarithm of the true abundance, φ is a probe-specific effect, and ε accounts for the nonspecific error. This is the additive-multiplicative-error model for microarray data used by RMA and also the *vsn* algorithm [10]. The algorithms differ in the way that B is removed and an estimate of θ is obtained.

Quality assessment of the calibrated data

We now produce a clustering heatmap and another PCA plot using the calibrated data.

PCA analysis

First, we perform a PCA analysis of the calibrated data analogously to the one with the raw data:

```
exp_palmieri <- Biobase::exprs(palmieri_eset_norm)
PCA <- prcomp(t(exp_palmieri), scale = FALSE)

percentVar <- round(100*PCA$sdev^2/sum(PCA$sdev^2),1)
sd_ratio <- sqrt(percentVar[2] / percentVar[1])

dataGG <- data.frame(PC1 = PCA$x[,1], PC2 = PCA$x[,2],
                     Disease =
                       Biobase::pData(palmieri_eset_norm)$Factor.Value.disease.,
                     Phenotype =
                       Biobase::pData(palmieri_eset_norm)$Factor.Value.phenotype.)

(qplot(PC1, PC2, data = dataGG,
       color = Phenotype,
       shape = Disease,
       main = "PCA plot of the calibrated data",
       size = I(2),
       xlab = paste0("PC1, VarExp: ", percentVar[1], "%"),
       ylab = paste0("PC2, VarExp: ", percentVar[2], "%"),
       asp = sd_ratio)
+ scale_colour_brewer(palette = "Set2"))
```

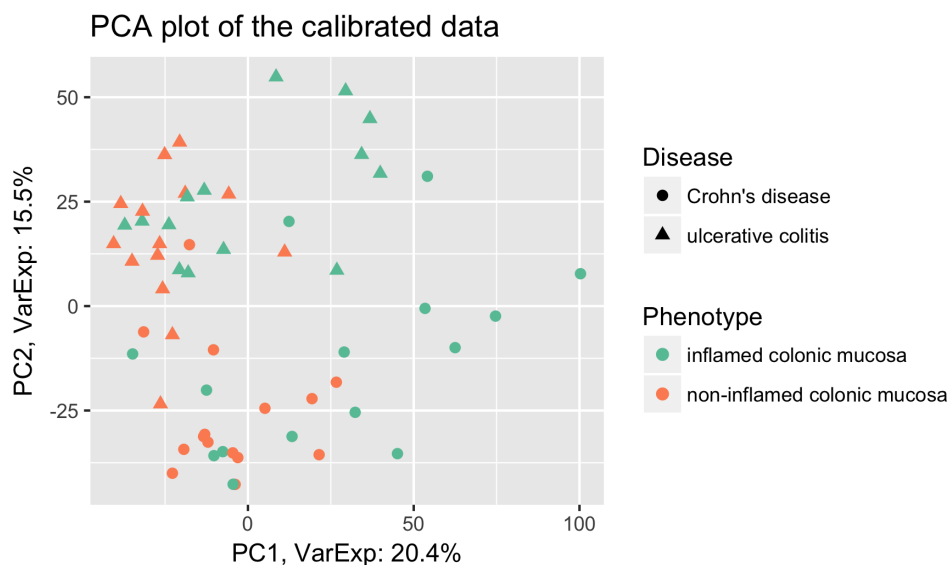


Figure 5. PCA plot of the calibrated data.

In comparison to the first PCA analysis before RMA (Figure 2, we see that now the first principal component separates between the tissues types (Figure 5). This indicates that now differential expression between the tissue types is the dominant source of variation. Not that the second principal components separates the diseases.

Heatmap clustering analysis

In order to display a heatmap of the sample-to-sample distances, we first compute the distances using the `dist` function. We need to transpose the expression values since the function computes the distances between the rows (i.e. genes in our case) by default. The default distance is the Euclidean one. However this can be changed and we choose the Manhattan distance here (it uses absolute distances along rectangular paths instead of squared distances of the direct path), as it is more robust. We set the diagonal of the distance matrix to NA in order to increase the contrast of the color coding. Those diagonal entries do not contain information since the distance of a sample to itself is always equal to zero.

```
phenotype_names <- ifelse(str_detect(pData
                                   (palmieri_eset_norm)$Factor.Value.phenotype.,
                                   "non"), "non-infl.", "infl.")

disease_names <- ifelse(str_detect(pData
                                   (palmieri_eset_norm)$Factor.Value.phenotype.,
                                   "Chron"), "CD.", "UC")

row_names_dataframe <-
  data.frame("Phenotype" = phenotype_names, "Disease" = disease_names)
row_names_for_heatmap <-
  paste(row_names_dataframe$Phenotype, row_names_dataframe$Disease, sep = " ")

dists <- as.matrix(dist(t(exp_palmieri), method = "manhattan"))
colnames(dists) <- NULL
diag(dists) <- NA
rownames(dists) <- row_names_for_heatmap
hmcol <- colorRampPalette(brewer.pal(9, "PuOr"))(255)

pheatmap(dists, col = (hmcol),
          legend = TRUE,
          legend_breaks = c(min(dists, na.rm = TRUE),
                             max(dists, na.rm = TRUE)),
          legend_labels = (c("low difference", "high difference")),
          main = "Distance clustering heatmap for the calibrated samples")
```

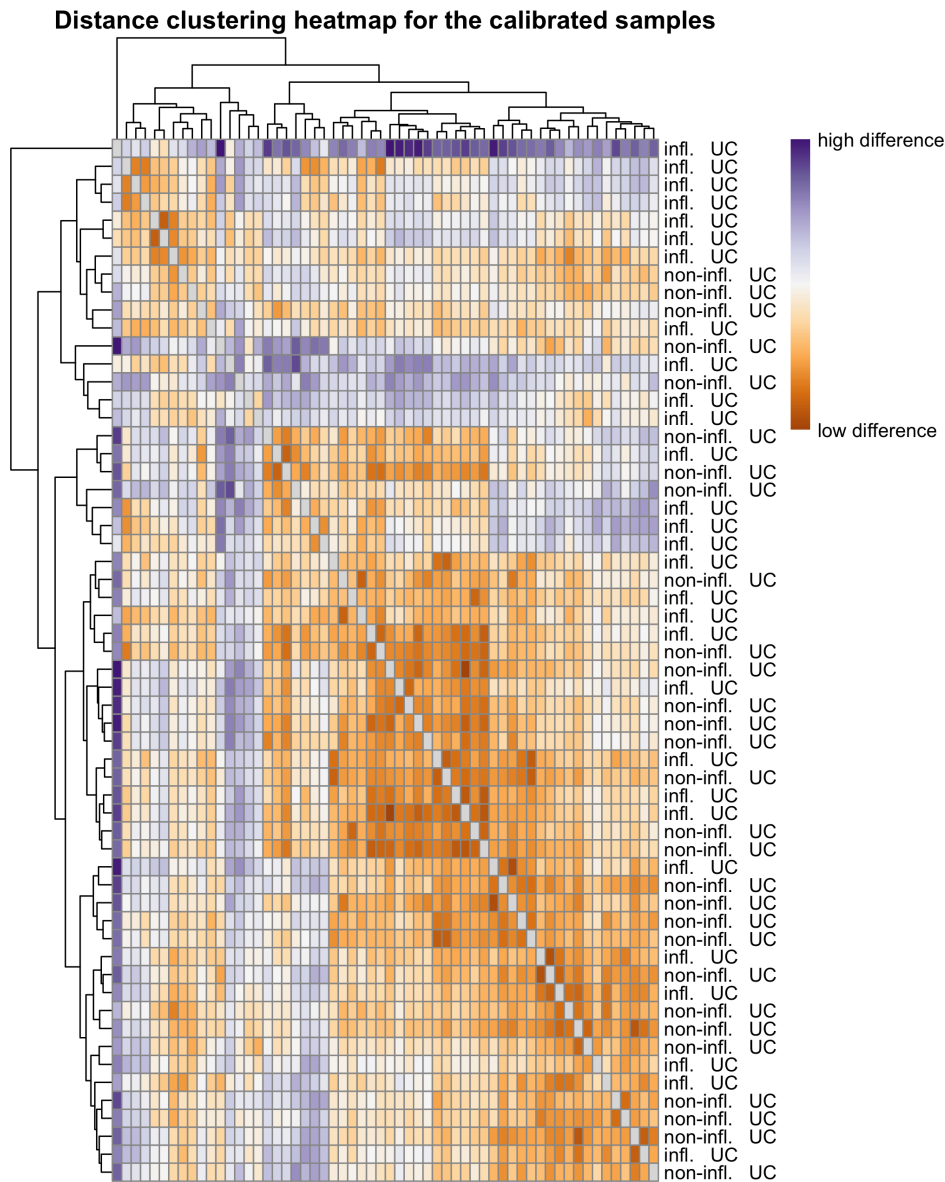


Figure 6. Heatmap of the sample to sample distances for the calibrated data.

On the heatmap plot (Figure 6) we also see that the samples do not cluster strongly by tissue, confirming the impression from the PCA plot (Figure 5) that the separation between the tissues is not perfect. The purple stripes in the heatmap might correspond to outliers that could potentially be removed. We produce the same heatmap again with the sample identifiers as row names in order to see whether the outliers in the heatmap are the same ones as the outliers in the RLE plot beforehand:

```
rownames(dists) <- row.names(pData(palmieri_eset_norm))
hmcol <- colorRampPalette(brewer.pal(9, "PuOr"))(255)
colnames(dists) <- NULL
diag(dists) <- NA

pheatmap(dists, col = (hmcol), legend = TRUE,
  legend_breaks = c(min(dists, na.rm = TRUE),
    max(dists, na.rm = TRUE)),
  legend_labels = (c("low difference", "high difference")),
  main = "Distance clustering heatmap for the calibrated samples")
```

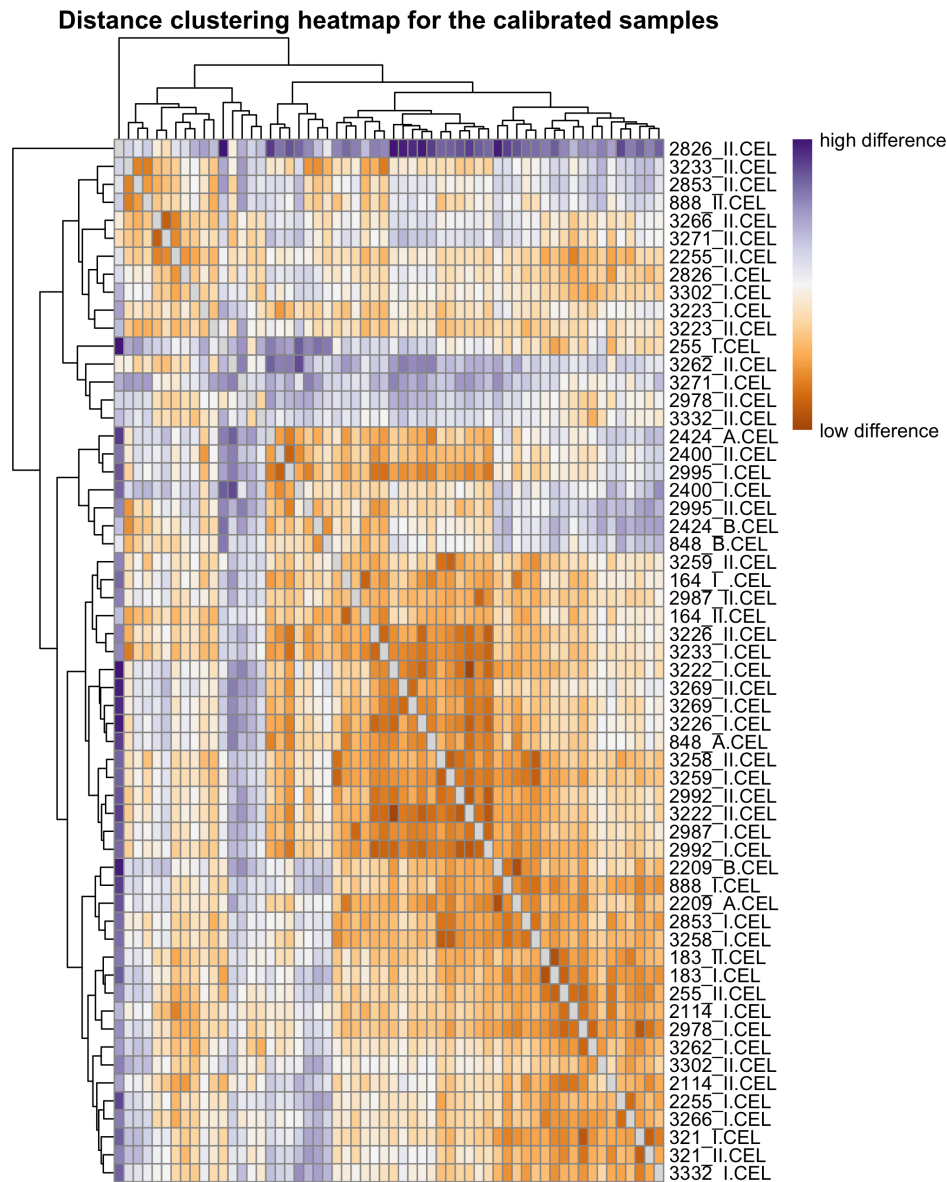


Figure 7. Heatmap with sample IDs

The outliers that could be flagged here are 2826_II, 3262_II, 3271_I, 2978_II and 3332_II (Figure 7). 2826_II, 3262_II and 3332_II were found to be outliers in both RLE and heatmap analysis and could therefore potentially be removed; however, in order to stay as close as possible to the original paper, we continue with the complete set of samples. Note again that more elaborate metrics to identify and remove outliers are provided by the *arrayQualityMetrics* package.

Filtering based on intensity

We now filter out lowly expressed genes. Microarray data commonly show a large number of probes in the background intensity range. These probes also do not change much across arrays. Hence they combine a low variance with a low intensity. Thus, they could end up being detected as differentially expressed although they are barely above the “detection” limit and are not very informative in general.

We will perform a “soft” intensity based filtering here, since this is recommended by *limma*’s [11, 12] user guide (a package we will use below for the differential expression analysis).

However, note that a variance based filter might exclude a similar set of probes in practice. We calculate the row-wise medians from the expression data, as they represent the transcript medians, and assign them to *palmieri_medians*. From this we create a histogram:

```
palmieri_medians <- rowMedians(Biobase::exprs(palmieri_eset_norm))
```

```
hist_res <- hist(palmieri_medians, 100, col = "cornsilk1", freq = FALSE,
  main = "Histogram of the median intensities",
  border = "antiquewhite4",
  xlab = "Median intensities")
```

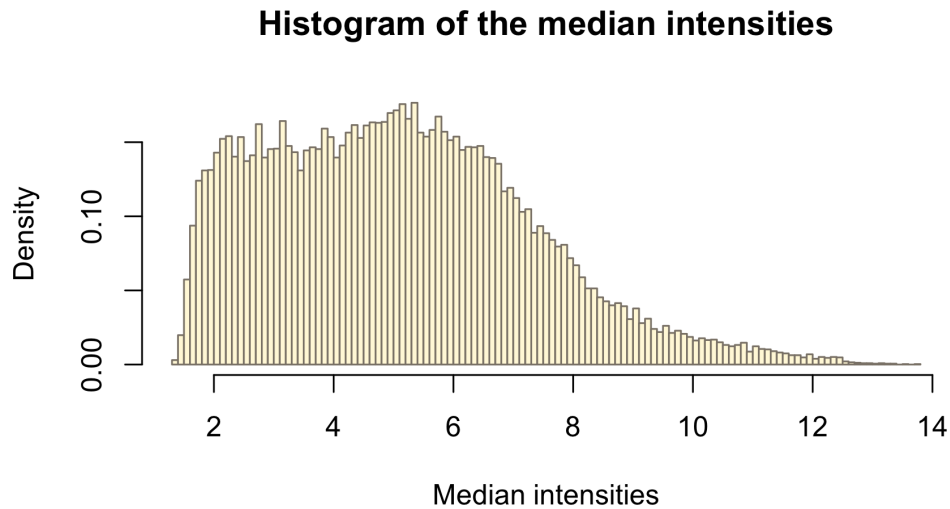


Figure 8. Histogram of the median intensities per gene

In the histogram of the gene-wise medians (Figure 8), we can clearly see an enrichment of low medians on the left hand side. These represent the genes we want to filter. In order to infer a cutoff from the data, we inspect the histogram. We visually set a cutoff line `man_threshold` to the left of the histogram peak in order not to exclude too many genes. Transcripts that don't have intensities larger than the threshold in at least as many arrays as the smallest experimental group are excluded.

In order to do so, we first have to get a list with the number of samples (=arrays) (`no_of_samples`) in the experimental groups:

```
no_of_samples <-
  table(paste0(pData(palmieri_eset_norm)$Factor.Value.disease., "_",
    pData(palmieri_eset_norm)$Factor.Value.phenotype.))
no_of_samples
```

```

      Crohn's disease_inflamed colonic mucosa
                                15
      Crohn's disease_non-inflamed colonic mucosa
                                15
      ulcerative colitis_inflamed colonic mucosa
                                14
      ulcerative colitis_non-inflamed colonic mucosa
                                14
```

We inspect the histogram and set a manual threshold visually. In our example, we choose a threshold of 4. We plot the same histogram as before and add the threshold line with the `abline()` function.

```
man_threshold <- 4

hist_res <- hist(palmieri_medians, 100, col = "cornsilk", freq = FALSE,
  main = "Histogram of the median intensities",
  border = "antiquewhite4",
  xlab = "Median intensities")

abline(v = man_threshold, col = "coral4", lwd = 2)
```

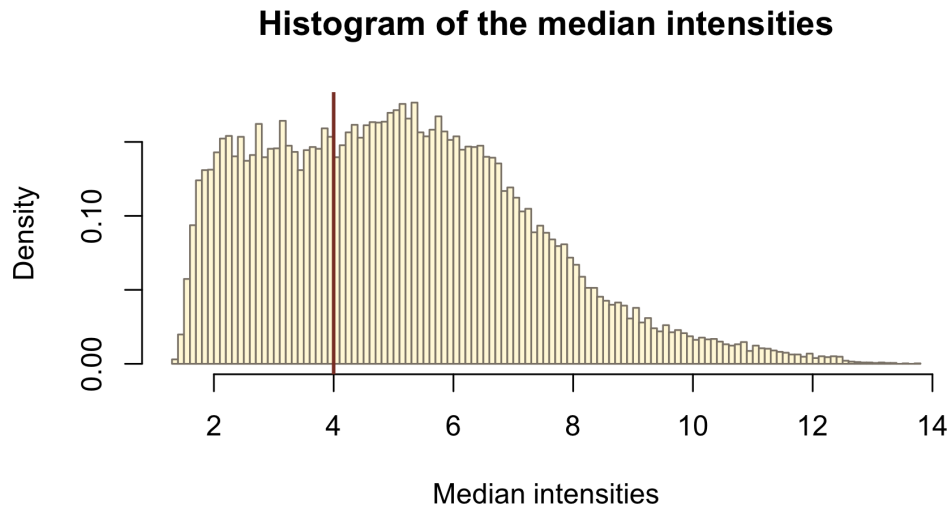



Figure 9. Histogram of the median intensities per gene with manual intensity filtering threshold (red line).

Once we are content with the manual threshold, we filter out all transcripts that don't have intensities greater than the threshold in at least as many arrays as the smallest experimental group (14,) which we define as `samples_cutoff`.

A function `idx_man_threshold` is then applied to each row, i.e. to each transcript across all arrays. It evaluates whether the number of arrays where the median intensity passes the threshold (`sum(x > man_threshold)`) is greater than the `samples_cutoff` and returns TRUE or FALSE for each row, i.e. each transcript.

We then create a table of `idx_man_threshold` in order to summarize the results, in order to get an overview over how many genes are filtered out. In the last step, we subset our expression set to `palmieri_manfiltered` and keep the TRUE elements of `idx_man_threshold`.

```
samples_cutoff <- min(no_of_samples)

idx_man_threshold <- apply(Biobase::exprs(palmieri_eset_norm), 1,
  function(x){
    sum(x > man_threshold) >= samples_cutoff})
table(idx_man_threshold)

idx_man_threshold
FALSE TRUE
10493 22804

palmieri_manfiltered <- subset(palmieri_eset_norm, idx_man_threshold)
```

Annotation of the transcript clusters

Before we continue with the linear models for microarrays and differential expression, we describe how to add "feature data", i.e. annotation information to the transcript cluster identifiers stored in the `featureData` of our `ExpressionSet`. We use the function `select` from *AnnotationDbi* to query the gene symbols and associated short descriptions for the transcript clusters. For each cluster, we add the gene symbol (`SYMBOL`) and a short description of the gene the cluster represents (`GENENAME`).

In a second step, we filter out the probes that don't map to a gene, i.e. that don't have a gene symbol assigned.

```
anno_palmieri <- AnnotationDbi::select(hugene10sttranscriptcluster.db,
  keys = (featureNames(palmieri_manfiltered)),
  columns = c("SYMBOL", "GENENAME"),
  keytype = "PROBEID")

anno_palmieri <- subset(anno_palmieri, !is.na(SYMBOL))
```

Removing multiple mappings

Many transcript-cluster identifiers will map to multiple gene symbols, i.e. they can't be unambiguously assigned.

We compute a summary table in the code below to see how many there are. First, we group `anno_palmieri` for their PROBEID; that way, the subsequent operations aren't carried through for each single row, but for each group, i.e. each PROBEID. We then summarize the groups and indicate the number of different genes assigned to a transcript cluster in the column `no_of_matches`. Finally, we filter for PROBEIDs with multiple matches, i.e. `no_of_matches > 1`.

With `dim(probe_stats)`, we can see how many probes have been mapped to multiple genes.

```
anno_grouped <- group_by(anno_palmieri, PROBEID)
anno_summarized <-
  dplyr::summarize(anno_grouped, no_of_matches = n_distinct(SYMBOL))
anno_filtered <- filter(anno_summarized, no_of_matches > 1)

probe_stats <- anno_filtered

dim(probe_stats)
```

```
[1] 1771    2
```

We have close to 2000 transcript clusters that map to multiple gene symbols. It is difficult to decide which mapping is "correct". Therefore, we exclude these transcript clusters.

We want to remove those probe IDs that match the ones in `probe_stats`, as those are the probes with multiple mappings. We assign these IDs to the variable `ids_to_exclude`. Then, we generate `palmieri_final`, an expression set without the `ids_to_exclude`.

```
ids_to_exclude <- (featureNames(palmieri_manfiltered) %in% probe_stats$PROBEID)

table(ids_to_exclude)
```

```
ids_to_exclude
FALSE TRUE
21033 1771
```

```
palmieri_final <- subset(palmieri_manfiltered, !ids_to_exclude)

validObject(palmieri_final)
```

```
[1] TRUE
```

As we have just excluded probe IDs from the assay data, we now have to also exclude them from the feature data `anno_palmieri`. Recall that `fData` enables us to access the feature data of an expression set. Until now, no feature data whatsoever is stored in the `fData(palmieri_final)`. Only the row names are named as the row names of the assay data by default, which are the PROBEIDs of the transcripts.

Therefore, we generate a column PROBEID in `fData(palmieri_final)` and assign the row names of `fData(palmieri_final)` to it. Then, we left-join `fData(palmieri_final)` with `anno_palmieri`, which already contains the columns "SYMBOL" and "GENENAME". A left-join keeps the rows and columns of the first argument and adds the corresponding column entries of the second argument.

By left-joining with `anno_palmieri`, we thus add the "SYMBOL" and "GENENAME" columns from `anno_palmieri` for only the PROBEIDs that are in `fData(palmieri_final)` and thus get the feature Data for the filtered probes.

```
fData(palmieri_final)$PROBEID <- rownames(fData(palmieri_final))
fData(palmieri_final) <- left_join(fData(palmieri_final), anno_palmieri)
```

```
Joining, by = "PROBEID"
```

```
# restore rownames after left_join
rownames(fData(palmieri_final)) <- fData(palmieri_final)$PROBEID

validObject(palmieri_final)
```

```
[1] TRUE
```

Building custom annotations

Alternatively, one can re-map the probes of the array to a current annotation. A workflow to do this for Illumina arrays is given in Arloth et al. [13]. Essentially, the individual probe sequences are re-aligned to an in-silico “exome” that consists of all annotated transcript exons.

In any case, the package *pdInfoBuilder* can be used to build custom annotation packages for use with *oligo*. In order to do this, PGF / CLF files (called “Library files” on the Affymetrix website) as well as the probeset annotations are required. The probesets typically represent small stretches of the genome (such as a single exon) and multiple probesets are then used to form a transcript cluster.

The CLF file contains information about the location of individual probes on the array. The PGF file then contains the individual probe sequences and shows the probeset they belong to. Finally, The probeset annotation .csv then contains information about which probesets are used in which transcript cluster. Commonly, multiple probesets are used in one transcript cluster and some probesets are contained in multiple transcript clusters.

Linear models

In order to analyse which genes are differentially expressed between inflamed- and non-inflamed tissue, we have to fit a linear model to our expression data. Linear models are the “workhorse” for the analysis of experimental data. They can be used to analyse almost arbitrarily complex designs, however they also take a while to learn and understand and a thorough description is beyond the scope of this workflow.

Mike Love’s and Michael Irizarry’s genomics class is a very good resource, especially its section on interactions and contrasts. It might also be helpful to learn a bit of linear algebra to understand the concepts better. The Khan Academy offers nice (and free) online courses.

Linear models for microarrays

We now apply linear models to microarrays. Specifically, we discuss how to use the *limma* package for differential expression analysis. The package is designed to analyze complex experiments involving comparisons between many experimental groups simultaneously while remaining reasonably easy to use for simple experiments. The main idea is to fit a linear model to the expression data for each gene. Empirical Bayes and other methods are used to borrow information across genes for the residual variance estimation leading to “moderated” t -statistics, and stabilizing the analysis for experiments with just a small number of arrays [12]. Conceptually, the final per gene variance is a mix of a prior variance and the per gene variance.

Typical experimental designs are discussed in chapter 9 of *limma* “User Guide”, which can be found on the Bioconductor landing page of *limma*.

In the following, we use appropriate design and contrast matrices for our linear models and fit a linear model to each gene separately.

A linear model for the data

The original paper is interested in changes in transcription that occur between inflamed and adjacent non-inflamed mucosal areas of the colon. This is studied in both inflammatory bowel disease types.

For building our linear model, we have to think about which experimental variables we want to consider. As we want to find differential expression between non-inflamed and inflamed tissue, in principle, those are the only two variables we would have to consider.

However, since we have two arrays per individual patient, we have a “Paired Samples” design (see section 9.4 of the *limma* user guide).

Thus, the first factor we need is a blocking factor for the individuals that will absorb differences in expression between them. Therefore, we block on patients, which means that the patient IDs become variables of the linear model.

Then we create factors that give us the grouping for the diseases and the tissue types (non-inflamed and inflamed). We furthermore simplify the names of the diseases to UC and DC and the names of the tissue types to nI and I.

Finally, we create two design matrices, one for each of the two diseases as we will analyze them separately in order to follow the analysis strategy of the original paper closely (one could also fit a joint model to the complete data set; however, the two diseases might show very different characteristics so that a joint fit might not be appropriate).

```
individual <-
  as.character(Biobase::pData(palmieri_final)$Characteristics.individual.)

tissue <- str_replace_all(Biobase::pData(palmieri_final)$Factor.Value.phenotype.,
  " ", "_")

tissue <- ifelse(tissue == "non-inflamed_colonic_mucosa",
  "nI", "I")

disease <-
  str_replace_all(Biobase::pData(palmieri_final)$Factor.Value.disease.,
  " ", "_")

disease <-
  ifelse(disease == "Crohn's_disease", "CD", "UC")

i_CD <- individual[disease == "CD"]
design_palmieri_CD <- model.matrix(~ 0 + tissue[disease == "CD"] + i_CD)
colnames(design_palmieri_CD)[1:2] <- c("I", "nI")
rownames(design_palmieri_CD) <- i_CD

i_UC <- individual[disease == "UC"]
design_palmieri_UC <- model.matrix(~ 0 + tissue[disease == "UC"] + i_UC)
colnames(design_palmieri_UC)[1:2] <- c("I", "nI")
rownames(design_palmieri_UC) <- i_UC
```

We can inspect the design matrices:

```
head(design_palmieri_CD[, 1:6])
```

	I	nI	i_CD183	i_CD2114	i_CD2209	i_CD2255
164	0	1	0	0	0	0
164	1	0	0	0	0	0
183	0	1	1	0	0	0
183	1	0	1	0	0	0
2114	0	1	0	1	0	0
2114	1	0	0	1	0	0

```
head(design_palmieri_UC[, 1:6])
```

	I	nI	i_UC2424	i_UC2987	i_UC2992	i_UC2995
2400	0	1	0	0	0	0
2400	1	0	0	0	0	0
2424	0	1	1	0	0	0
2424	1	0	1	0	0	0
2987	0	1	0	1	0	0
2987	1	0	0	1	0	0

In the design matrix, the rows represent the patient array, and the columns are the variables we include in our linear model. For example, the first two rows of the design matrix `design_palmieri_CD` correspond to the two arrays for individual “164”.

The design matrix entries are 0 or 1 and thus tells us which variables are “active” for which sample:

It can be seen as a switch that turns the single variables “on” (with a 1 at the corresponding position) or “off” (with a 0 at the corresponding position) for each row, i. e. each patient array. If we take a closer look at the

single rows, we see that for each sample, there are two “ones” assigned: one to one of the variables “nI” or “I” corresponding to the tissue the array came from, and one to the corresponding patient-specific blocking variable.

Note that in the linear model, individual 164 serves as the baseline for all other individuals and thus isn't included in the sample variables.

Before heading on to find all differentially expressed genes for both diseases, we will first have a look on how this works in principle for one gene. We will fit the linear model for one gene and run a t-test in order to see whether the gene is differentially expressed or not.

Analysis of differential expression based on a single gene

For linear model fitting and subsequent testing for differential expression by t-test, we will pick the gene with the PROBEID 8164535. It has the gene symbol *CRAT* and will be named as such in the following code.

Illustration of the fitted linear model on the *CRAT* gene

Before fitting the linear model, we have a look at the expression intensities of this gene for each patient in non-inflamed and inflamed tissue, respectively:

```
tissue_CD <- tissue[disease == "CD"]
crat_expr <- Biobase::exprs(palmieri_final)["8164535", disease == "CD"]
crat_data <- as.data.frame(crat_expr)
colnames(crat_data)[1] <- "org_value"
crat_data <- mutate(crat_data, individual = i_CD, tissue_CD)

crat_data$tissue_CD <- factor(crat_data$tissue_CD, levels = c("nI", "I"))

ggplot(data = crat_data, aes(x = tissue_CD, y = org_value,
                             group = individual, color = individual)) +
  geom_line() +
  ggtitle("Expression changes for the CRAT gene")
```

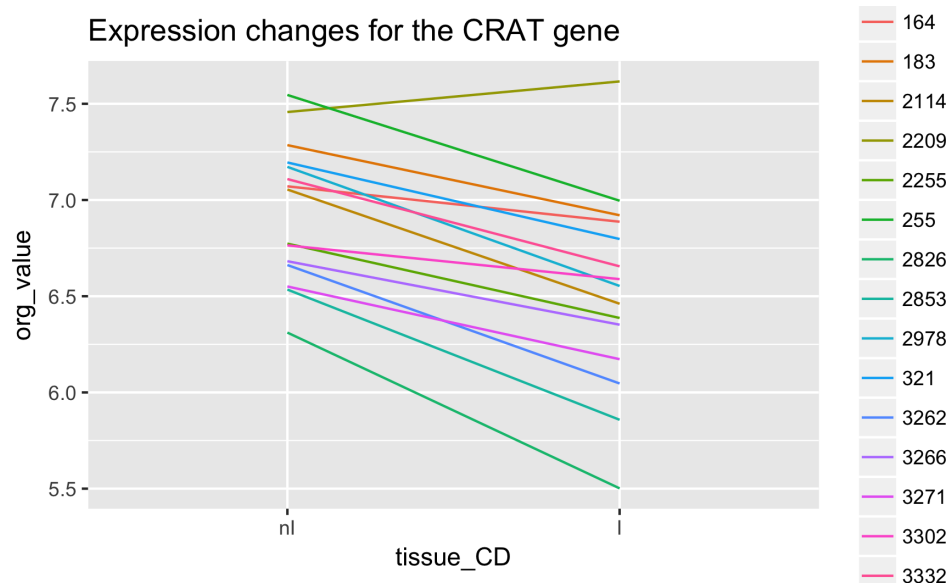


Figure 10. Visualization of expression changes

We see that overall, this gene is expressed less in inflamed tissue (Figure 10). We also see that the absolute expression intensities vary a lot between patients. However, we have already taken care of this problem by introducing blocking factors based on the individuals, which allows us to compare the tissues within each individual as represented by the single lines.

If we hadn't blocked for individuals, the linear model would treat them interchangeably and a graphical depiction would only include a single line. Since the individuals have very different baseline expression levels, this would lead to a very high variance of the estimated fold changes.

We now compute the variable coefficients by fitting a linear model. We get a vector `crat_coef` with one entry for each variable.

```
crat_coef <- lmFit(palmieri_final[,disease == "CD"],
                  design = design_palmieri_CD)$coefficients["8164535",]

crat_coef
```

	I	nI	i_CD183	i_CD2114	i_CD2209	i_CD2255	i_CD255	i_CD2826
	6.76669	7.19173	0.12382	-0.22145	0.55759	-0.39905	0.29204	-1.07285
	i_CD2853	i_CD2978	i_CD321	i_CD3262	i_CD3266	i_CD3271	i_CD3302	i_CD3332
	-0.78285	-0.11633	0.01692	-0.62480	-0.46209	-0.61732	-0.30257	-0.09709

In order to now obtain the expression values fitted by the model, we multiply the design matrix with this vector of coefficients `crat_coef`.

Recall that for every row in the design matrix (i.e. every patient sample) only the variables with a 1 in the design matrix are taken into account for calculating the fitted expression value.

This means that as output of the multiplication, we get a vector `crat_fitted` whose entries are the sum of relevant variable coefficients for each sample, respectively.

For example, the fitted value for patient sample 2114_I.CEL is 6.9703: it is the sum of the respective activated variable coefficients “nI” (7.1917) and “i_CD2114” (-0.2215).

```
crat_fitted <- design_palmieri_CD %*% crat_coef
rownames(crat_fitted) <- names(crat_expr)
colnames(crat_fitted) <- "fitted_value"

crat_fitted
```

	fitted_value
164_I_.CEL	7.192
164_II.CEL	6.767
183_I.CEL	7.316
183_II.CEL	6.891
2114_I.CEL	6.970
2114_II.CEL	6.545
2209_A.CEL	7.749
2209_B.CEL	7.324
2255_I.CEL	6.793
2255_II.CEL	6.368
255_I.CEL	7.484
255_II.CEL	7.059
2826_I.CEL	6.119
2826_II.CEL	5.694
2853_I.CEL	6.409
2853_II.CEL	5.984
2978_I.CEL	7.075
2978_II.CEL	6.650
321_I.CEL	7.209
321_II.CEL	6.784
3262_I.CEL	6.567
3262_II.CEL	6.142
3266_I.CEL	6.730
3266_II.CEL	6.305
3271_I.CEL	6.574
3271_II.CEL	6.149
3302_I.CEL	6.889
3302_II.CEL	6.464
3332_I.CEL	7.095
3332_II.CEL	6.670

Let's visualize the difference between non-inflamed and inflamed tissue again after fitting:

```

crat_data$fitted_value <- crat_fitted

ggplot(data = crat_data, aes(x = tissue_CD, y = fitted_value,
                             group = individual, color = individual)) +
  geom_line() +
  ggtitle("Fitted expression changes for the CRAT gene")

```

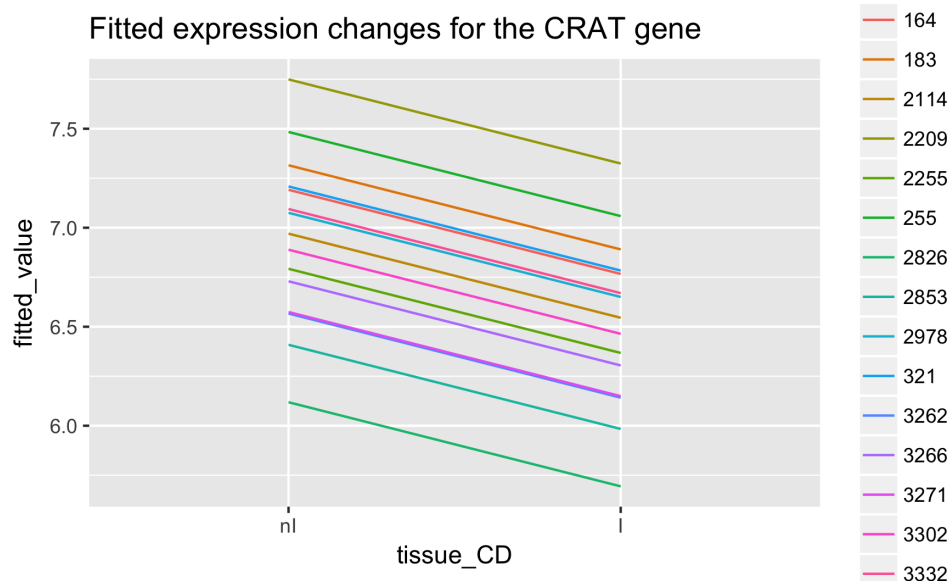


Figure 11. Expression changes for the CRAT gene

Note that the difference of the fitted expression values between inflamed– and non–inflamed samples of one patient is the same for all patients and is determined by the difference between the variable coefficients of I (6.7667) and nI (7.1917), which is -0.425 (Figure 11).

This is the case because the same blocking variable is activated by the design matrix for both samples from a single patient, leading to a comparison within patients only. These blocking variables correct the fitted tissue specific expression values towards the expression levels of the individual patients. Therefore the final estimate is like an average of all the within–individual differences.

The “difference” between non–inflamed and inflamed tissue of -0.425 is actually a log2 fold change, as our expression data is on the log2 scale. -0.425 therefore is our log2 fold change for the CRAT gene.

Differential expression analysis of the CRAT gene

In order to test whether the gene is differentially expressed or not, a *t*–test with the null hypothesis that there is no difference in the expression between non–inflamed and inflamed tissue is carried out. Our blocking design is conceptually similar to a paired *t*–test for which the statistic is given by:

$$t = \frac{\bar{d}}{s/\sqrt{n}}$$

Where, \bar{d} is the mean difference between the individuals. The paired *t*–test computes the variance s^2 from the paired differences. This is lower than the variance of a standard *t*–test and thus the paired *t*–test has higher power as long as the expression values for the same individual are correlated (see e.g. the article on Wikipedia).

We thus have improved the power of the ordinary *t*–test by reducing the variance via blocking on individuals.

We now conduct the *t*–test on the linear model in order to find out whether the difference between non–inflamed– and inflamed tissue differs significantly from 0:

```
crat_noninflamed <- na.exclude(crata_data$org_value[tissue == "nI"])
crat_inflamed <- na.exclude(crata_data$org_value[tissue == "I"])
res_t <- t.test(crat_noninflamed , crat_inflamed , paired = TRUE)
res_t
```

Paired t-test

```
data: crat_noninflamed and crat_inflamed
t = 6.8, df = 14, p-value = 8e-06
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.2919 0.5581
sample estimates:
mean of the differences
      0.425
```

We get a low p-value close to 0 and thus can conclude that the *CRAT* gene is differentially expressed between non-inflamed and inflamed tissue.

Note that the p-value isn't exactly the same one as below when analyzing the differential expression of all genes. The reason is a variance moderation performed by *limma*.

Contrasts and hypotheses tests

We now fit the linear model for all genes and define appropriate contrasts to test hypotheses of interest.

We want to compare the inflamed to the non-inflamed tissue. Thus, we create a contrast matrix consisting of only one contrast "I-nI": *limma*'s function `makeContrasts` creates this matrix from a symbolic description of the contrast of interest.

We now fit a linear model to our data and apply the `contrasts.fit()` function to it in order to find genes with significant differential expression between non-inflamed and inflamed tissue.

We apply the empirical Bayes variance moderation method to it via the `eBayes()` function, which computes moderated *t*-statistics: As in microarray analysis, the number of arrays often is quite small, variance estimation is difficult. Using a combination of the per-gene-variance and a prior variance we can improve the variance estimate. Hence the term "moderation". "Empirical Bayes" means that the prior is estimated from the data.

The result of the `eBayes()` step is that the individual variances are shrunk towards the prior value.

```
contrast_matrix_CD <- makeContrasts(I-nI, levels = design_palmieri_CD)

palmieri_fit_CD <- eBayes(contrasts.fit(lmFit(palmieri_final[,disease == "CD"],
      design = design_palmieri_CD),
      contrast_matrix_CD))

contrast_matrix_UC <- makeContrasts(I-nI, levels = design_palmieri_UC)

palmieri_fit_UC <- eBayes(contrasts.fit(lmFit(palmieri_final[,disease == "UC"],
      design = design_palmieri_UC),
      contrast_matrix_UC))
```

Extracting results

Finally, we extract the number of differentially expressed genes. Results can be extracted by use of the `topTable` function. We extract the results for both Crohn's disease and ulcerative colitis, and the results are sorted by their absolute *t*-statistics. As a diagnostic check, we also plot the p-value histogram: We expect a uniform distribution for the p-values that correspond to true null hypotheses, while a peak near zero shows an enrichment for low p-values corresponding to differentially expressed (DE) genes.

Note that if the p-value distribution for a dataset is very different from the ones in the histograms below, this might lead to quality loss in the subsequent analysis. Reasons for a divergent p-value-distribution might be batch effects or a lack of consideration of other blocking factors in the design matrix. Thus, try to include

possible blocking factors and batches and rerun the analysis. If this does not help, empirical Bayes / null estimation methods for multiple testing are useful.

A good starting point to learn about these methods is the article on false discovery rate estimation by Korbinian Strimmer [14] and chapter 1–6 of Efron's book on Large-Scale Inference [15].

```
table_CD <- topTable(palmieri_fit_CD, number = Inf)
head(table_CD)
```

	PROBEID	SYMBOL	GENENAME	logFC		
	7928695	FAM213A	family with sequence similarity 213 member A	-0.5990		
	8123695	ECI2	enoyl-CoA delta isomerase 2	-0.4855		
	8164535	CRAT	carnitine O-acetyltransferase	-0.4250		
	8009746	SLC16A5	solute carrier family 16 member 5	-0.5182		
	7952249	USP2	ubiquitin specific peptidase 2	-0.8484		
	8105348	GPX8	glutathione peroxidase 8 (putative)	0.8312		
	AveExpr	t	P.Value	adj.P.Val	B	
	7928695	7.739	-7.059	1.383e-06	0.02092	5.305
	8123695	6.876	-6.317	5.907e-06	0.02092	4.028
	8164535	6.732	-6.230	7.037e-06	0.02092	3.872
	8009746	5.562	-6.206	7.386e-06	0.02092	3.829
	7952249	5.606	-6.203	7.429e-06	0.02092	3.824
	8105348	5.301	6.074	9.656e-06	0.02092	3.590

```
hist(table_CD$P.Value, col = brewer.pal(3, name = "Set2")[1],
     main = "inflamed vs non-inflamed - Crohn's disease", xlab = "p-values")
```

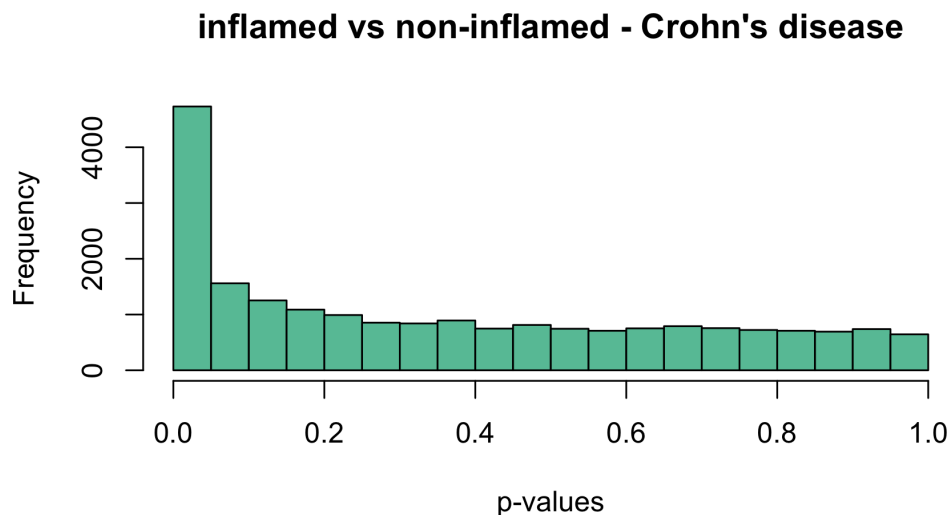


Figure 12. Histogram of the p-values for Crohn's disease.

```
table_UC <- topTable(palmieri_fit_UC, number = Inf)
head(table_UC)
```

	PROBEID	SYMBOL	GENENAME	logFC	AveExpr
	8003875	SPNS2	sphingolipid transporter 2	0.7412	6.478
	8082012	SLC15A2	solute carrier family 15 member 2	0.8061	5.319
	7952290	TRIM29	tripartite motif containing 29	1.0140	5.855
	7961693	LDHB	lactate dehydrogenase B	0.3968	9.534
	8072015	GRK3	G protein-coupled receptor kinase 3	0.4713	5.584
	8096070	BMP3	bone morphogenetic protein 3	-1.6961	6.420
	t	P.Value	adj.P.Val	B	
	8003875	7.801	4.553e-07	0.003983	6.441
	8082012	7.744	5.033e-07	0.003983	6.352
	7952290	7.482	8.009e-07	0.003983	5.937
	7961693	7.401	9.265e-07	0.003983	5.806
	8072015	7.308	1.097e-06	0.003983	5.654
	8096070	-7.289	1.136e-06	0.003983	5.623

```
hist(table_UC$P.Value, col = brewer.pal(3, name = "Set2")[2],
     main = "inflamed vs non-inflamed - Ulcerative colitis", xlab = "p-values")
```

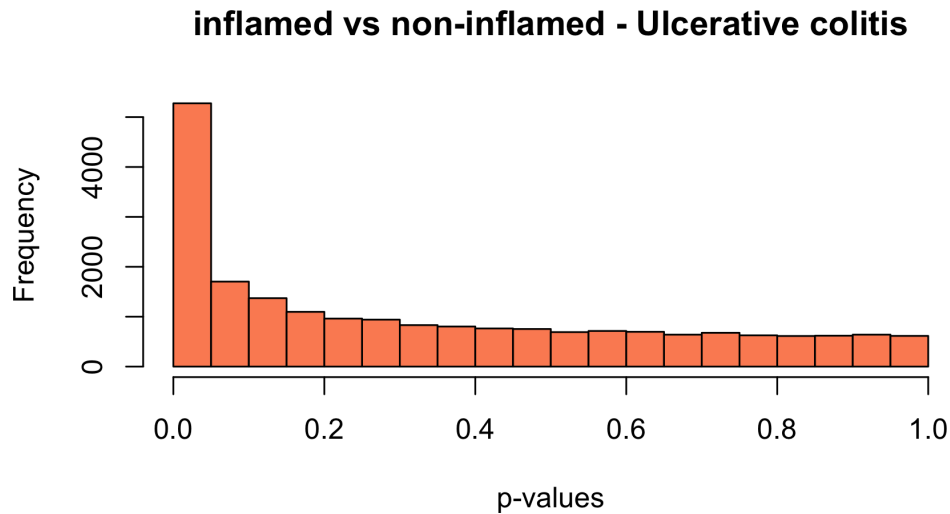


Figure 13. Histogram of the p-values for ulcerative colitis.

Multiple testing FDR, and comparison to the paper results

In the original paper, a p-value of 0.001 was used as a significance cutoff. Using this we get 947 genes identified as differentially expressed for UC:

```
nrow(subset(table_UC, P.Value < 0.001))
```

```
[1] 947
```

However, it is impossible to determine a precise bound on the number of false positive genes in this list. All that we can say using p-values is that we have at most $21033 \text{ (total number of tests)} \times 0.01 = 210.33$ false positive genes in our list. Therefore, by choosing a p-value cutoff of 0.01, as much as 22.21% of our genes identified as differentially expressed might be false positives.

Thus, we can see that the “raw” p-values are very “liberal” when looking at many tests simultaneously. We therefore need error rates adapted to the multiple testing situation. By far the most popular one in molecular biology is the **false discovery rate** or FDR for short. As we have seen, the FDR of our genes list using a simple p-value cutoff might be quite high.

On the other hand, we can see a clear peak in the p-value histogram (Figure 13), caused by the differentially expressed genes. There we expect the actual FDR of our list to be lower.

The FDR at a given given cutoff is given by the “adjusted” p-value in the results table.

```
tail(subset(table_UC, P.Value < 0.001))
```

PROBEID	SYMBOL	GENENAME	logFC
7915640	7915640	EIF2B3	0.2795
7894577	7894577	<NA>	0.4048
7897877	7897877	TNFRSF1B	0.3564
8142671	8142671	WASL	-0.4281
7941946	7941946	NDUFV1	-0.3832
8140371	8140371	TMEM120A	-0.3657

	AveExpr	t	P.Value	adj.P.Val	B
7915640	6.271	3.957	0.0009867	0.02203	-0.6478
7894577	5.598	3.956	0.0009892	0.02205	-0.6501
7897877	6.656	3.956	0.0009899	0.02205	-0.6508
8142671	7.907	-3.955	0.0009906	0.02205	-0.6515
7941946	8.867	-3.952	0.0009976	0.02218	-0.6581
8140371	6.519	-3.951	0.0009990	0.02219	-0.6594

The adjusted p-value for a raw p-value of 0.001 in the table is 0.0222, which is an order of magnitude lower than the FDR we can infer from p-values alone.

So although this is not recommended in general, we also use a p-value cutoff at 0.001 in the following in order to be able to compare our workflow results to the paper results.

The paper results can be downloaded as excel files from <http://links.lww.com/IBD/A795> and should be saved as an .xlsx file named `palmieri_DE_res.xlsx` in your working directory.

Note that while the paper uses p-value cutoffs, it also reports the corresponding FDRs (just as we did for the UC data here).

For a p-value cutoff of 0.001, the corresponding FDRs are 0.05 in Crohn's disease and 0.02 in ulcerative colitis. There are four tables in total, giving the list of up and downregulated genes in CD and UC, respectively. We calculate the overlap between our results and the ones from the paper as the ratio of the genes that were found in both analyses and the genes that were only found in the paper.

We also calculate the total number of differentially expressed genes that we find in our workflow analysis.

```
fpath <- system.file("extdata", "palmieri_DE_res.xlsx", package = "maEndToEnd")
palmieri_DE_res <- sapply(1:4, function(i) read.xlsx(cols = 1, fpath,
                                                    sheet = i, startRow = 4))

names(palmieri_DE_res) <- c("CD_UP", "CD_DOWN", "UC_UP", "UC_DOWN")
palmieri_DE_res <- lapply(palmieri_DE_res, as.character)
paper_DE_genes_CD <- Reduce("c", palmieri_DE_res[1:2])
paper_DE_genes_UC <- Reduce("c", palmieri_DE_res[3:4])

overlap_CD <- length(intersect(subset(table_CD, P.Value < 0.001)$SYMBOL,
                                   paper_DE_genes_CD)) / length(paper_DE_genes_CD)

overlap_UC <- length(intersect(subset(table_UC, P.Value < 0.001)$SYMBOL,
                                   paper_DE_genes_UC)) / length(paper_DE_genes_UC)

overlap_CD
```

```
[1] 0.6443
```

```
overlap_UC
```

```
[1] 0.6731
```

```
#add total number of genes found with our method
total_genenumber_CD <- length(subset(table_CD, P.Value < 0.001)$SYMBOL)
total_genenumber_UC <- length(subset(table_UC, P.Value < 0.001)$SYMBOL)

total_genenumber_CD
```

```
[1] 575
```

```
total_genenumber_UC
```

```
[1] 947
```

We find 575 (CD) and 947 (UC) differentially expressed genes (“DE–genes”).

In the paper, 298 (CD) and 520 (UC) DE–genes were found for the two diseases at the same cutoff. This higher number of DE–genes identified is probably due to the increased power of the blocking according to the individuals and the moderated variance estimation that *limma* performs.

We see that we get a moderate overlap of 0.6443 for CD and 0.6731 for UC, showing that both analysis lead to somewhat comparable results.

Gene ontology (GO) based enrichment analysis

As just discussed, it is recommended to use an FDR cutoff in differential expression analysis rather than a p–value cutoff, since this way you control an explicitly defined error rate and the results are easier to interpret and to compare. For the following enrichment analysis, we create tables with differentially expressed genes for CD and UC, respectively, and choose an FDR cutoff of 10%. Here, we focus on the CD subset of the data.

```
DE_genes_CD <- subset(table_CD, adj.P.Val < 0.1)$PROBEID
```

We can now try to characterize the identified differentially expressed genes a bit better by performing a GO enrichment analysis. Essentially the gene ontology (<http://www.geneontology.org/>) is a hierarchically organized collection of functional gene sets [16, 17, 18].

Matching the background set of genes

The function `genefinder` from the *genefilter* package [19] will be used to find a background set of genes that are similar in expression to the differentially expressed genes. We then check whether the background has roughly the same distribution of average expression strength as the foreground.

We do this in order not to select a biased background since the gene set testing is performed by a simple Fisher test on a 2x2 table. Note that this approach is very similar to commonly used web tools like GOrilla [20].

For every differentially expressed gene, we try to find genes with similar expression with `genefinder`. The `genefinder` function returns two lists for each gene: one with the indices of the background genes found and one with the distances to the DE–genes. We have to extract the PROBEIDs, which correspond to the indices. We do that by using the `sapply` function, which gives us a single matrix with the DE–genes as column names and the PROBEIDs of the corresponding background genes in the cells below.

We then create a vector `back_genes` containing all background gene PROBEIDs.

In order to eliminate foreground genes, i.e. DE–genes, from the `back_genes` set, we use the `setdiff` function. It returns all elements from the first argument (`back_genes`) that are not part of the second argument (`DE_genes_CD`). With the `intersect` function, we verify that we were successful: it should return 0, as there shouldn’t be any intersect anymore between `back_genes` and `DE_genes_CD`.

We create a multidensity plot with mean expression on the x–axis and curves for all genes, foreground genes and background genes, respectively (Figure 14). We want to see whether the background genes show a plot similar to the foreground genes so that the background is not biased for the gene enrichment analysis.

```
back_genes_idx <- genefilter::genefinder(palmieri_final,
                                       as.character(DE_genes_CD),
                                       method = "manhattan", scale = "none")
```

```
back_genes_idx <- sapply(back_genes_idx, function(x)x$indices)
```

```
back_genes <- featureNames(palmieri_final)[back_genes_idx]
back_genes <- setdiff(back_genes, DE_genes_CD)
```

```
intersect(back_genes, DE_genes_CD)
```

```
character(0)
```

```
length(back_genes)
```

```
[1] 9756
```

```

multidensity(list(
  all = table_CD[, "AveExpr"] ,
  fore = table_CD[DE_genes_CD , "AveExpr"],
  back = table_CD[rownames(table_CD) %in% back_genes, "AveExpr"]),
  col = c("#e46981", "#ae7ee2", "#a7ad4a"),
  xlab = "mean expression",
  main = "DE genes for CD-background-matching")

```

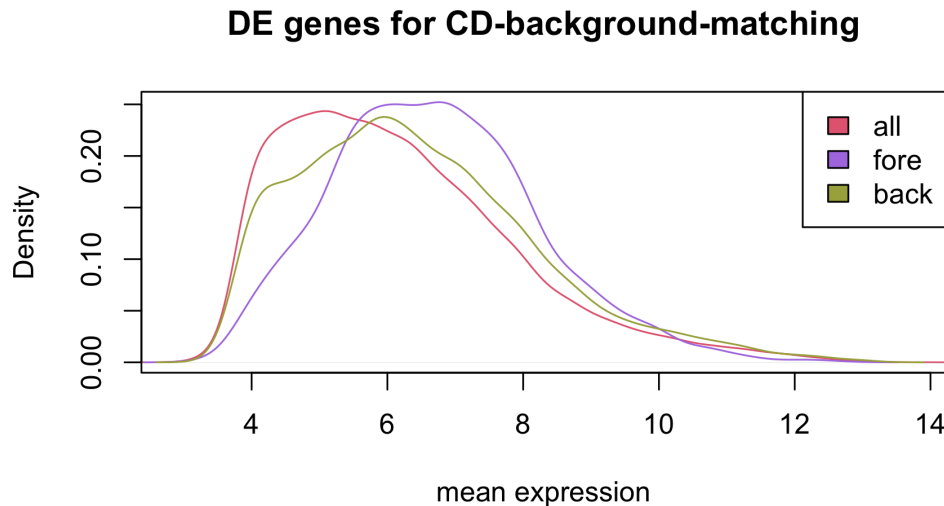


Figure 14. Selecting a background set of genes for the gene ontology analysis.

When comparing the “background gene” curve to the “foreground gene” curve, we see a similar curve shape, indicating a sensible background matching (Figure 14). Note that the right-shift of the “foreground-gene” curve in comparison to the “background-gene” curve indicates that DE-genes are generally very highly expressed, so that it wasn’t possible to find background-genes with exactly equal overall expression distribution.

The “all gene” curve has the leftmost curve maximum, this can be explained by a high number of lowly expressed genes in all samples and shows that a background matching is sensible in order to avoid biases.

For the actual testing of which GO gene sets are enriched in inflamed tissue, we use the *topGO* package which implements a nice interface to Fisher testing and also has additional algorithms taking the GO structure into account, by e.g. only reporting the most specific gene set in the hierarchy [21].

The GO has three top ontologies: Cellular component (CC), biological processes (BP), and molecular function (MF). For illustrative purposes we limit ourselves to the BP category here.

Running topGO

topGO requires a *topGOdata* object containing the necessary information for the analysis. We follow the steps described in the *topGO* vignettes: First, we will create a named vector `all_genes` with all genes to be analyzed, i.e. DE-genes and background genes:

1. we create an `in_universe` vector by using the `%in%` matching function. We want to know which elements from `gene_IDs` are also contained in `DE_genes_CD` and `back_genes`, as the latter two are our gene universe we use for enrichment analysis. We get a vector `in_universe` with the length of `gene_IDs` that has the entry `TRUE` when the corresponding gene in `gene_IDs` could be also found in `DE_genes_CD` or `back_genes`, and `FALSE` otherwise.
2. We do the same for our DE-genes and call this vector `in_selection`.
3. We create the `all_genes` vector:
 - a) First, we select all the elements from `in_selection` that are `TRUE` in `in_universe` by applying `all_genes <- in_selection[in_universe]`.
 - b) Then, we convert the elements in `all_genes` from `TRUE` and `FALSE` to 0 and 1 by converting the vector to an integer vector. This way, each element in the vector is a 0 if the corresponding gene is a background gene and a 1 if the corresponding gene is a DE-gene. Also, we convert the vector to a factor.

c) We name the vector elements with the corresponding `gene_IDs`.

```
gene_IDs <- rownames(table_CD)
in_universe <- gene_IDs %in% c(DE_genes_CD , back_genes)
in_selection <- gene_IDs %in% DE_genes_CD

all_genes <- in_selection[in_universe]
all_genes <- factor(as.integer(in_selection[in_universe]))
names(all_genes) <- gene_IDs[in_universe]
```

We now initialize the *topGO* data set, using the GO annotations contained in the annotation data base for the chip we are using. The `nodeSize` parameter specifies a minimum size of a GO category we want to use: i.e. here, categories with less than 10 genes are not included in the testing.

```
top_GO_data <- new("topGOdata", ontology = "BP", allGenes = all_genes,
  nodeSize = 10, annot = annFUN.db, affyLib = "hugene10sttranscriptcluster.db")
```

Now the tests can be run. *topGO* offers a wide range of options, for details see the paper [21] or the package vignette.

We run two common tests: an ordinary Fisher test for every GO category, and the “elim” algorithm, which tries to incorporate the hierarchical structure of the GO and tries to “decorrelate” it in order to report the most specific significant term in the hierarchy.

The algorithm starts processing the nodes/GO categories from the highest (bottommost) level and then iteratively moves to nodes from a lower level. If a node is scored as significant, all of its genes are marked as removed in all ancestor nodes. This way, the “elim” algorithm aims at finding the most specific node for every gene.

The test uses a 0.01 p-value cutoff by default.

```
result_top_GO_elim <-
  runTest(top_GO_data, algorithm = "elim", statistic = "Fisher")
result_top_GO_classic <-
  runTest(top_GO_data, algorithm = "classic", statistic = "Fisher")
```

We can now inspect the results. We look at the top 100 GO categories according to the “Fisher elim” algorithm. The function `GenTable` produces a table of significant GO categories, the function `printGenes` gives genes annotated to them; the significant ones are denoted with a “2” in the “raw p-value” column, the non-significant ones with a “1”. We therefore select `raw p-value == 2`. Note that we don’t get the actual p-values here because our `all_genes` vector doesn’t contain this information; it only tells us whether a gene is differentially expressed or not.

```
res_top_GO <- GenTable(top_GO_data, Fisher.elim = result_top_GO_elim,
  Fisher.classic = result_top_GO_classic,
  orderBy = "Fisher.elim" , topNodes = 100)

genes_top_GO <- printGenes(top_GO_data, whichTerms = res_top_GO$GO.ID,
  chip = "hugene10sttranscriptcluster.db", geneCutOff = 1000)

res_top_GO$sig_genes <- sapply(genes_top_GO, function(x){
  str_c(paste0(x[x$'raw p-value' == 2, "Symbol.id"],";"),
    collapse = "")
})

head(res_top_GO[,1:8], 20)
```

	GO.ID	Term	Annotated	Significant
1	GO:0032496	response to lipopolysaccharide	223	96
2	GO:0006954	inflammatory response	442	193
3	GO:0051897	positive regulation of protein kinase B ...	100	49
4	GO:0050900	leukocyte migration	282	134
5	GO:0030335	positive regulation of cell migration	318	130
6	GO:0006911	phagocytosis, engulfment	54	29

7	GO:0030198	extracellular matrix organization	223	96
8	GO:0070098	chemokine-mediated signaling pathway	45	25
9	GO:0070374	positive regulation of ERK1 and ERK2 cas...	144	57
10	GO:0007186	G-protein coupled receptor signaling pat...	441	146
11	GO:0048661	positive regulation of smooth muscle cel...	57	32
12	GO:0030574	collagen catabolic process	48	25
13	GO:0002675	positive regulation of acute inflammator...	20	14
14	GO:0030593	neutrophil chemotaxis	54	32
15	GO:0042493	response to drug	566	177
16	GO:0001937	negative regulation of endothelial cell ...	30	18
17	GO:0016525	negative regulation of angiogenesis	70	32
18	GO:0030449	regulation of complement activation	44	23
19	GO:0006067	ethanol metabolic process	12	10
20	GO:0033540	fatty acid beta-oxidation using acyl-CoA...	12	10
Expected Rank in Fisher.classic Fisher.elim Fisher.classic				
1	49.11	56	3.1e-10	1.0e-12
2	97.34	1	1.0e-09	1.8e-25
3	22.02	115	2.2e-09	2.2e-09
4	62.10	6	1.3e-07	6.6e-22
5	70.03	39	3.2e-07	1.2e-14
6	11.89	205	3.5e-07	3.5e-07
7	49.11	57	7.7e-07	1.0e-12
8	9.91	231	9.7e-07	9.7e-07
9	31.71	242	1.3e-06	1.3e-06
10	97.12	150	2.6e-06	2.6e-08
11	12.55	149	4.3e-06	2.1e-08
12	10.57	283	4.7e-06	4.7e-06
13	4.40	290	6.0e-06	6.0e-06
14	11.89	125	7.1e-06	3.3e-09
15	124.65	171	7.5e-06	8.8e-08
16	6.61	297	7.6e-06	7.6e-06
17	15.42	303	8.6e-06	8.6e-06
18	9.69	311	1.0e-05	1.0e-05
19	2.64	318	1.1e-05	1.1e-05
20	2.64	319	1.1e-05	1.1e-05

Visualization of the GO-analysis results

A graph of the results can also be produced. Here we visualize the three most significant nodes according to the Fisher elim algorithm in the context of the GO hierarchy.

```
showSigOfNodes(top_GO_data, score(result_top_GO_elim), firstSigNodes = 3,
               useInfo = 'def')
```



A pathway enrichment analysis using reactome

```
entrez_ids <- mapIds(hugene10sttranscriptcluster.db,
  keys = rownames(table_CD),
  keytype = "PROBEID",
  column = "ENTREZID")
```

```
reactome_enrich <- enrichPathway(gene = entrez_ids[DE_genes_CD],
                                universe = entrez_ids[c(DE_genes_CD,
                                                         back_genes)],
                                organism = "human",
                                pvalueCutoff = 0.05,
                                qvalueCutoff = 0.9,
                                readable = TRUE)
```



```

reactome_enrich@result$Description <- paste0(str_sub(
  reactome_enrich@result$Description, 1, 20),
  "...")

head(summary(reactome_enrich))[1:6]

```

ID	Description	GeneRatio	BgRatio
R-HSA-8978868	R-HSA-8978868 Fatty acid metabolis...	54/1380	104/5934
R-HSA-6785807	R-HSA-6785807 Interleukin-4 and 13...	41/1380	77/5934
R-HSA-6783783	R-HSA-6783783 Interleukin-10 signa...	22/1380	31/5934
R-HSA-556833	R-HSA-556833 Metabolism of lipids...	154/1380	454/5934
R-HSA-380108	R-HSA-380108 Chemokine receptors ...	19/1380	26/5934
R-HSA-1474244	R-HSA-1474244 Extracellular matrix...	74/1380	188/5934

	pvalue	p.adjust
R-HSA-8978868	1.463e-10	1.494e-07
R-HSA-6785807	9.450e-09	4.824e-06
R-HSA-6783783	2.208e-08	7.516e-06
R-HSA-556833	4.904e-08	1.252e-05
R-HSA-380108	9.824e-08	2.006e-05
R-HSA-1474244	3.898e-07	6.633e-05

Note that we trimmed pathway names to 20 characters.

Visualizing the reactome based analysis results

The *reactomePA* package offers nice visualization capabilities. The top pathways can be displayed as a bar chart that displays all categories with a p-value below the specified cutoff (Figure 16).

```
barplot(reactome_enrich)
```

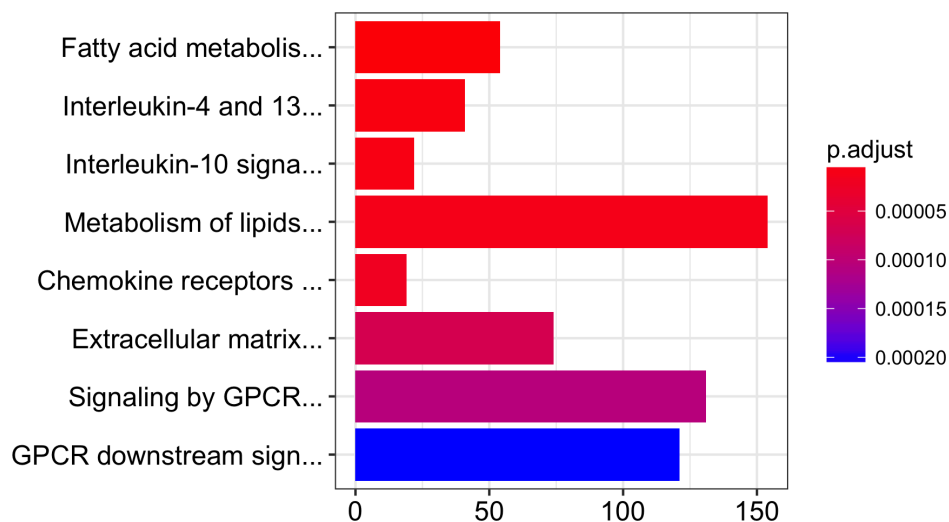


Figure 16. Enriched Reactome pathways and their p-values as a bar chart.

The “enrichment map” from the package *enrichplot* displays the results of the enrichment analysis as a graph, where the color represents the p-value of the pathway and the edge-thickness (that is the line connecting two pathways) is proportional to the number of overlapping genes between two pathways.

```
emapplot(reactome_enrich, showCategory = 10)
```

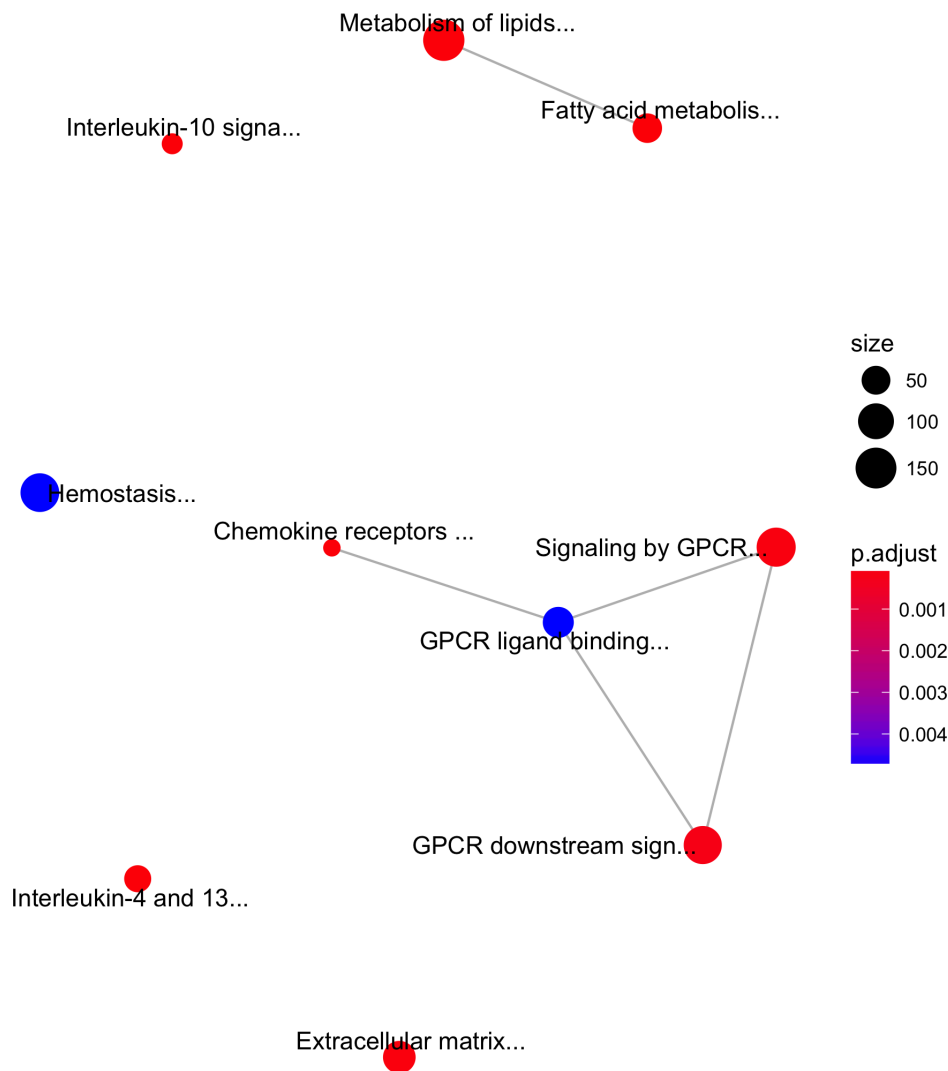


Figure 17. Enriched Reactome pathways enrichment results as a graph.

Again, the graph in Figure 17 shows pathways related to signalling and immune response.

The package *clusterProfiler* [25] can also perform these analyses using downloaded KEGG data. Furthermore, the package *EnrichmentBrowser* [26] additionally offers network-based enrichment analysis of individual pathways. This allows the mapping of the expression data at hand to known regulatory interactions.

Session information

As the last part of this document, we call the function *sessionInfo*, which reports the version numbers of R and all the packages used in this session. It is good practice to always keep such a record of this as it will help to track down what has happened in case an R script ceases to work or gives different results because the functions have been changed in a newer version of one of your packages. By including it at the bottom of a script, your reports will become more reproducible.

The session information should also *always* be included in any emails to the Bioconductor support site along with all code used in the analysis.

```
gc()
```

	used	(Mb)	gc trigger	(Mb)	limit	(Mb)	max used	(Mb)
Ncells	11597817	619.4	2.310e+07	1234	NA	2.310e+07	1234	
Vcells	384279464	2931.9	1.166e+09	8896	16384	1.166e+09	8895	

```
length(getLoadedDLLs())
```

```
[1] 99
```

```
sessionInfo()
```

```
R version 3.5.0 (2018-04-23)
Platform: x86_64-apple-darwin15.6.0 (64-bit)
Running under: macOS High Sierra 10.13.4

Matrix products: default
BLAS: /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/Vers
LAPACK: /Library/Frameworks/R.framework/Versions/3.5/Resources/lib/libRlapack.dylib
```

```
locale:
```

```
[1] en_IE.UTF-8/en_IE.UTF-8/en_IE.UTF-8/C/en_IE.UTF-8/en_IE.UTF-8
```

```
attached base packages:
```

```
[1] grid      parallel  stats4    stats      graphics  grDevices  utils
[8] datasets  methods   base
```

```
other attached packages:
```

```
[1] magick_1.9                      pd.hg.u95a_3.12.0
[3] githubinstall_0.2.2            bindrcpp_0.2.2
[5] DOSE_3.7.0                     reshape2_1.4.3
[7] knitr_1.20                     BiocWorkflowTools_1.6.1
[9] maEndToEnd_1.0.3              Rgraphviz_2.24.0
[11] tidyr_0.8.1                   RCurl_1.95-4.10
[13] bitops_1.0-6                  openxlsx_4.1.0
[15] clusterProfiler_3.8.1         ReactomePA_1.24.0
[17] limma_3.36.1                  multcomp_1.4-8
[19] TH.data_1.0-8                 MASS_7.3-49
[21] survival_2.41-3              DAAG_1.22
[23] mvtnorm_1.0-7                 pheatmap_1.0.10
[25] biomaRt_2.36.1                genefilter_1.62.0
[27] topGO_2.32.0                  SparseM_1.77
[29] graph_1.58.0                  stringr_1.3.1
[31] arrayQualityMetrics_3.36.0     ArrayExpress_1.40.0
[33] RColorBrewer_1.1-2            gplots_3.0.1
[35] LSD_4.0-0                     dplyr_0.7.5
[37] ggplot2_2.2.1                 geneplotter_1.58.0
[39] annotate_1.58.0               XML_3.98-1.11
[41] lattice_0.20-35              BiocStyle_2.8.1
[43] reactome.db_1.64.0            pd.hugene.1.0.st.v1_3.14.1
[45] DBI_1.0.0                     oligo_1.44.0
[47] oligoClasses_1.42.0           RSQlite_2.1.1
[49] Biostrings_2.48.0             XVector_0.20.0
[51] hugene10sttranscriptcluster.db_8.7.0 org.Hs.eg.db_3.6.0
[53] GO.db_3.6.0                   GenomeInfoDbData_1.1.0
[55] DO.db_2.9                     AnnotationDbi_1.42.1
[57] Biobase_2.40.0                DelayedArray_0.6.0
[59] BiocParallel_1.14.1           IRanges_2.14.10
[61] S4Vectors_0.18.2             BiocGenerics_0.26.0
[63] matrixStats_0.53.1           BiocInstaller_1.30.0
[65] devtools_1.13.5
```

```
loaded via a namespace (and not attached):
```

```
[1] tidyselect_0.2.4              htmlwidgets_1.2
[3] beadarray_2.30.0              munsell_0.4.3
[5] codetools_0.2-15              preprocessCore_1.42.0
[7] units_0.5-1                   withr_2.1.2
[9] colorspace_1.3-2              GOSeqSim_2.6.0
[11] rstudioapi_0.7                setRNG_2013.9-1
[13] labeling_0.3                  git2r_0.21.0
```

[15] hwriter_1.3.2	bit64_0.9-7
[17] rprojroot_1.3-2	xfun_0.1
[19] affxparser_1.52.0	R6_2.2.2
[21] GenomeInfoDb_1.16.0	illuminaio_0.22.0
[23] gridSVG_1.6-0	fgsea_1.6.0
[25] assertthat_0.2.0	scales_0.5.0
[27] ggraph_1.0.1	nnet_7.3-12
[29] enrichplot_1.1.2	gtable_0.2.0
[31] Cairo_1.5-9	affy_1.58.0
[33] sandwich_2.4-0	rlang_0.2.0
[35] splines_3.5.0	lazyeval_0.2.1
[37] acepack_1.4.1	checkmate_1.8.5
[39] yaml_2.1.19	backports_1.1.2
[41] qvalue_2.12.0	Hmisc_4.1-1
[43] tools_3.5.0	bookdown_0.7
[45] affyio_1.50.0	ff_2.2-14
[47] ggribes_0.5.0	Rcpp_0.12.17
[49] plyr_1.8.4	base64enc_0.1-3
[51] progress_1.1.2	zlibbioc_1.26.0
[53] purrr_0.2.4	prettyunits_1.0.2
[55] rpart_4.1-13	openssl_1.0.1
[57] viridis_0.5.1	cowplot_0.9.2
[59] zoo_1.8-1	SummarizedExperiment_1.10.1
[61] ggrepel_0.8.0	cluster_2.0.7-1
[63] tinytex_0.5	magrittr_1.5
[65] data.table_1.11.4	evaluate_0.10.1
[67] xtable_1.8-2	gcrma_2.52.0
[69] gridExtra_2.3	compiler_3.5.0
[71] tibble_1.4.2	KernSmooth_2.23-15
[73] htmltools_0.3.6	Formula_1.2-3
[75] udunits2_0.13	tweenr_0.1.5
[77] rappdirs_0.3.1	Matrix_1.2-14
[79] vsn_3.48.1	gdata_2.18.0
[81] bindr_0.1.1	igraph_1.2.1
[83] GenomicRanges_1.32.3	pkgconfig_2.0.1
[85] rvccheck_0.1.0	foreign_0.8-70
[87] foreach_1.4.4	BeadDataPackR_1.32.0
[89] affyPLM_1.56.0	digest_0.6.15
[91] rmarkdown_1.9	base64_2.0
[93] fastmatch_1.1-0	htmlTable_1.12
[95] curl_3.2	gtools_3.5.0
[97] graphite_1.26.1	jsonlite_1.5
[99] viridisLite_0.3.0	pillar_1.2.3
[101] httr_1.3.1	glue_1.2.0
[103] zip_1.0.0	UpSetR_1.3.3
[105] iterators_1.0.9	bit_1.1-13
[107] ggforce_0.1.2	stringi_1.2.2
[109] blob_1.1.1	latticeExtra_0.6-28
[111] caTools_1.17.1	memoise_1.1.0

Acknowledgements

The authors would like to thank Vladislava Milchevskaya, Julian Gehring and Mike Smith for helpful comments on and small contributions to the workflow.

This workflow draws a lot of inspiration from the Bioconductor books [27, 28] as well as Love et. al.'s workflow for gene level analysis of RNA-Seq data [29]. James W. MacDonald provided valuable information on the evolution of Affymetrix arrays in some of his posts of on the Bioconductor support site. BK would also like to thank him for some friendly personal correspondence about the annotation resources available for microarrays in Bioconductor.

Author Contributions

BK produced the first version of this article. BK implemented the main scaffold of the workflow and designed the workflow steps. SR implemented additional workflow steps and improved existing ones from version 1.

BK and SR wrote the article.

References

- [1] Orazio Palmieri, Teresa M. Creanza, Fabrizio Bossa, Orazio Palumbo, Rosalia Maglietta, Nicola Ancona, Giuseppe Corritore, Tiziana Latiano, Giuseppina Martino, Giuseppe Biscaglia, Daniela Scimeca, Michele P. De Petris, Massimo Carella, Vito Annese, Angelo Andriulli, and Anna Latiano. Genome-wide pathway analysis using gene expression data of colonic mucosa in patients with inflammatory bowel disease. *Inflammatory Bowel Diseases*, page 1, apr 2015. doi: 10.1097/mib.0000000000000370. URL <http://dx.doi.org/10.1097/MIB.0000000000000370>.
- [2] N. Kolesnikov, E. Hastings, M. Keays, O. Melnichuk, Y. A. Tang, E. Williams, M. Dylag, N. Kurbatova, M. Brandizi, T. Burdett, K. Megy, E. Pilicheva, G. Rustici, A. Tikhonov, H. Parkinson, R. Petryszak, U. Sarkans, and A. Brazma. ArrayExpress update—simplifying data submissions. *Nucleic Acids Research*, 43(D1):D1113–D1116, oct 2014. doi: 10.1093/nar/gku1057. URL <http://dx.doi.org/10.1093/nar/gku1057>.
- [3] Huber, W., Carey, V. J., Gentleman, R., Anders, S., Carlson, M., Carvalho, B. S., Bravo, H. C., Davis, S., Gatto, L., Girke, T., Gottardo, R., Hahne, F., Hansen, K. D., Irizarry, R. A., Lawrence, M., Love, M. I., MacDonald, J., Obenchain, V., Ole's, A. K., Pag'es, H., Reyes, A., Shannon, P., Smyth, G. K., Tenenbaum, D., Waldron, L., Morgan, and M. Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods*, 12(2):115–121, 2015. URL <http://www.nature.com/nmeth/journal/v12/n2/full/nmeth.3252.html>.
- [4] Benilton S Carvalho and Rafael A Irizarry. A framework for oligonucleotide microarray preprocessing. *Bioinformatics*, 26(19):2363–7, 2010. ISSN 1367-4803. doi: 10.1093/bioinformatics/btq431.
- [5] Audrey Kauffmann, Robert Gentleman, and Wolfgang Huber. arrayqualitymetrics—a bioconductor package for quality assessment of microarray data. *Bioinformatics*, 25(3):415–6, 2009.
- [6] R. A. Irizarry. Summaries of affymetrix GeneChip probe level data. *Nucleic Acids Research*, 31(4):15e–15, feb 2003. doi: 10.1093/nar/gng015. URL <http://dx.doi.org/10.1093/nar/gng015>.
- [7] B.M. Bolstad, R.A Irizarry, M. Astrand, and T.P. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185–193, jan 2003. doi: 10.1093/bioinformatics/19.2.185. URL <http://dx.doi.org/10.1093/bioinformatics/19.2.185>.
- [8] R. A. Irizarry. Exploration, normalization, and summaries of high density oligonucleotide array probe level data. *Biostatistics*, 4(2):249–264, apr 2003. doi: 10.1093/biostatistics/4.2.249. URL <http://dx.doi.org/10.1093/biostatistics/4.2.249>.
- [9] Luke C Gandolfo and Terence P Speed. Rle plots: Visualizing unwanted variation in high dimensional data. *PLoS ONE*, 2018. doi: 10.1371/journal.pone.0191629. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5798764/>.
- [10] Wolfgang Huber, Anja von Heydebreck, Holger Suetmann, Annemarie Poustka, and Martin Vingron. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics*, 18 Suppl. 1:S96–S104, 2002.
- [11] Matthew E Ritchie, Belinda Phipson, Di Wu, Yifang Hu, Charity W Law, Wei Shi, and Gordon K Smyth. limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47, 2015.
- [12] Gordon K Smyth. Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3(1):1–25, jan 2004. doi: 10.2202/1544-6115.1027. URL <http://dx.doi.org/10.2202/1544-6115.1027>.
- [13] Janine Arloth, Daniel M. Bader, Simone Röh, and Andre Altmann. Re-annotator: Annotation pipeline for microarray probe sequences. *PLOS ONE*, 10(10):e0139516, oct 2015. doi: 10.1371/journal.pone.0139516. URL <http://dx.doi.org/10.1371/journal.pone.0139516>.
- [14] Korbinian Strimmer. A unified approach to false discovery rate estimation. *BMC Bioinformatics*, 2008.
- [15] Bradley Efron, David Cox, Xiao-Li Meng, Susan Holmes, and Diana Gillooly, editors. *Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing and Prediction*. 2010. URL https://statweb.stanford.edu/~ckirby/brad/LSI/monograph_CUP.pdf.
- [16] Michael Ashburner, Catherine A. Ball, Judith A. Blake, David Botstein, Heather Butler, J. Michael Cherry, Allan P Davis, Kara Dolinski, Selina S. Dwight, Janan T. Eppig, Midori A. Harris, David P Hill, Laurie Issel-Tarver, Andrew Kasarskis, Suzanna Lewis, John C. Matese, Joel E. Richardson, Martin Ringwald, Gerald M. Rubin, and Gavin Sherlock. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, may 2000. doi: 10.1038/75556. URL <http://dx.doi.org/10.1038/75556>.
- [17] Gene Ontology Consortium. Gene ontology consortium: going forward. *Nucleic acids research*, 43(Database issue):D1049–56, January 2015. ISSN 0305-1048. doi: 10.1093/nar/gku1179. URL <http://europemc.org/articles/PMC4383973>.
- [18] L. du Plessis, N. Skunca, and C. Dessimoz. The what, where, how and why of gene ontology—a primer for bioinformaticians. *Briefings in Bioinformatics*, 12(6):723–735, feb 2011. doi: 10.1093/bib/bbr002. URL <http://dx.doi.org/10.1093/bib/bbr002>.
- [19] R. Bourgon, R. Gentleman, and W. Huber. Independent filtering increases detection power for high-throughput experiments. *Proceedings of the National Academy of Sciences*, 107(21):9546–9551, may 2010. doi: 10.1073/pnas.0914005107. URL <http://dx.doi.org/10.1073/pnas.0914005107>.

- [20] Eran Eden, Roy Navon, Israel Steinfeld, Doron Lipson, and Zohar Yakhini. GOrilla: a tool for discovery and visualization of enriched GO terms in ranked gene lists. *BMC Bioinformatics*, 10(1):48, 2009. doi: 10.1186/1471-2105-10-48. URL <http://dx.doi.org/10.1186/1471-2105-10-48>.
- [21] A. Alexa, J. Rahnenfuhrer, and T. Lengauer. Improved scoring of functional groups from gene expression data by decorrelating GO graph structure. *Bioinformatics*, 22(13):1600–1607, apr 2006. doi: 10.1093/bioinformatics/btl140. URL <http://dx.doi.org/10.1093/bioinformatics/btl140>.
- [22] David Croft, Antonio Fabregat Mundo, Robin Haw, Marija Milacic, Joel Weiser, Guanming Wu, Michael Caudy, Phani Garapati, Marc Gillespie, Maulik R. Kamdar, Bijay Jassal, Steven Jupe, Lisa Matthews, Bruce May, Stanislav Palatnik, Karen Rothfels, Veronica Shamovsky, Heeyeon Song, Mark Williams, Ewan Birney, Henning Hermjakob, Lincoln Stein, and Peter D'Eustachio. The reactome pathway knowledgebase. *Nucleic Acids Research*, 42(D1):D472–D477, nov 2013. doi: 10.1093/nar/gkt1102. URL <http://dx.doi.org/10.1093/nar/gkt1102>.
- [23] Antonio Fabregat, Steven Jupe, Lisa Matthews, Konstantinos Sidiropoulos, Marc Gillespie, Phani Garapati, Robin Haw, Bijay Jassal, Florian Korninger, Bruce May, Marija Milacic, Corina Duenas Roca, Karen Rothfels, Cristoffer Sevilla, Veronica Shamovsky, Solomon Shorser, Thawfeek Varusai, Guilherme Viteri, Joel Weiser, Guanming Wu, Lincoln Stein, Henning Hermjakob, and Peter D'Eustachio. The reactome pathway knowledgebase. *Nucleic Acids Research*, 46(D1):D649–D655, nov 2017. doi: 10.1093/nar/gkx1132. URL <https://doi.org/10.1093/nar/gkx1132>.
- [24] G. Yu, L.-G. Wang, G.-R. Yan, and Q.-Y. He. DOSE: an r/bioconductor package for disease ontology semantic and enrichment analysis. *Bioinformatics*, 31(4):608–609, oct 2014. doi: 10.1093/bioinformatics/btu684. URL <http://dx.doi.org/10.1093/bioinformatics/btu684>.
- [25] Guangchuang Yu, Li-Gen Wang, Yanyan Han, and Qing-Yu He. clusterProfiler: an r package for comparing biological themes among gene clusters. *OMICS: A Journal of Integrative Biology*, 16(5):284–287, may 2012. doi: 10.1089/omi.2011.0118. URL <http://dx.doi.org/10.1089/omi.2011.0118>.
- [26] Ludwig Geistlinger, Gergely Csaba, and Ralf Zimmer. Bioconductor's EnrichmentBrowser: seamless navigation through combined results of set- & network-based enrichment analysis. *BMC Bioinformatics*, 17(1), jan 2016. doi: 10.1186/s12859-016-0884-1. URL <http://dx.doi.org/10.1186/s12859-016-0884-1>.
- [27] Robert Gentleman, Vincent J. Carey, Wolfgang Huber, Rafael Irizarry, and Sandrine Dudoit, editors. *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*. Springer, 2005. URL <http://www.bioconductor.org/help/publications/books/bioinformatics-and-computational-biology-solutions>.
- [28] Florian Hahne, Wolfgang Huber, Robert Gentleman, and Seth Falcon. *Bioconductor Case Studies*. Use R. Springer, 2008. URL <http://www.bioconductor.org/help/publications/books/bioconductor-case-studies>.
- [29] Michael I. Love, Simon Anders, Vladislav Kim, and Wolfgang Huber. RNA-seq workflow: gene-level exploratory analysis and differential expression. *F1000Research*, oct 2015. doi: 10.12688/f1000research.7035.1. URL <http://dx.doi.org/10.12688/f1000research.7035.1>.