

Comparing methods for differential expression analysis of RNAseq data with the *compcodeR* package

Charlotte Soneson

November 18, 2013

Contents

1	Introduction	2
2	A sample workflow	2
2.1	Simulating data	2
2.2	Performing differential expression analysis	3
2.3	Comparing results from several differential expression methods	4
3	Using your own data	6
4	Providing your own differential expression code	6
5	The format of the data and result objects	6
5.1	The data object	6
5.2	The result object	8
6	The evaluation/comparison methods	9
6.1	ROC (one replicate/all replicates)	9
6.2	AUC	10
6.3	Type I error	10
6.4	FDR	11
6.5	FDR as a function of expression level	11
6.6	TPR	11
6.7	False discovery curves (one replicate/all replicates)	12
6.8	Fraction significant	12
6.9	Overlap, one replicate	13
6.10	Sorensen index, one replicate	13
6.11	MA plot	14
6.12	Spearman correlation	14
6.13	Score distribution vs number of outliers	15
6.14	Score distribution vs average expression level	15
7	Session info	16

1 Introduction

compcodeR is an *R* package that provides an interface to several popular methods for differential expression analysis of RNAseq data and contains functionality for comparing the outcomes and performances of several differential expression methods applied to the same data set. The package also contains a function for generating synthetic RNAseq counts, using the simulation framework described in more detail in

C Soneson and M Delorenzi: A comparison of methods for differential expression analysis of RNA-seq data. *BMC Bioinformatics* 2013, **14**:91.

This vignette provides a tutorial on how to use the different functionalities of the *compcodeR* package.

Currently, the differential expression interfaces provided in the package are restricted to comparisons between two conditions. However, many of the comparison functions are more general and can also be applied to test results from other contrast types, as well as to test results from other data types than RNAseq.

Since *compcodeR* provides interfaces to differential expression analysis methods implemented in other *R* packages, take care to cite the appropriate references if you use any of the interface functions (see the reference manual for more information). Also be sure to check the code that was used to run the differential expression analysis, using e.g. the `generateCodeHTMLs` function (see Section 2.3 for more information) to make sure that parameters etc. agree with your intentions.

2 A sample workflow

This section contains a sample workflow showing the main functionalities of the *compcodeR* package. We start by generating a synthetic count data set (Section 2.1), to which we then apply three different differential expression methods (Section 2.2). Finally, we compare the outcome of the three methods and generate a report summarizing the results (Section 2.3).

2.1 Simulating data

The simulations are performed following the description by (Soneson and Delorenzi, 2013). As an example, we use the `generateSyntheticData` function to generate a synthetic count data set containing 12,500 genes and two groups of 5 samples each, where 10% of the genes are simulated to be differentially expressed between the two groups (equally distributed between up- and downregulated in group 2 compared to group 1). Furthermore, the counts for all genes are simulated from a Negative Binomial distribution with the same dispersion in the two sample groups, and no outlier counts are introduced. We filter the data set by excluding only the genes with zero counts in all samples (i.e., those for which the total count is 0). This simulation setting corresponds to the one denoted B_{625}^{625} in (Soneson and Delorenzi, 2013). The following code creates the simulated data set and saves it to a file named "B_625_625_5spc_rep11.rds".

```
library(compcodeR)
```

```
B_625_625 <- generateSyntheticData(dataset = "B_625_625",  
  n.vars = 12500, samples.per.cond = 5, n.diffexp = 1250,  
  repl.id = 1, seqdepth = 1e+07, fraction.upregulated = 0.5,  
  between.group.diffdisp = FALSE, filter.threshold.total = 1,  
  filter.threshold.mediancpm = 0, fraction.non.overdispersed = 0,  
  output.file = "B_625_625_5spc_rep11.rds")
```

The `summarizeSyntheticDataSet` function provides functionality to check some aspects of the simulated data by generating a report summarizing the parameters that were used for the simulation, as well as including some diagnostic plots. In the current version, the report contains two MA-plots, showing the estimated average expression level and the log-fold change for all genes, indicating either the truly differentially expressed genes or the total number of outliers introduced for each gene. It also shows the log-fold changes estimated from the simulated data versus those underlying the simulation.

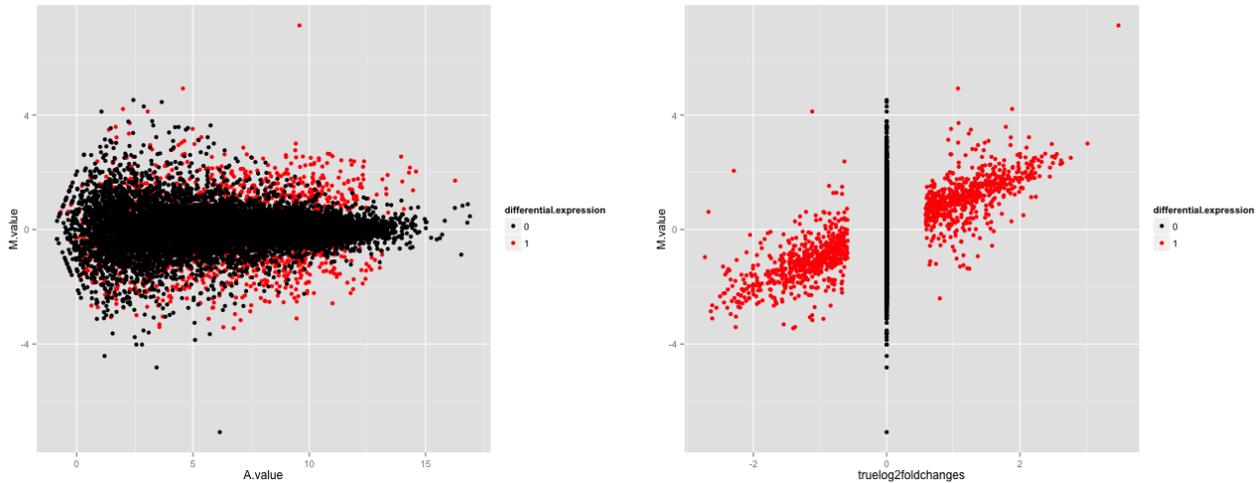


Figure 1: Example figures from the summarization report generated for a simulated data set. The left panel shows an MA plot, with the genes colored by the true differential expression status. The right panel shows the relationship between the true log-fold changes between the two sample groups underlying the simulation, and the estimated log-fold changes based on the simulated counts. Also here, the genes are colored by the true differential expression status.

```
summarizeSyntheticDataSet(data.set = "B_625_625_5spc_repl1.rds",
  output.filename = "B_625_625_5spc_repl1_datacheck.html")
```

Figure 1 shows two of the figures generated by this function. The left panel shows an MA plot with the genes colored by the true differential expression status. The right panel shows the relationship between the true log-fold changes between the two sample groups underlying the simulation, and the estimated log-fold changes based on the simulated counts.

2.2 Performing differential expression analysis

We will now apply some of the interfaced differential expression methods to find genes that are differentially expressed between the two conditions in the simulated data set. This is done through the `runDiffExp` function, which is the main interface for performing differential expression analyses in *compcodeR*. The code below applies three differential expression methods to the data set generated above: the `voom` transformation from the *limma* package (combined with *limma* for differential expression), the exact test from the *edgeR* package, and a regular t-test applied directly on the count level data.

```
runDiffExp(data.file = "B_625_625_5spc_repl1.rds", result.extent = "voom.limma",
  Rmdfunction = "voom.limma.createRmd", output.directory = ".",
  norm.method = "TMM")
runDiffExp(data.file = "B_625_625_5spc_repl1.rds", result.extent = "edgeR.exact",
  Rmdfunction = "edgeR.exact.createRmd", output.directory = ".",
  norm.method = "TMM", trend.method = "movingave", disp.type = "tagwise")
runDiffExp(data.file = "B_625_625_5spc_repl1.rds", result.extent = "ttest",
  Rmdfunction = "ttest.createRmd", output.directory = ".",
  norm.method = "TMM")
```

The code needed to perform each of the analyses is provided in the `*.createRmd` functions. To obtain a list of all available `*.createRmd` functions (and hence of the available differential expression methods), we can use the `listcreateRmd()` function

```
listcreateRmd()  
  
## [1] "baySeq.createRmd"  
## [2] "DESeq.GLM.createRmd"  
## [3] "DESeq.nbinom.createRmd"  
## [4] "DESeq2.createRmd"  
## [5] "DSS.createRmd"  
## [6] "EBSeq.createRmd"  
## [7] "edgeR.exact.createRmd"  
## [8] "edgeR.GLM.createRmd"  
## [9] "HTSDiff.createRmd"  
## [10] "logcpm.limma.createRmd"  
## [11] "NBPSeq.createRmd"  
## [12] "NOISeq.prenorm.createRmd"  
## [13] "SAMseq.createRmd"  
## [14] "sqrtcpm.limma.createRmd"  
## [15] "TCC.createRmd"  
## [16] "ttest.createRmd"  
## [17] "voom.limma.createRmd"  
## [18] "voom.ttest.createRmd"  
## [19] "vst.limma.createRmd"  
## [20] "vst.ttest.createRmd"
```

You can also apply your own differential expression method to the simulated data (see Section 4).

2.3 Comparing results from several differential expression methods

Once we have obtained the results of the differential expression analyses (either by the methods interfaced by *compcodeR* or in other ways, see Section 4), we can compare the results and generate an HTML report summarizing the results from the different methods from many different aspects (see Section 6 for an overview of the comparison metrics). The comparison is performed by the `runComparisonGUI` function, to which we provide a list of directories containing our result files, and the directory where the final report will be generated. Since the three result files above were saved in the current working directory, we can run the following code to perform the comparison:

```
runComparisonGUI(input.directories = ".", output.directory = ".",  
recursive = FALSE)
```

This opens a graphical user interface (Figure 2) where we can select which of the data files available in the input directories that should be included as a basis for the comparison, and which comparisons to perform (see Section 6 for an overview). Through this interface, we can also set p-value cutoffs for significance and other parameters that will govern the behaviour of the comparison.

Important! When you have modified a value in one of the textboxes (p-value cutoffs etc.), press Return on your keyboard to confirm the assignment of the new value to the parameter. You can also always check in the resulting comparison report that the correct values were recognized and used for the comparisons.

After the selections have been made, the function will perform the comparisons and generate an HTML report, which is saved in the designated output directory and automatically named `compcodeR_report_<timestamp>.html`. Please note that depending on the number of compared methods and the number of included data sets, this step may take several minutes. *compcodeR* will notify you when the report is ready (with a message **Done!** in the console).

The comparison function will also generate a subdirectory called `compcodeR_code`, where the R code used to perform each of the comparisons is detailed in HTML reports, and another subdirectory called `compcodeR_figure`, where the

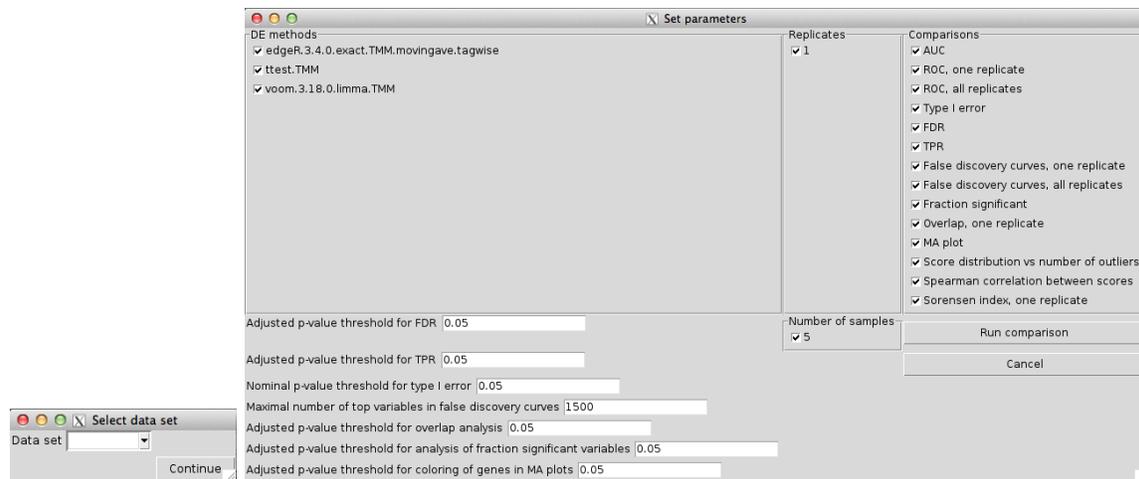


Figure 2: Screenshot of the graphical user interface used to select data set (left) and set parameters (right) for the comparison of differential expression methods. The available choices for the “Data set”, “DE methods”, “Number of samples” and “Replicates” are automatically generated from the result objects available in the designated input directories. Only one data set can be used for the comparison. In the lower part of the window we can set (adjusted) p-value thresholds for each comparison method separately. For example, we can evaluate the true FDR at one adjusted p-value threshold, and estimate the TPR for another adjusted p-value threshold. We can also set the maximal number of top-ranked variables that will be considered for the false discovery curves.

plots generated in the comparison are saved. The code HTML reports can also be generated manually for a given result file (given that the code component is present in the result object, see Section 5.2), using the function `generateCodeHTMLs`. For example, to generate a report containing the code that was run to perform the t-test above, as well as the output from the *R* console, we can write:

```
generateCodeHTMLs("B_625_625_5spc_repl1_ttest.rds", ".")
```

These reports are useful to check that there were no errors or warnings when running the differential expression analyses.

The use of the GUI requires that the *rpanel* package is installed, which in turns requires that *BWidget* is installed on your computer. Moreover, if you wish to compare a large number of methods or if you are working on a very small screen, the use of the GUI may be troublesome. If you wish to circumvent the graphical user interface, it is possible to call the function `runComparison` directly. In this case, you need to supply the function with a list of result files to use as a basis for the comparison. Moreover, you need to provide a list of parameters (p-value thresholds, the differential expression methods to include in the comparison, etc.). The following code provides an example, given the data generated above.

```
file.table <- data.frame(input.files = c("B_625_625_5spc_repl1_voom.limma.rds",
  "B_625_625_5spc_repl1_ttest.rds", "B_625_625_5spc_repl1_edgeR.exact.rds"),
  stringsAsFactors = FALSE)
parameters <- list(incl.nbr.samples = NULL, incl.replicates = NULL,
  incl.dataset = "B_625_625", incl.de.methods = NULL,
  fdr.threshold = 0.05, tpr.threshold = 0.05, typeI.threshold = 0.05,
  ma.threshold = 0.05, fdc.maxvar = 1500, overlap.threshold = 0.05,
  fracsign.threshold = 0.05, comparisons = c("auc", "fdr",
  "tpr", "ma", "correlation"))
runComparison(file.table = file.table, parameters = parameters,
  output.directory = ".")
```

By setting `incl.nbr.samples`, `incl.replicates` and `incl.de.methods` to `NULL`, we ask *compcodeR* to include all data sets provided in the `file.table`. By providing a vector of values, it is possible to limit the selection to a subset of the provided files. Please note that the values given to `incl.replicates` and `incl.nbr.samples` are matched with values of the `info.parameters$repl.id` and `info.parameters$samples.per.cond` slots in the `data/result` objects, and that the values given to `incl.de.methods` are matched with values stored in the `method.names$full.name` slot of the result objects. If the values do not match, the corresponding result object will not be considered in the comparison. Consult the package manual for the full list of comparison methods available for use with the `runComparison` function.

3 Using your own data

The *compcodeR* package provides a straightforward function for simulating count data (`generateSyntheticData`). However, it is easy to apply the interfaced differential expression methods to your own data, given that it is formatted in a way that is compatible with the functions (see Section 5.1 below for a description of the data format).

4 Providing your own differential expression code

The *compcodeR* package provides an interface for calling some of the most commonly used differential expression methods developed for RNAseq data. However, it is easy to incorporate your own favorite method. In principle, this can be done in one of two ways:

1. Write a `XXX.createRmd` function (where `XXX` corresponds to your method), similar to the ones provided in the package, which creates a `.Rmd` file containing the code that is run to perform the differential expression analysis. Then call this function through the `runDiffExp` function. When implementing your function, make sure that the output is structured as described in Section 5.2. The `*.createRmd` functions provided in the package take the following input arguments:
 - `data.path` – the path to the `.rds` file containing the data object to which the differential expression will be applied.
 - `result.path` – the path to the `.rds` file where the result object will be stored.
 - `codefile` – the name of the code (with extension `.Rmd`) where the code will be stored.
 - any arguments for setting parameters of the differential expression analysis.
2. Run the differential expression analysis completely outside the *compcodeR* package, and save the result in the format described in Section 5.2.

5 The format of the data and result objects

This section details the format of the data and result objects generated and used by the *compcodeR* package. The format guidelines below must be followed if you apply the functions in the package to a data set of your own, or to differential expression results generated outside the package. Note that for most of the functionality of the package, the objects should be saved separately to files with a `.rds` extension, and the path of this object is provided to the functions.

5.1 The data object

The data object used by *compcodeR* is a list with the following named components:

- `count.matrix` [class `matrix`] (**mandatory**) – the count matrix, with rows representing genes and columns representing samples.
- `sample.annotations` [class `data.frame`] (**mandatory**) – sample annotations. Each row corresponds to one sample, and each column to one annotation. The data objects generated within the *compcodeR* package have two annotations:
 - `condition` [class `character` or `numeric`] (**mandatory**) – the class for each sample. Currently the differential expression implementations in the package supports only two-group comparisons, hence the `condition` should have exactly two unique values.
 - `depth.factor` [class `numeric`] – the depth factor for each sample. This factor, multiplied by the sequencing depth, corresponds to the target library size of the sample when simulating the counts.

- `info.parameters` [class list] – a list detailing the parameter values that have been used for the simulation.
 - `dataset` [class character] (**mandatory**) – the name of the data set.
 - `samples.per.cond` [class numeric] – the number of samples in each of the two conditions.
 - `n.diffexp` [class numeric] – the number of genes that are simulated to be differentially expressed between the two conditions.
 - `repl.id` [class numeric] – a replicate number, which can be set to differentiate between different instances generated with the exact same simulation settings.
 - `seqdepth` [class numeric] – the "base" sequencing depth that was used for the simulations. For each sample, it is modified by multiplication with a value sampled uniformly between `minfact` and `maxfact` to generate the actual sequencing depth for the sample.
 - `minfact` [class numeric] – the lower bound on the values used to multiply the `seqdepth` to generate the actual sequencing depth for the individual samples.
 - `maxfact` [class numeric] – the upper bound on the values used to multiply the `seqdepth` to generate the actual sequencing depth for the individual samples.
 - `fraction.upregulated` [class numeric] – the fraction of the differentially expressed genes that were simulated to be upregulated in condition 2 compared to condition 1. Must be in the interval $[0, 1]$. The remaining differentially expressed genes are simulated to be downregulated in condition 2.
 - `between.group.diffdisp` [class logical] – whether the counts from the two conditions were simulated with different dispersion parameters.
 - `filter.threshold.total` [class numeric] – the filter threshold that is applied to the total count across all samples. All genes for which the total count is below this number have been excluded from the data set in the simulation process.
 - `filter.threshold.mediancpm` [class numeric] – the filter threshold that is applied to the median count per million (cpm) across all samples. All genes for which the median cpm is below this number have been excluded from the data set in the simulation process.
 - `fraction.non.overdispersed` [class numeric] – the fraction of the genes in the data set that are simulated according to a model without overdispersion (i.e., a Poisson model).
 - `random.outlier.high.prob` [class numeric] – the fraction of extremely high random outlier counts introduced in the simulated data sets. Please consult (Soneson & Delorenzi 2013) for a detailed description of the different types of outliers.
 - `random.outlier.low.prob` [class numeric] – the fraction of extremely low random outlier counts introduced in the simulated data sets. Please consult (Soneson & Delorenzi 2013) for a detailed description of the different types of outliers.
 - `single.outlier.high.prob` [class numeric] – the fraction of extremely high single outlier counts introduced in the simulated data sets. Please consult (Soneson & Delorenzi 2013) for a detailed description of the different types of outliers.
 - `single.outlier.low.prob` [class numeric] – the fraction of extremely low single outlier counts introduced in the simulated data sets. Please consult (Soneson & Delorenzi 2013) for a detailed description of the different types of outliers.
 - `effect.size` [class numeric] – the degree of differential expression, i.e., a measure of the minimal effect size.
 - `uID` [class character] (**mandatory**) – a unique identification number given to each data set. For the data sets simulated with *compcodeR*, it consists of a sequence of 10 randomly selected alpha-numeric characters.
- `filtering` [class character] – a summary of the filtering that has been applied to the data.
- `variable.annotations` [class data.frame] – annotations for each of the variables in the data set. Each row corresponds to one variable, and each column to one annotation. No annotation is mandatory, however some of them are necessary if the differential expression results for the data are going to be used for certain comparison tasks. The data sets simulated within *compcodeR* have the following named variable annotations:
 - `truedispersions.S1` [class numeric] – the true dispersions used in the simulations of the counts for the samples in condition 1.
 - `truedispersions.S2` [class numeric] – the true dispersions used in the simulations of the counts for the samples in condition 2.
 - `truemeans.S1` [class numeric] – the true mean values used in the simulations of the counts for the samples in condition 1.

- `truemeans.S2` [class numeric] – the true mean values used in the simulations of the counts for the samples in condition 2.
- `n.random.outliers.up.S1` [class numeric] – the number of extremely high random outliers introduced for each gene in condition 1.
- `n.random.outliers.up.S2` [class numeric] – the number of extremely high random outliers introduced for each gene in condition 2.
- `n.random.outliers.down.S1` [class numeric] – the number of extremely low random outliers introduced for each gene in condition 1.
- `n.random.outliers.down.S2` [class numeric] – the number of extremely low random outliers introduced for each gene in condition 2.
- `n.single.outliers.up.S1` [class numeric] – the number of extremely high single outliers introduced for each gene in condition 1.
- `n.single.outliers.up.S2` [class numeric] – the number of extremely high single outliers introduced for each gene in condition 2.
- `n.single.outliers.down.S1` [class numeric] – the number of extremely low single outliers introduced for each gene in condition 1.
- `n.single.outliers.down.S2` [class numeric] – the number of extremely low single outliers introduced for each gene in condition 2.
- `M.value` [class numeric] – the estimated log₂-fold change between conditions 1 and 2 for each gene. These values were estimated using the *edgeR* package.
- `A.value` [class numeric] – the estimated average expression in conditions 1 and 2 for each gene. These values were estimated using the *edgeR* package.
- `truelog2foldchanges` [class numeric] – the “true” log₂-fold changes between conditions 1 and 2 for each gene, based on the parameters used for simulation.
- `upregulation` [class numeric] – a binary annotation (0/1) indicating which genes are simulated to be upregulated in condition 2 compared to condition 1.
- `downregulation` [class numeric] – a binary annotation (0/1) indicating which genes are simulated to be downregulated in condition 2 compared to condition 1.
- `differential.expression` [class numeric] (**mandatory** for many comparisons, such as the computation of false discovery rates, true positive rates, ROC curves, false discovery curves etc.) – a binary annotation (0/1) indicating which genes are simulated to be differentially expressed in condition 2 compared to condition 1. In other words, the sum of `upregulation` and `downregulation`.

To apply the functions of the package to a data object of the type detailed above, it needs to be saved to a file with extension `.rds`. To save the object data to the file `saveddata.rds`, simply type

```
saveRDS(data, "saveddata.rds")
```

5.2 The result object

The result objects generated by applying one of the differential expression methods interfaced through *compcodeR* is an extension of the data object described in the previous paragraph. In addition to the components described above, the result objects generated by the differential expression methods in this package contain the following named components:

- `analysis.date` [class character] – the date and time when the differential expression analysis was performed.
- `package.version` [class character] – the version of the packages used in the differential expression analysis.
- `method.names` [class list] (**mandatory**) – a list containing the names of the differential expression method, which should be used to identify the results in the comparison. Contains two components:
 - `short.name` [class character] – a short name, used for convenience.
 - `full.name` [class character] – a fully identifying name of the differential expression method, including e.g. version numbers and parameter values. In the method comparisons, the results will be grouped based on the `full.name`.
- `code` [class character] – the code that was used to run the differential expression analysis, in R markdown (`.Rmd`) format.
- `result.table` [class data.frame] (**mandatory**) – a data frame containing the results of the differential expression analysis. Each row corresponds to one gene, and each column to a value generated by the analysis. The precise

columns will depend on the method applied. The following columns are used by at least one of the methods interfaced by *compcodeR*:

- `pvalue` [class numeric] – the nominal p-values.
- `adjpvalue` [class numeric] – p-values adjusted for multiple comparisons.
- `logFC` [class numeric] – estimated log-fold changes between the two conditions.
- `score` [class numeric] (**mandatory**) – the score that will be used to rank the genes in order of significance. Note that **high scores always signify differential expression, that is, a strong association with the predictor**. For example, for methods returning a nominal p-value the score is generally obtained as $1 - \text{pvalue}$.
- `FDR` [class numeric] – the false discovery rate estimate.
- `posterior.DE` [class numeric] – the posterior probability of differential expression.
- `prob.DE` [class numeric] – The conditional probability of differential expression.
- `lfdr` [class numeric] – the local false discovery rate.
- `statistic` [class numeric] – a test statistic from the differential expression analysis.
- `dispersion.S1` [class numeric] – dispersion estimate in condition 1.
- `dispersion.S2` [class numeric] – dispersion estimate in condition 2.

For many of the comparison methods, the naming of the result columns is important. For example, the p-value column must be named `pvalue` in order to be recognized by the comparison method computing type I error. Similarly, either an `adjpvalue` or an `FDR` column must be present in order to apply the comparison methods requiring adjusted p-value/FDR cutoffs. If both are present, the `adjpvalue` column takes precedence over the `FDR` column.

6 The evaluation/comparison methods

This section provides an overview of the methods that are provided by *compcodeR* for comparing differential expression results obtained by different methods. The selection of which methods to apply is made through a graphical user interface that is opened when the `runComparisonGUI` function is called (see Figure 2). Alternatively, the selected methods can be supplied to the `runComparison` function directly, to circumvent the GUI.

6.1 ROC (one replicate/all replicates)

This method computes a ROC curve for either a single representative among the replicates of a given data set, or for all replicates (in separate plots). The ROC curves are generated by plotting the true positive rate (TPR, on the y-axis) versus the false positive rate (FPR, on the x-axis) when varying the cutoff on the `score` (see the description of the result object above). A good ranking method gives a ROC curve which passes close to the upper left corner of the plot, while a bad method gives a ROC curve closer to the diagonal. Calculation of the ROC curves requires that the differential expression status of each gene is provided (see the description of the data object in Section 5.1). Figure 3 shows an example of ROC curves for a single replicate.

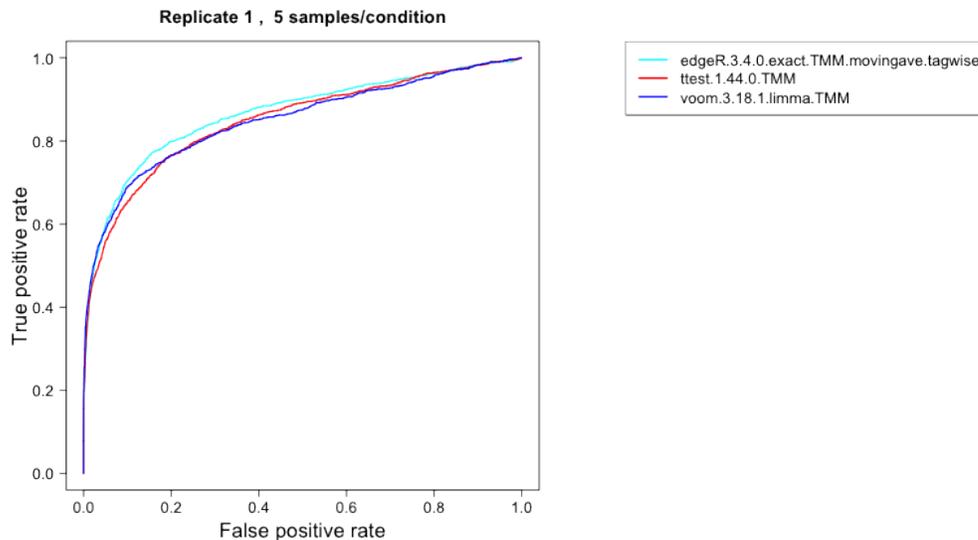


Figure 3

6.2 AUC

This comparison method computes the area under the ROC curve (see section 6.1) and represents the result in boxplots, where each box summarizes the results for one method across all replicates of a data set. A good ranking method gives to a large value of the AUC. This requires that the differential expression status of each gene is provided (see the description of the data object in Section 5.1). Figure 4 shows an example.

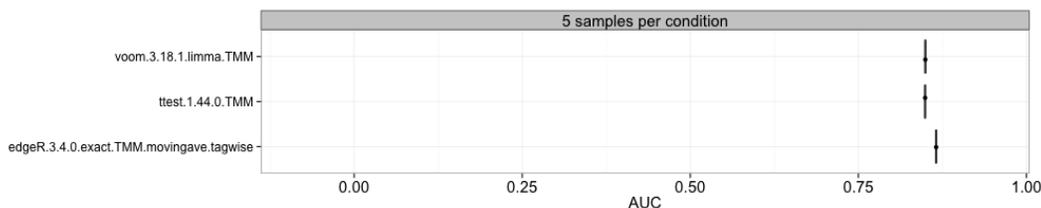


Figure 4

6.3 Type I error

This approach computes the type I error (the fraction of genes that are truly non-differentially expressed but yet called significant at a given nominal p-value threshold) for each method and sample size separately, and represent in boxplots, where each box summarizes the results for one method across all replicates of a data set. This requires that the differential expression status of each gene is provided (see the description of the data object above) and that the `pvalue` column is present in the `result.table` of the included result objects (see the description of the result object above). An example is given in Figure 5. The dashed vertical line represents the imposed nominal p-value threshold, and we wish that the observed type I error is lower than this threshold.

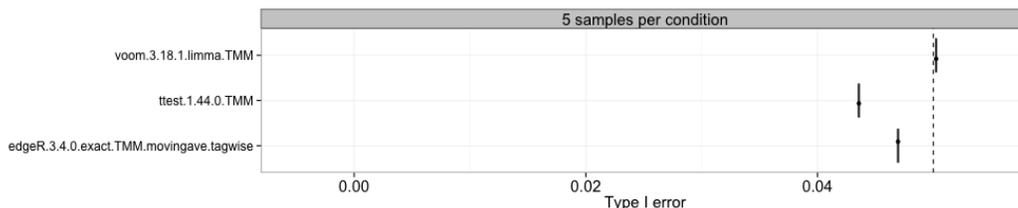


Figure 5

6.4 FDR

Here, we compute the observed false discovery rate (the fraction of the genes called significant that are truly non-differentially expressed at a given adjusted p-value/FDR threshold) for each method and sample size separately, and represent the estimates in boxplots, where each box summarizes the results for one method across all replicates of a data set. This requires that the differential expression status of each gene is provided (see the description of the data object above) and that the `adjpvalue` or `FDR` column is present in the `result.table` of the included result objects (see the description of the result object above). An example is given in Figure 6. The dashed vertical line represents the imposed adjusted p-value threshold (that is, the level at which we wish to control the false discovery rate).

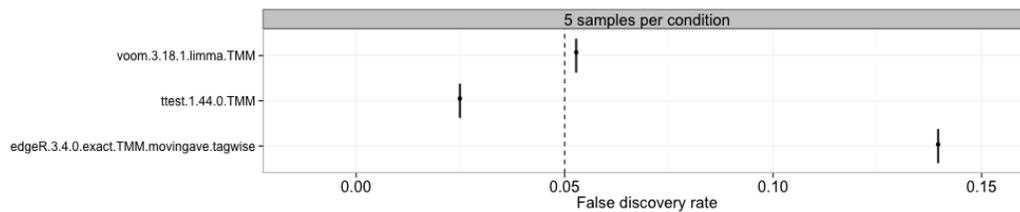


Figure 6

6.5 FDR as a function of expression level

Instead of looking at the overall false discovery rate, this method allows us to study the FDR as a function of the average expression level of the genes. For each data set, the average expression levels are binned into 10 bins of equal size (i.e., each containing 10% of the genes), and the FDR is computed for each of them. The results are shown by means of boxplots, summarizing the results across all replicates of a data set. Figure 7 shows an example.

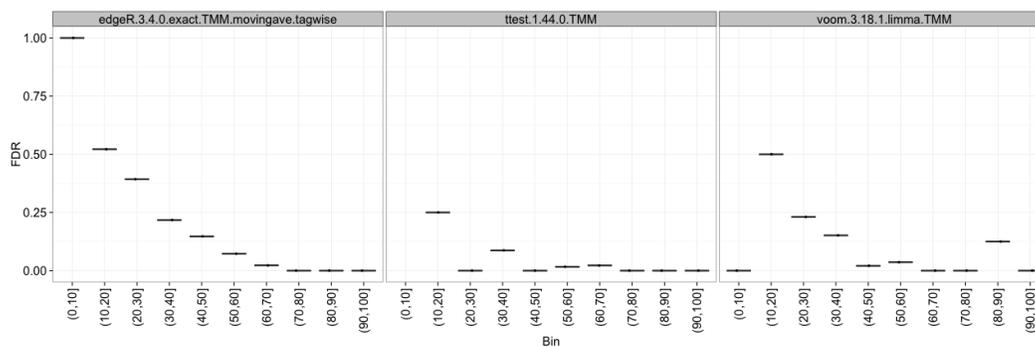


Figure 7

6.6 TPR

With this approach, we compute the observed true positive rate (the fraction of the truly differentially expressed genes that are called significant at a given adjusted p-value/FDR threshold) for each method and sample size separately, and represent the estimates in boxplots, where each box summarizes the results for one method across all replicates of a data set. This requires that the differential expression status of each gene is provided (see the description of the data object above) and that the `adjpvalue` or `FDR` column is present in the `result.table` of the included result objects (see the description of the result object above). Figure 8 shows an example.

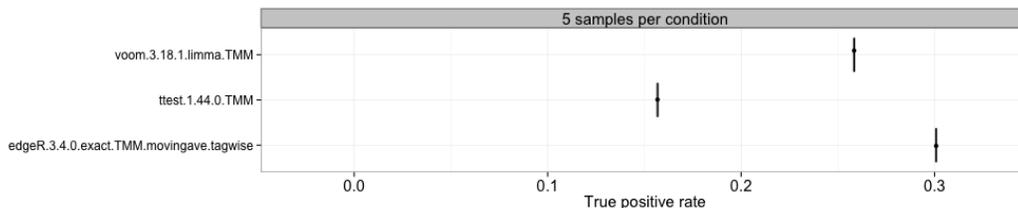


Figure 8

6.7 False discovery curves (one replicate/all replicates)

This choice plots the false discovery curves, depicting the number of false discoveries (i.e., truly non-differentially expressed genes) that are encountered when traversing the ranked list of genes ordered in decreasing order by the score column of the `result.table` (see the description of the result object above). As for the ROC curves, the plots can be made for a single replicate or for all replicates (each in a separate figure). Figure 9 shows an example for a single replicate. A well-performing method is represented by a slowly rising false discovery curve (in other words, few false positives among the top-ranked genes).

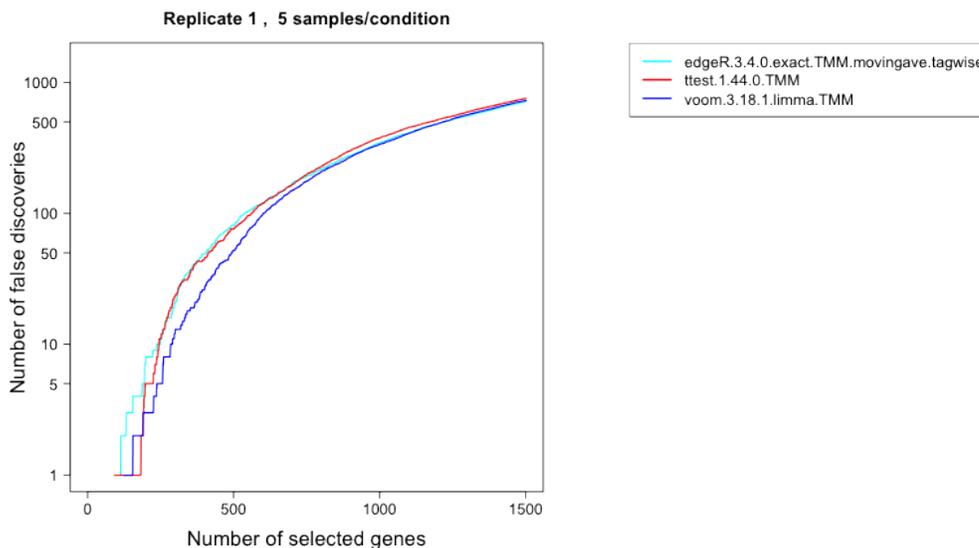


Figure 9

6.8 Fraction significant

We can also compute the fraction of the genes that are called significant at a given adjusted p-value/FDR threshold for each method and sample size separately, and represent the estimates in boxplots, where each box summarizes the results for one method across all replicates of a data set. This requires that the `adjpvalue` or `FDR` column is present in the `result.table` of the included result objects (see the description of the result object above). An example is given in Figure 10.

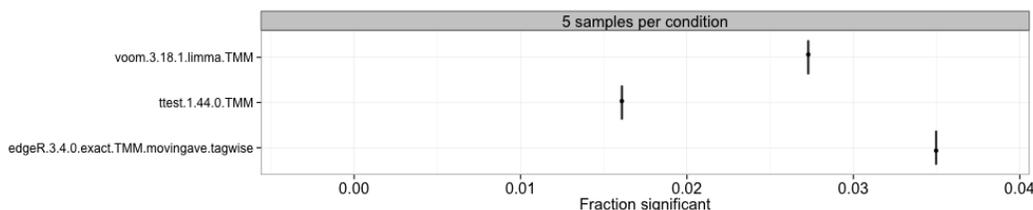


Figure 10

6.9 Overlap, one replicate

For each pair of methods, this approach computes the overlap between the sets of genes called differentially expressed by each of them at a given adjusted p-value/FDR threshold. Only one representative of the replicates of a data set is used. This requires that the `adjpvalue` or `FDR` column is present in the `result.table` of the included result objects (see the description of the result object above). Since the overlap depends on the number of genes called differentially expressed by the different methods, a normalized overlap measure (the Sorensen index) is also provided (see Section 6.10 below). The results are represented as a table of the sizes of the overlaps between each pair of methods.

6.10 Sorensen index, one replicate

For each pair of methods, this approach computes a normalized pairwise overlap value between the sets of genes called differentially expressed by each of them at a given adjusted p-value/FDR threshold. Only one representative of the replicates of a data set is used. This requires that the `adjpvalue` or `FDR` column is present in the `result.table` of the included result objects (see the description of the result object above). The Sorensen index (also called the Dice coefficient) for two sets A and B is given by

$$S(A, B) = \frac{2|A \cap B|}{|A| + |B|},$$

where $|\cdot|$ denotes the cardinality of a set (the number of elements in the set). The results are represented in the form of a table, and also as a colored heatmap, where the color represents the degree of overlap between the sets of differentially expressed genes found by different methods. An example is shown in Figure 11.

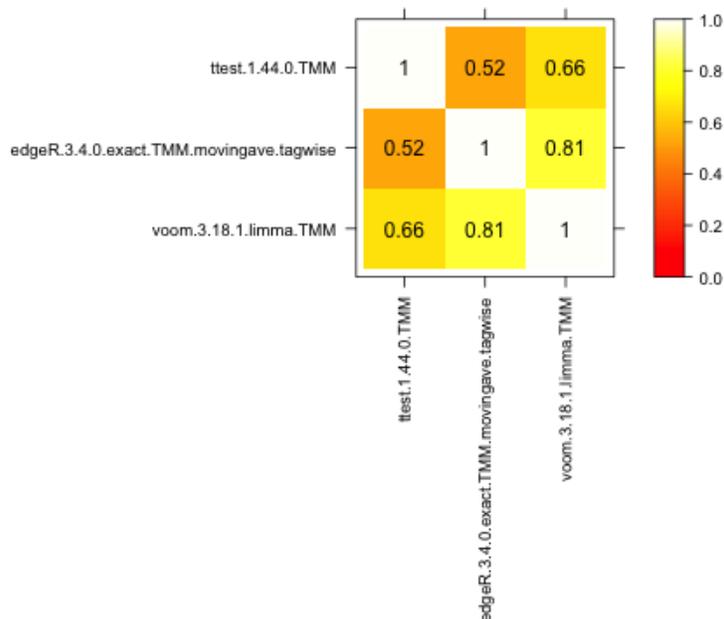


Figure 11

6.11 MA plot

With this approach we can construct MA-plots (one for each differential expression method) for one replicate of a data set, depicting the estimated log-fold change (on the y-axis) versus the estimated average expression (on the x-axis). The genes that are called differentially expressed are marked with color. Figure 12 shows an example.

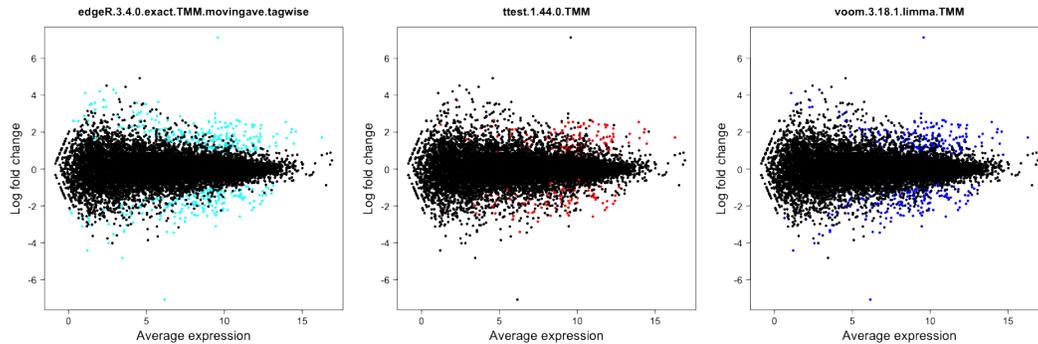


Figure 12

6.12 Spearman correlation

This approach computes the pairwise Spearman correlation between the gene scores obtained by different differential expression methods (the score component of the `result.table`, see the description of the result object above). The correlations are visualized by means of a color-coded table and used to construct a dissimilarity measure for hierarchical clustering of the differential expression methods. Examples are given in Figures 13 and 14.

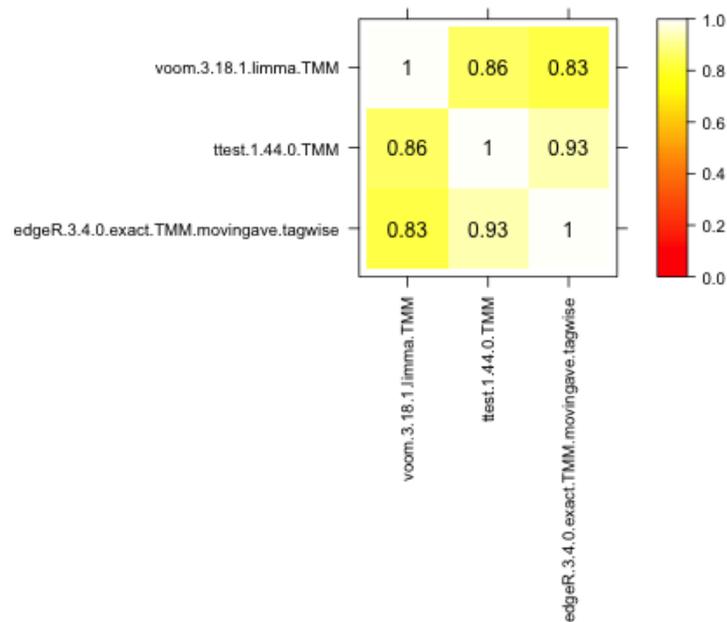


Figure 13

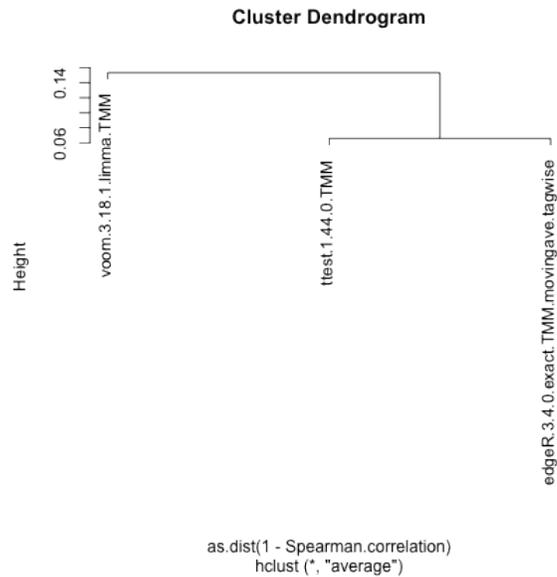


Figure 14

6.13 Score distribution vs number of outliers

This method plots the distribution of the gene scores obtained by different differential expression methods (the score component of the `result.table`, see the description of the result object above) as a function of the number of outliers imposed for the genes. The distributions are visualized by means of violin plots. Figure 15 shows an example, where there are no outliers introduced. Recall that a high score corresponds to more significant genes.

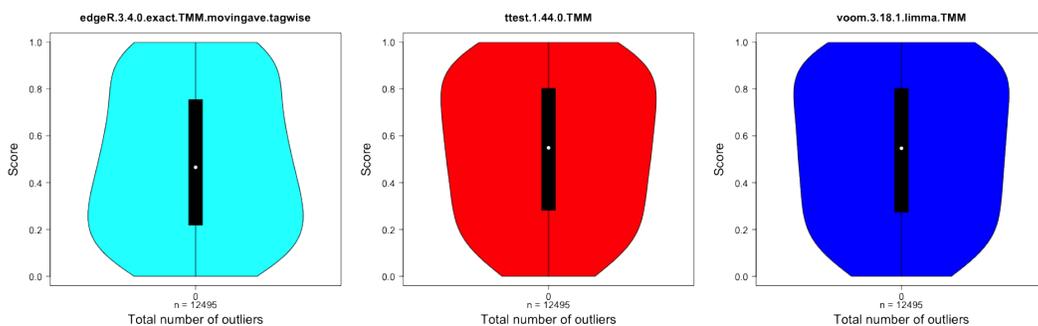


Figure 15

6.14 Score distribution vs average expression level

This method plots the gene scores (the score component of the `result.table`, see the description of the result object above) as a function of the average expression level of the gene. An example is shown in Figure 16. Recall that high scores correspond to more significant genes. The colored line shows the trend in the relationship between the two variables by means of a loess fit.

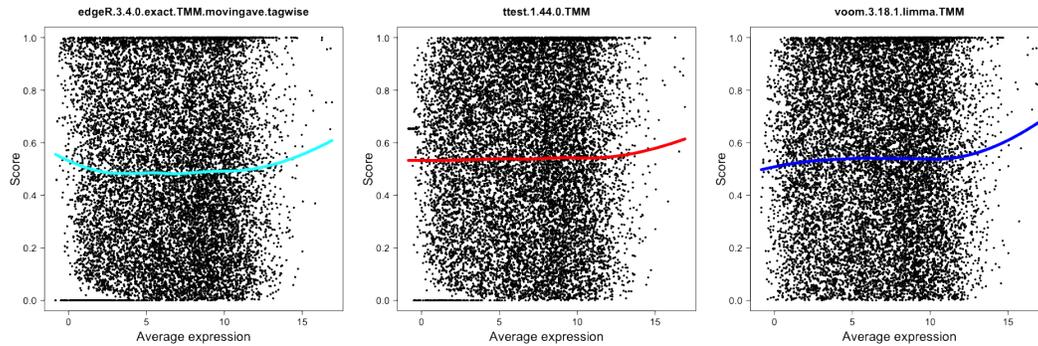


Figure 16

7 Session info

- R version 3.0.2 (2013-09-25), x86_64-apple-darwin10.8.0
- Locale: sv_SE.UTF-8/sv_SE.UTF-8/sv_SE.UTF-8/C/sv_SE.UTF-8/sv_SE.UTF-8
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, tcltk, utils
- Other packages: caTools 1.16, compcodeR 0.1.0, edgeR 3.4.0, gdata 2.13.2, ggplot2 0.9.3.1, gplots 2.12.1, gtools 3.1.1, KernSmooth 2.23-10, knitr 1.5, lattice 0.20-24, limma 3.18.3, markdown 0.6.3, MASS 7.3-29, modeest 2.1, ROCR 1.0-5, sm 2.2-5.3, stringr 0.6.2, vioplot 0.2
- Loaded via a namespace (and not attached): BiocStyle 1.0.0, bitops 1.0-6, colorspace 1.2-4, dichromat 2.0-0, digest 0.6.3, evaluate 0.5.1, formatR 0.10, gtable 0.1.2, highr 0.3, labeling 0.2, munsell 0.4.2, plyr 1.8, proto 0.3-10, RColorBrewer 1.0-5, reshape2 1.2.2, scales 0.2.3, tools 3.0.2