GOexpress: Visualise microarray and RNAseq data using gene ontology annotations

Kévin Rue-Albrecht, Paul A. McGettigan, Belinda Hernández, David A. Magee, Nicolas C. Nalpas, Andrew C. Parnell, Stephen V. Gordon, and David E. MacHugh

October 17, 2014

Contents

1 Introduction

1.1 The origin and purpose of `GOexpress`

The idea leading to the `GOexpress` R package emerged from a set of plotting functions I regularly copy-pasted across various complex multifactorial transcriptomics studies from both microarray and RNA-seq platforms. Those functions were repeatedly used to visualise the expression profile of genes across groups of samples, to annotate technical gene identifiers from both microarray and RNA-seq platforms (i.e. probesets, Ensembl gene identifiers) with their associated gene name, and to evaluate the clustering of samples based on genes participating in a common cellular function or location (i.e. gene ontology). While developing the `GOexpress` package and discussing its features with colleagues and potential users, a few more features were added, to enhance and complement the initial functions, leading to the present version of the package.

Complex multifactorial experiments have become the norm in many research fields, thanks to the decrease in cost of high-throughput transcriptomics platforms and the barcoding/multiplexing of samples on the RNA-seq platform. While much effort has been (correctly!) spent on the development of adequate statistical frameworks for the processing of raw expression data, much of the genewise visualisation is left to the end-user. However, data summarisation and visualisation can be a daunting task in multifactorial experiments, or require large amounts of copy-pasting to investigate the expression profile of a handful or genes and cellular pathways.

Tested on multiple RNA-seq and microarray datasets, `GOexpress` offers an extendable set of data-driven plotting functions readily applicable to the output of widely used analytic packages estimating (differential) gene expression. Once the initial analysis and filtering of `GOexpress` results is complete — literaly two command lines —, each gene and gene ontology is accessible by a single line of code to produce high-quality graphics. In short, the `GOexpress` package is a software package developed based on real experimental datasets to ease the visualisation and interpretation of multifactorial transcriptomics data by bioinformaticians and biologists, while striving to keep it a simple, fast, and intuitive toolkit.

Notably, the use of the `biomaRt` package enables `GOexpress` to support and annotate gene expression matrix from any species and any microarray platform present in the Ensembl BioMart:
`http://www.ensembl.org/biomart/martview`.

1.2 Purpose of this document

This User's Guide was intended as a helpful description of the main features implemented in the `GOexpress` package, as well as a tutorial taking the user through a typical analysis pipeline that `GOexpress` was designed for. While an example usage will be provided for each function of the package, the many arguments of each function cannot realistically be demonstrated in this Guide, and we kindly ask users to also read the individual R vignettes accompanying the corresponding package functions for further details.

2 Before you start

2.1 Installation

Installing `GOexpress` should be as easy as running the two lines below:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("GOexpress")
```

Installation issues should be reported to the Bioconductor mailing list.

2.2 Getting help

The `GOexpress` package is still at an early stage of development and may require some fine-tuning or yet undetected bug fixes. Please contact the maintainer with a copy of the error message and the command run.

Despite our efforts to repeatedly test the `GOexpress` package on in-house datasets of both microarray and RNA-seq platforms, and of human and bovine origin, many of the species and microarrays present in the Ensembl BioMart have not been tested yet. Do contact the package maintainer if you feel that the error is on our part:

```
> maintainer("GOexpress")
```

```
[1] "Kevin Rue-Albrecht <kevin.rue@ucdconnect.ie>"
```

Interesting suggestions for additional package functions, or improvement of existing ones are most welcome and may be implemented when time allows. Alternatively, we also encourage users to fork the GitHub repository of the project, develop and test their own feature(s), and finally generate a pull request to integrate it to the original repository (`https://github.com/kevinrue/GOexpress`).

As for all Bioconductor packages, the Bioconductor support site is the best place to seek advice with a large and active community of Bioconductor users. More detailed information is available at `http://www.bioconductor.org/help/support`.

2.3 Citing `GOexpress`

The work underlying `GOexpress` has not been formally published yet. A manuscript is in preparation and will be submitted to a bioinformatics journal in due course. In the meantime, users of the package are encouraged to cite the `GOexpress` package using the `citation` method of the `utils` package, as follows:

```
> citation(package="GOexpress")

To cite package 'GOexpress' in publications use:

  Kevin Rue-Albrecht (2014). GOexpress: Visualise microarray and RNAseq
  data using gene ontology annotations. R package version 1.0.1.
  https://github.com/kevinrue/GOexpress-release

A BibTeX entry for LaTeX users is

  @Manual{,
    title = {GOexpress: Visualise microarray and RNAseq data using gene ontology annotatio
    author = {Kevin Rue-Albrecht},
    year = {2014},
    note = {R package version 1.0.1},
    url = {https://github.com/kevinrue/GOexpress-release},
  }
```

3 Quick start

3.1 Input data

Despite their different underlying technologies, microarray and RNA-seq analytic pipelines typically yield a matrix measuring the expression level of each gene in each sample. Commonly, this expression matrix will be filtered to retain only genes qualified as "informative" (e.g. > 1 cpm in at least N replicates; N being the number of replicates for a given set of experimental conditions); genes lowly expressed are removed to limit the False Discovery Rate (FDR) of differentially expressed genes induced by the larger variability of expression at the lower end of the dynamic range.

`GOexpress` requires this prefiltered normalised expression matrix to be accompanied by an `AnnotatedDataFrame` object of the `Biobase` package providing phenotypic information for each of those samples (e.g. unique identifier, treatment, time-point). `GOexpress` expects those two variables in an `ExpressionSet` container of the `Biobase` package, both simplifying the manipulation of the data and, most importantly, ensuring interoperatibility with other packages that handle Bioconductor `ExpressionSet` objects. The other fields of the `ExpressionSet` container may be left empty as `GOexpress` does not currently access them.

To use the analytic part of the `GOexpress` package, the phenotypic dataframe — `phenodata` slot of the `ExpressionSet` — must contain at least one column containing an experimental factor made of two or more levels in the strict meaning of "factor" and "levels" in the R programming language. The above `ExpressionSet` and the name of the column containing such a factor are the minimal two input variables required for the `GO_analyse` method to work. Additional arguments may be required, in particular for microarray datasets, but those are discussed later in section 3.2.

In the examples below, we will use the sample dataset `AlvMac` provided with the package and made of a subset of 100 bovine Ensembl gene identifiers (rows)

measured in 117 samples (columns). This sample data includes a data-frame detailing a number of phenotypic information fields describing each sample.

Let us load those two variables in the workspace, and load the `GOexpress` packge as well:

```
> data(AlvMac) # import the training dataset
> library(GOexpress) # load the GOexpress package
```

Now, the expression matrix and phenotypic data of the `ExpressionSet` container can be accessed using dedicated methods from the `Biobase` package:

```
> exprs(AlvMac)[1:5,1:5] # Subset of the expression data
```

```
                  N1178_CN_24H N1178_CN_2H N1178_CN_48H N1178_CN_6H
ENSBTAG00000020733     1.575566  1.72415180    0.9304433   1.7318466
ENSBTAG00000006240     7.328855  7.05983676    7.5340557   6.8728357
ENSBTAG00000007616     0.317048  0.12274232   -0.4276224  -0.3592233
ENSBTAG00000002921     3.911831  4.31849784    4.3629270   4.3152549
ENSBTAG00000023938     1.307170 -0.08024698    0.9304433   0.6832164
                  N1178_MB_24H
ENSBTAG00000020733    1.8419688
ENSBTAG00000006240    7.5894739
ENSBTAG00000007616   -0.3215346
ENSBTAG00000002921    3.9548748
ENSBTAG00000023938    1.4408185
```

```
> head(pData(AlvMac)) # Subset of the phenotypic information
```

```
                     File       Sample Animal Treatment Time  Group Timepoint
N1178_CN_24H N1178_CN_24H N1178_CN_24H  N1178        CN  24H CN_24H        24
N1178_CN_2H   N1178_CN_2H  N1178_CN_2H  N1178        CN   2H  CN_2H         2
N1178_CN_48H N1178_CN_48H N1178_CN_48H  N1178        CN  48H CN_48H        48
N1178_CN_6H   N1178_CN_6H  N1178_CN_6H  N1178        CN   6H  CN_6H         6
N1178_MB_24H N1178_MB_24H N1178_MB_24H  N1178        MB  24H MB_24H        24
N1178_MB_2H   N1178_MB_2H  N1178_MB_2H  N1178        MB   2H  MB_2H         2
```

An advantage of the `ExpressionSet` container is that it takes care of the compatibility between the expression matrix and the phenotypic information data-frame. For instance, it will check that samples names do not differ between expression matrix and phenotypic information.

However, importantly , it will not check one key requirement of `GOexpress`, namely that the expression matrix must have row names made of unique and supported feature identifiers (i.e. microarray probeset or Ensembl gene identifier) which will be later queried against the Ensembl BioMart server to fetch gene and gene ontology annotations. Users can visually inspect that adequate row names are used in the expression matrix:

```
> head(rownames(exprs(AlvMac))) # Subset of gene identifiers
```

```
[1] "ENSBTAG00000020733" "ENSBTAG00000006240" "ENSBTAG00000007616"
[4] "ENSBTAG00000002921" "ENSBTAG00000023938" "ENSBTAG00000011304"
```

Note that in the training dataset, the `Time` column of `pData(targets)` is an R factor while the `Timepoint` column is a numeric vector. The fomer is useful for grouping the samples for the analysis, while the latter is better suited to plot gene expression profiles while respecting the relative distance between the time-points. See section 3.7.1 for examples using of a numeric value or a factor as the variable of the X-axis.

3.2 Main analysis

In this example, we search for GO terms enriched in genes that cluster best the samples according their `Treatment` level. But first, let us make sure that the `Treatment` column of `pData(targets)` is indeed an R factor:

```
> is.factor(AlvMac$Treatment) # assertion test

[1] TRUE

> AlvMac$Treatment # visual inspection

  [1] CN CN CN CN MB MB MB MB TB TB TB TB CN CN CN CN MB MB MB MB TB TB TB TB CN
 [26] CN CN CN MB MB MB MB TB TB TB TB CN CN CN CN MB MB MB MB TB TB TB TB CN CN
 [51] CN CN MB MB MB MB TB TB TB TB CN CN CN CN MB MB MB MB TB TB TB TB CN CN CN
 [76] MB MB MB TB TB TB CN CN CN CN MB MB MB MB TB TB TB TB CN CN CN CN MB MB MB
[101] MB TB TB TB TB CN CN CN CN MB MB MB MB TB TB TB TB
Levels: CN MB TB
```

In this case, it is already a properly formatted factor. If that was not the case, the following line of code would convert the column to an R factor and allow to continue the analysis (note that in some cases, it may be preferrable to order the different levels of a factor, for an example see factor `Time`):

```
> AlvMac$Treatment <- factor(AlvMac$Treatment)
```

Now, we use the random forest statistical framework to score each gene on its ability to cluster samples from different treatments separately. Then the method will score each GO term by aggregating the average rank of its associated genes:

```
> AlvMac_results <- GO_analyse(eSet = AlvMac, f = "Treatment")

First feature identifier in dataset: ENSBTAG00000020733
Looks like Ensembl gene identifier.
Loading detected dataset btaurus_gene_ensembl ...
Object of class 'Mart':
 Using the ensembl BioMart database
 Using the btaurus_gene_ensembl dataset
Fetching ensembl_gene_id/go_id mappings from BioMart ...
Fetching GO_terms description from BioMart ...
Analysis using method randomForest on factor Treatment for 100
genes. This may take a few minutes ...
ntree      OOB      1      2      3
  100:  52.99% 28.21% 69.23% 61.54%
  200:  52.99% 28.21% 64.10% 66.67%
```

```
  300:  52.99% 28.21% 58.97% 71.79%
  400:  58.12% 30.77% 64.10% 79.49%
  500:  54.70% 30.77% 58.97% 74.36%
  600:  57.26% 30.77% 64.10% 76.92%
  700:  58.97% 33.33% 64.10% 79.49%
  800:  57.26% 33.33% 58.97% 79.49%
  900:  55.56% 30.77% 58.97% 76.92%
 1000:  54.70% 30.77% 56.41% 76.92%
Fetching gene description from BioMart ...
Merging score into result table ...
```

At this stage, it is a good idea to save the result variable into an R data-file using the `save` function. Firstly, because the stochastic aspect of the sampling approach implemented by the `randomForest` package may return slightly different scores in each run (as opposed to the ANOVA scores). Secondly, because the BioMart web service may occasionally be temporarily down, disabling this essential step of the analysis.

Note that the above command does not specify which species the data originated from. As mentioned in the output messages, the first feature identifier in the expression matrix was used to determine the corresponding species and type of data. This is a fairly straightforward process for Ensembl gene identifiers (e.g. in the prefix 'ENSBTA', 'BT' indicates *Bos taurus*).

However, it can be more difficult to identify the microarray used to obtain a certain dataset, as many Affymetrix chips contain probesets named with the pattern 'AFFX.*'. In cases where the microarray platform cannot be detected automatically, we recommend users to use the `microarray` argument of the `GO_analyse` method. The list of valid values for the microarray argument is available in the `microarray2dataset` data frame which can be loaded in the workspace using:

```
> data(microarray2dataset)
```

This step is the longest step in the pipeline, but merely takes a couple of minutes to analyse approximately 12,000 genes in 117 samples with default parameters of 1,000 trees build using approximately 220 random predictor genes per iteration on a standard Ubuntu 12.04 server.

The output variable of the analysis summarises the parameters of the analysis and can easily be browsed with standard R methods:

```
> names(AlvMac_results) # Data slot names

[1] "GO"      "mapping" "genes"   "factor"  "method"  "subset"  "ntree"
[8] "mtry"

> str(AlvMac_results) # Details of data slots

List of 8
 $ GO     :'data.frame':        13044 obs. of  7 variables:
  ..$ go_id         : chr [1:13044] "GO:0070427" "GO:0004534" "GO:0033091" "GO:0070673" ..
  ..$ ave_rank      : num [1:13044] 5.5 8 9 9 22 ...
  ..$ ave_score     : num [1:13044] 2.51 1.78 1.598 1.598 0.778 ...
```

```
  ..$ total_count   : num [1:13044] 2 1 1 1 1 1 1 1 8 1 ...
  ..$ data_count    : num [1:13044] 2 1 1 1 1 1 1 1 7 1 ...
  ..$ name_1006     : chr [1:13044] "nucleotide-binding oligomerization domain containing
  ..$ namespace_1003: chr [1:13044] "biological_process" "molecular_function" "biological_
 $ mapping:'data.frame':        188237 obs. of  2 variables:
  ..$ gene_id: chr [1:188237] "ENSBTAG00000020495" "ENSBTAG00000020495" "ENSBTAG0000002049
  ..$ go_id  : chr [1:188237] "GO:0005515" "GO:0006661" "GO:1900027" "GO:0032587" ...
 $ genes  :'data.frame':        100 obs. of  4 variables:
  ..$ Score             : num [1:100] 4.7 3.42 2.84 2.35 2.33 ...
  ..$ Rank              : int [1:100] 1 2 3 4 5 6 7 8 9 10 ...
  ..$ external_gene_name: chr [1:100] "TNIP3" "BIKBA" "PIK3AP1" "IL17" ...
  ..$ description       : chr [1:100] "TNFAIP3 interacting protein 3 [Source:HGNC Symbol;A
 $ factor : chr "Treatment"
 $ method : chr "randomForest"
 $ subset : NULL
 $ ntree  : num 1000
 $ mtry   : Named num 20
  ..- attr(*, "names")= chr "Features"
```

> head(AlvMac_results$GO, n=5) # Ranked table of GO terms (subset)

```
          go_id ave_rank ave_score total_count data_count
11021 GO:0070427      5.5 2.5097118           2          2
1506  GO:0004534      8.0 1.7795628           1          1
6491  GO:0033091      9.0 1.5979435           1          1
11109 GO:0070673      9.0 1.5979435           1          1
8453  GO:0045204     22.0 0.7783044           1          1
                                                                    name_1006
11021 nucleotide-binding oligomerization domain containing 1 signaling pathway
1506                                          5'-3' exoribonuclease activity
6491                      positive regulation of immature T cell proliferation
11109                                            response to interleukin-18
8453                                               MAPK export from nucleus
         namespace_1003
11021 biological_process
1506  molecular_function
6491  biological_process
11109 biological_process
8453  biological_process
```

> head(AlvMac_results$genes, n=5) # Ranked table of genes (subset)

```
                     Score Rank external_gene_name
ENSBTAG00000047107 4.702878    1              TNIP3
ENSBTAG00000016683 3.421480    2              BIKBA
ENSBTAG00000019872 2.840264    3            PIK3AP1
ENSBTAG00000002150 2.346941    4               IL17
ENSBTAG00000012522 2.331399    5             ZNF283

ENSBTAG00000047107
ENSBTAG00000016683 Bos taurus nuclear factor of kappa light polypeptide gene enhancer in B
```

```
ENSBTAG00000019872                                                    ph
ENSBTAG00000002150                                                    Bo
ENSBTAG00000012522
```

```
> head(AlvMac_results$mapping) # Gene to gene ontology mapping table (subset)
```

```
             gene_id      go_id
1 ENSBTAG00000020495 GO:0005515
2 ENSBTAG00000020495 GO:0006661
3 ENSBTAG00000020495 GO:1900027
4 ENSBTAG00000020495 GO:0032587
5 ENSBTAG00000020495 GO:0019902
6 ENSBTAG00000020495 GO:0035091
```

3.3 Filtering of results

In the above raw results of the analysis, all three types (i.e. namespaces) of GO terms are merged in a single ranked table. It is however possible to extract the GO terms corresponding to a single namespace.

```
> BP <- subset_scores(result = AlvMac_results,
+     namespace = "biological_process")
> MF <- subset_scores(result = AlvMac_results,
+     namespace = "MF") # alias for "molecular_function"
> CC <- subset_scores(result = AlvMac_results,
+     namespace = "CC") # cellular_component
```

Importantly, an early-identified bias of the scoring function is that GO terms associated with fewer genes are favored at the top of the ranking table. This is due to the fact that it is much easier for a group of 5 genes (e.g. "`B cell apoptotic process`") to have an high average rank than it is for a group of 6,000 genes (e.g. "`protein binding`"). Indeed, the highest possible average rank of 5 genes is 3 while it is 3,000 for a group of 6,000 genes.

However, in our experience, this bias has some advantages. First, it implicitely favors specific and well-defined GO terms (e.g. "`negative regulation of T cell apoptotic process`") as opposed to vague and uninformative GO terms (e.g. "`cytoplasm`"). Secondly, we observed many top-ranking GO terms associated with a single gene. Those GO terms are consequently susceptible to single-gene events and artefacts in the expression data, as opposed to GO terms with a reasonable number of associated genes. Using the above filtering function, it is straightforward to filter out those GO terms with only a handful of associated genes:

```
> BP.5 <- subset_scores(result = AlvMac_results,
+     namespace = "biological_process",
+     total = 5) # requires 5 or more associated genes
> MF.10 <- subset_scores(result = AlvMac_results,
+     namespace = "molecular_function",
+     total = 10)
> CC.15 <- subset_scores(result = AlvMac_results,
+     namespace = "cellular_component",
+     total = 15)
```

On the other hand, GO terms associated with thousands of genes have a very limited sensitivity, as they require a proportionally large number of associated genes to cluster the sample groups in order to emerge among the top ranking scores.

Furthermore, the inherent hierarchical structure and "granularity" of gene ontology terms can easily be browsed by using increasingly large values of the `total` filter. Note that this filter retains only GO terms associated with a minimal given count of genes in the Ensembl BioMart. It is also possible to use the `data` argument to filter for GO terms associated with a certain count of genes in the given expression dataset, although this approach is obviously more data-dependent and less robust.

3.4 Details of the top-ranking GO terms

Once the GO terms are scored and ranked (and filtered), the top-ranking GO terms in the filtered object are those most enriched in genes with expression levels that best cluster the predefined groups of samples based on the levels of the factor considered (`raw_results$factor`).

In this example, we list the top-10 filtered "Biological Process" GO terms extracted above and their statistics (automatically ranked by increasing average rank of their associated genes):

```
> head(BP.5$GO, n=10)
```

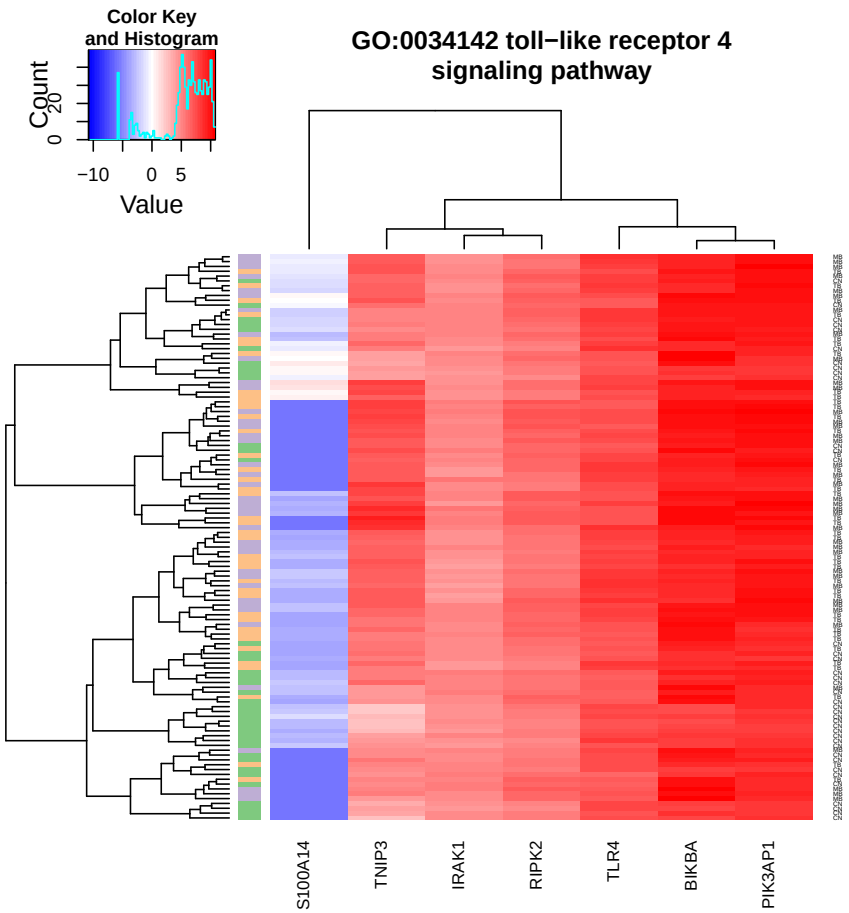
```
          go_id ave_rank ave_score total_count data_count
6715  GO:0034142   35.375 1.7813469           8          7
6709  GO:0034134   57.800 0.9792316           5          3
11025 GO:0070431   62.800 1.0038847           5          2
11267 GO:0071223   73.000 0.3885458           6          2
3797  GO:0010745   81.200 0.6842960           5          1
3881  GO:0010888   81.200 0.6842960           5          1
6330  GO:0032747   81.600 0.4693881           5          1
6353  GO:0032790   82.200 0.3845268           5          1
580   GO:0001961   82.600 0.3195887           5          1
8145  GO:0043551   83.800 0.2303027           5          1
                                                                    name_1006
6715                                      toll-like receptor 4 signaling pathway
6709                                      toll-like receptor 2 signaling pathway
11025 nucleotide-binding oligomerization domain containing 2 signaling pathway
11267                                     cellular response to lipoteichoic acid
3797       negative regulation of macrophage derived foam cell differentiation
3881                                        negative regulation of lipid storage
6330                            positive regulation of interleukin-23 production
6353                                                    ribosome disassembly
580                  positive regulation of cytokine-mediated signaling pathway
8145                       regulation of phosphatidylinositol 3-kinase activity
          namespace_1003
6715  biological_process
6709  biological_process
11025 biological_process
```

```
11267 biological_process
3797  biological_process
3881  biological_process
6330  biological_process
6353  biological_process
580   biological_process
8145  biological_process
```

3.5 Hierarchical clustering of samples based on gene expression associated with a GO term

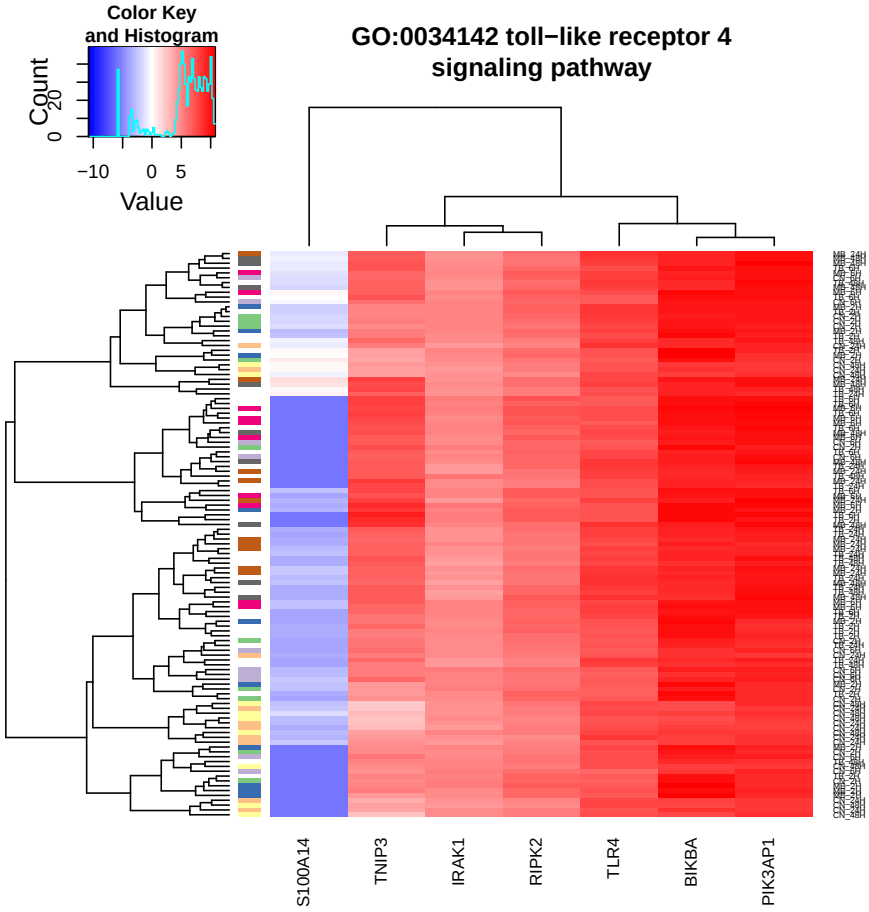
In the previous section, we identified the GO terms most enriched for genes clustering samples according to their treatment. We will now generate for the top-ranked GO term ("`toll-like receptor 4 signaling pathway`") a heatmap to visualise simulatenously the clustering of samples and the expression level of each gene in each sample:

```
> heatmap_GO(go_id = "GO:0034142", result = BP.5, eSet=AlvMac, cexRow=0.4,
+     cexCol=1, cex.main=1, main.Lsplit=30)
```



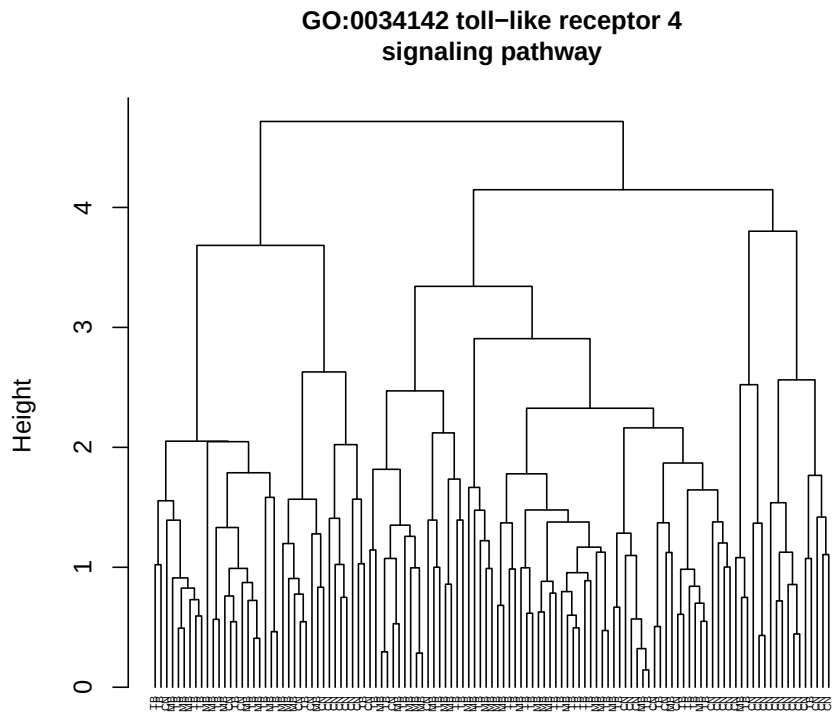
Importantly, users should be aware that the highest ranked GO term may not cluster samples at all; all GO terms are scored in the process, and even if none of them return a suitable clustering, there will still be a higest score. In this example, we can observe a group of "Control" (i.e. untreated; green color) samples clustering together at the bottom of the heatmap. Re-labelling of samples by `Group` (i.e. combination of treatment and time-point) reveals that those samples are mainly 24 and 48 hours-post-infection control samples:

```
> heatmap_GO(go_id = "GO:0034142", result = BP.5, eSet=AlvMac, f="Group",
+     cexCol=1, cex.main=1, main.Lsplit=30)
```



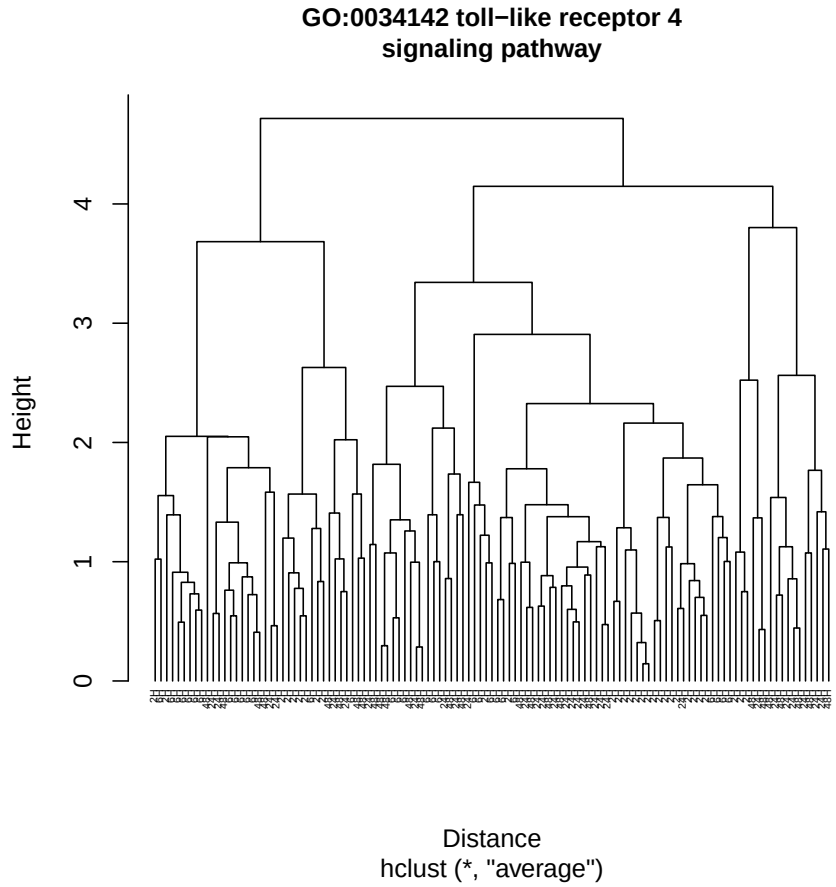
Alternatively, it is possible to focus only on the hirearchical clustering of samples. The following code will build a dendrogram clustering samples using the expression data of the subset of genes associated with the "`toll-like receptor 4 signaling pathway`" gene ontology:

```
> cluster_GO(go_id = "GO:0034142", result = BP.5, eSet=AlvMac,
+     cex.main=1, cex=0.4, main.Lsplit=30)
```



Similarly to the `heatmap_GO` method above, among the various arguments of the `cluster_GO` function, the argument `f` can be used to specify an alternative column present in `pData(targets)` to label the samples in the dendrogram according to that phenotypic information. Obviously, if the same set of genes is being used, the clustering of samples will remain the same no matter the column chosen; however, the alternative labelling of samples may reveal relevant sub-clustering of samples.

```
> cluster_GO(go_id = "GO:0034142", result = BP.5, eSet=AlvMac, f = "Time",
+     cex.main=1, cex=0.4, main.Lsplit=30)
```



3.6 Details of genes associated with a GO term

Following the identification of relevant GO terms in the above sections, users may want to have a closer look at the individual genes associated with a given GO term:

```
> table_genes(go_id = "GO:0034142", result = BP.5)
```

```
                     Score Rank external_gene_name
ENSBTAG00000047107 4.7028777    1              TNIP3
ENSBTAG00000016683 3.4214801    2              BIKBA
ENSBTAG00000016085 0.4579504   75              IRAK1
ENSBTAG00000021377 0.4969283   67            S100A14
ENSBTAG00000019872 2.8402641    3            PIK3AP1
ENSBTAG00000006240 0.7333314   25               TLR4
ENSBTAG00000015271 1.5979435    9              RIPK2
ENSBTAG00000024340        NA   NA               <NA>

ENSBTAG00000047107
ENSBTAG00000016683 Bos taurus nuclear factor of kappa light polypeptide gene enhancer in B
```

```
ENSBTAG00000016085                                    Bos taurus interleukin-1 rece
ENSBTAG00000021377                                           Bos taurus S100 calci
ENSBTAG00000019872                                                           ph
ENSBTAG00000006240                                                          Bos
ENSBTAG00000015271                                  Bos taurus receptor-interacting s
ENSBTAG00000024340
```

Note that the default behaviour of the above function is to return a table of all the genes associated with the GO term based on the annotations retrieved from the Ensembl BioMart. For obvious reasons, genes present in the BioMart but absent from the expression dataset will be absent from the score table and consequently lack annotations. It is possible to restrict the above table to only genes present in the expression dataset using the `data.only` argument.

If only the feature identifiers associated with a given GO identifier are needed, users may use the function below:

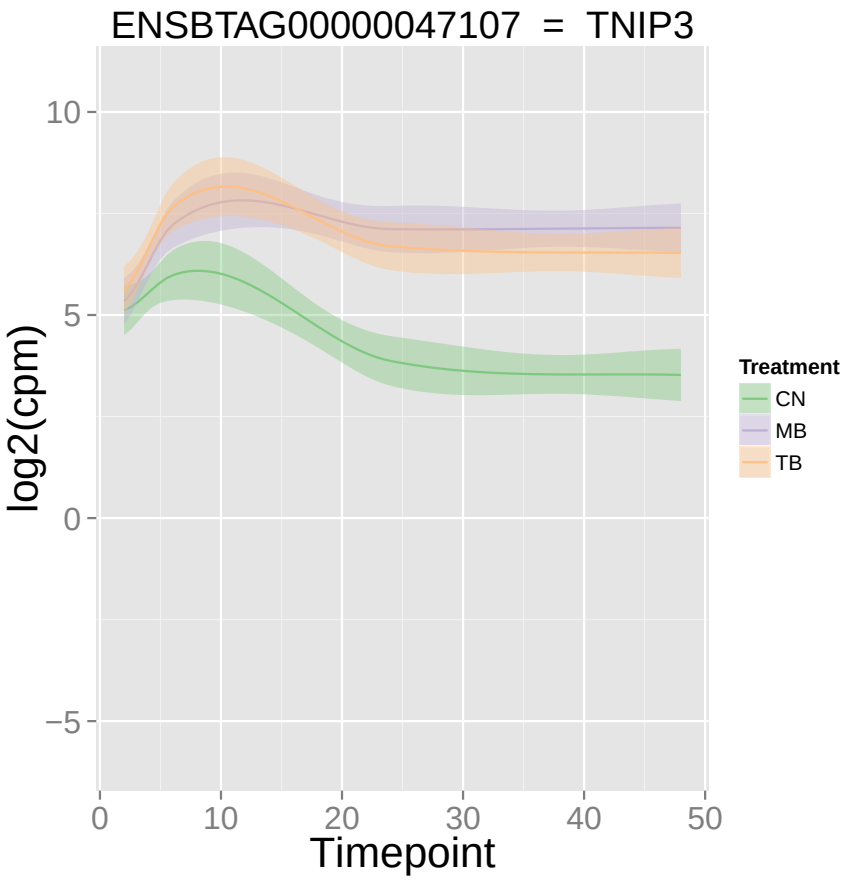
```
> list_genes(go_id = "GO:0034142", result = BP.5)

[1] "ENSBTAG00000047107" "ENSBTAG00000016683" "ENSBTAG00000016085"
[4] "ENSBTAG00000021377" "ENSBTAG00000019872" "ENSBTAG00000006240"
[7] "ENSBTAG00000015271"
```

3.7 Expression profile of a gene by group

3.7.1 Using the unique feature identifier

In the above section, we listed the genes associated with a particular gene ontology. In our example, the respective score and rank of each gene estimates the capacity of the gene to cluster the samples according to the treatment factor. The genes that best cluster the samples will have the highest scores and the lowest ranks. Those genes will likely produce the expression profiles with the most consistent differential expression between the treatment groups over time. Here is one example:

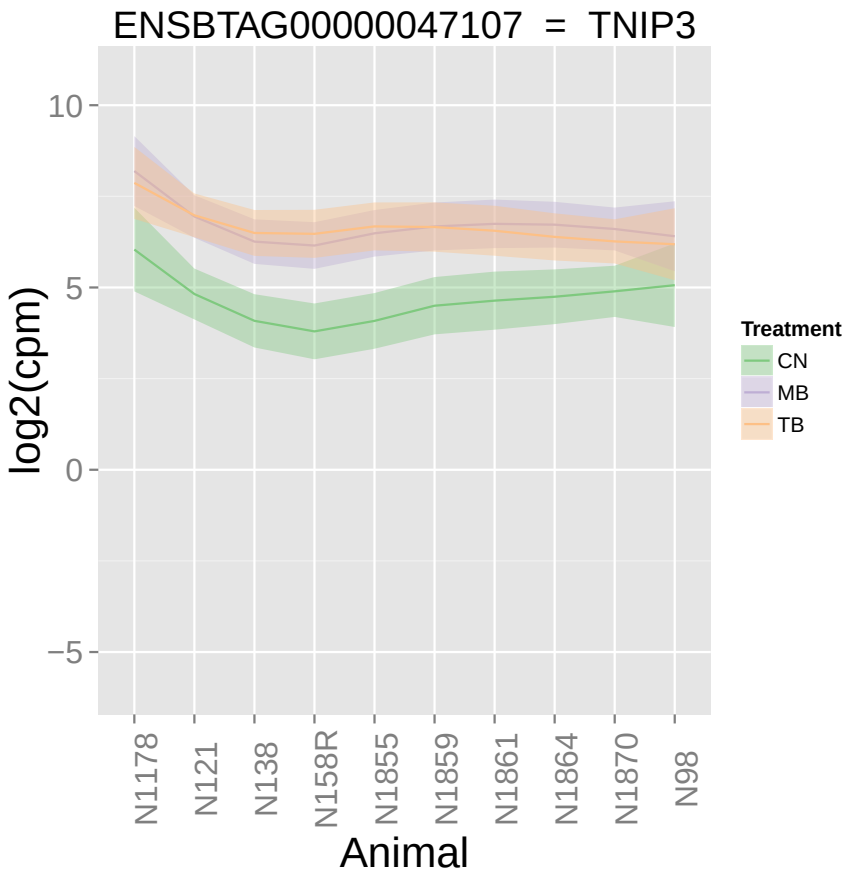
```
> expression_plot(gene_id = "ENSBTAG00000047107", result = BP.5, eSet=AlvMac,
+     x_var = "Timepoint", title.size=1.5,
+     legend.title.size=10, legend.text.size=10, legend.key.size=15)
```



Note that "`Timepoint`" is another column of `pData(targets)`. That column does not encode a factor, but a numeric vector. This difference enables the plotting function to respect the relative distance between the time-points for an output more representative of the actual time-scale.

To investigate the impact of other factors on the expression level of the same gene, users are encouraged to use the `f` and `x_var` arguments to specify alternate grouping factor and X variable, respectively. Note that the `geom_smooth` of the `ggplot2` package may fail if a minimal number of replicates is not available to calculate proper confidence intervals. Here is another valid example separating samples by animal on the X axis.

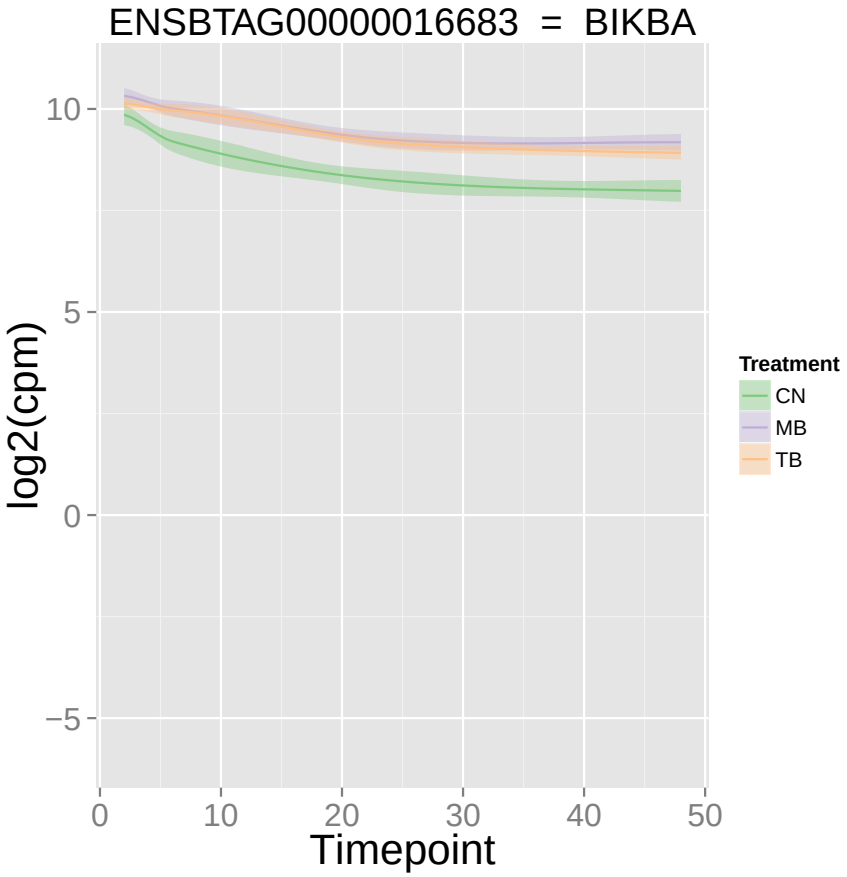
```
> expression_plot(gene_id = "ENSBTAG00000047107", result = BP.5, eSet=AlvMac,
+     x_var = "Animal", title.size=1.5, axis.text.angle=90,
+     legend.title.size=10, legend.text.size=10, legend.key.size=15)
```



3.7.2 Using the associated gene name

It is also possible to visualise the expression profile of genes from their associated gene names if any. This is a more human-friendly version of the function presented in the previous subsection:

```
> expression_plot_symbol(gene_symbol = "BIKBA", result = BP.5, eSet=AlvMac,
+     x_var = "Timepoint", title.size=1.5,
+     legend.title.size=10, legend.text.size=10, legend.key.size=15)

Fetching feature identifier(s) annotated to BIKBA ...
Unique gene id found for BIKBA
Plotting ENSBTAG00000016683
```



However, the benefits of this feature are balanced by the fact that genes lacking an associated gene name cannot be visualised in this manner, and that some gene symbols are associated with multiple Ensembl gene identifiers and probesets (e.g. 'RPL36A'). In the latter case, we turned the ambiguity into an additional useful feature: a lattice is created, and each of the multiple features associated with the given gene symbol are plotted simultaneously in the lattice. Subsequently, each of the sub-figures plotted may be re-plotted by itself using the `index` argument as indicated in the accompanying message printed in the R console.

3.8 Expression profile of a gene by individual sample series

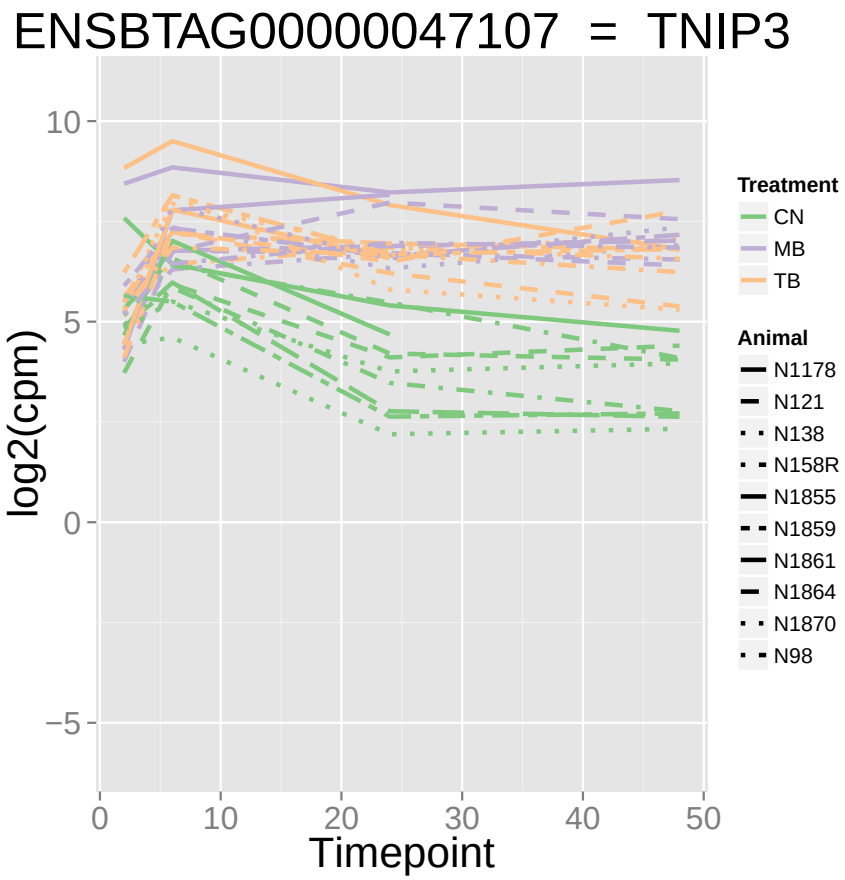
3.8.1 Using the unique feature identifier

It may be useful to track and visualise the expression profile of genes in each individual sample series, rather than their average. This could help identify outliers within sample groups, or visually compare paired samples, for instance.

In the `AlvMac` dataset, samples from each of the animals were subjected to all three treatments in parallel (i.e. paired samples). In the figure below, a

sample series is defined by a given `Animal` and a given `Treatment`. Each sample series is then tracked over time, and coloured according to the `Treatment` factor (default, factor stored in `raw_results$factor`):

```
> AlvMac$Animal.Treatment <- paste(AlvMac$Animal, AlvMac$Treatment, sep="_")
> expression_profiles(gene_id = "ENSBTAG00000047107", result = AlvMac_results,
+                     eSet=AlvMac, x_var = "Timepoint", line.size=1,
+                     seriesF="Animal.Treatment", linetypeF="Animal",
+                     legend.title.size=10, legend.text.size=10,
+                     legend.key.size=15)
```

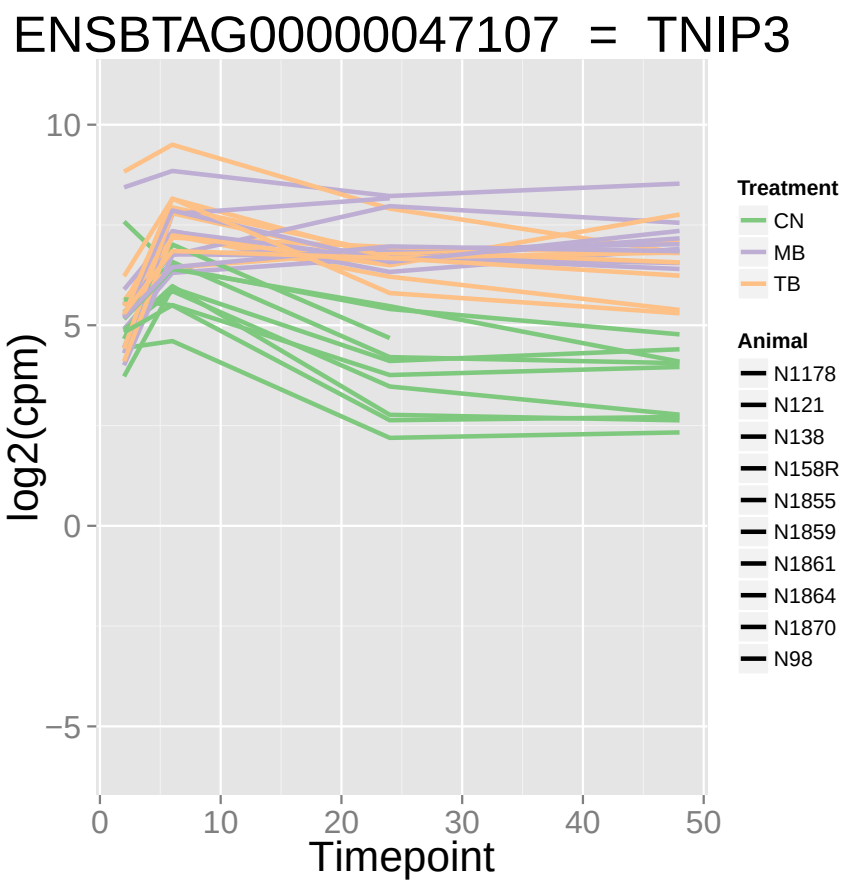


In the figure above, the `linetypeF` helps to highlight samples from an animal which start at unusually high expression values, while those samples progressively return to expression values similar to other samples in their respective treatment groups.

If omitted, the `linetypeF` argument will mirror the `colourF`, which can be useful for colour-blind people. Alternatively, a single line-type can be applied to all groups using the `argument` as follows:

```
> expression_profiles(gene_id = "ENSBTAG00000047107", result = AlvMac_results,
+                     eSet=AlvMac, x_var = "Timepoint", lty=rep(1,10),
```

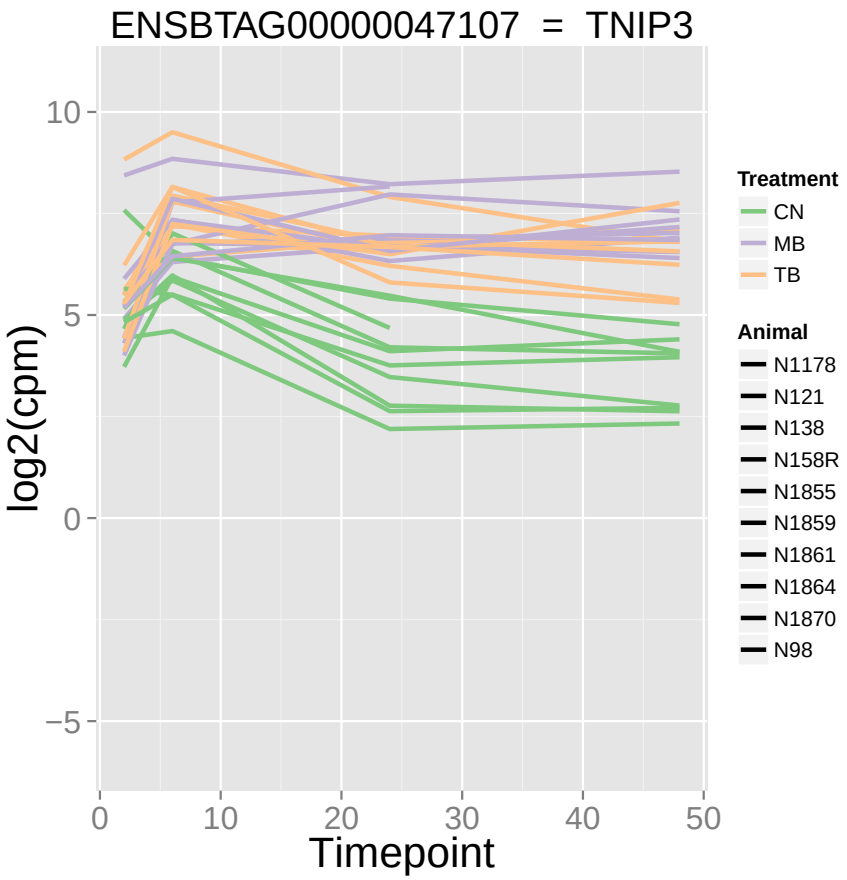
```
+                     # use line-type 1 for all 10 groups
+                     seriesF="Animal.Treatment", linetypeF="Animal",
+                     legend.title.size=10, legend.text.size=10,
+                     legend.key.size=15, line.size=1)
```



3.8.2 Using the associated gene name

Similarly to the `expression_plot` method, an alternative was implemented to use gene names instead of Ensembl gene identifiers. An example:

```
> expression_profiles_symbol(gene_symbol="TNIP3", result = AlvMac_results,
+                     x_var = "Timepoint", linetypeF="Animal", line.size=1,
+                     eSet=AlvMac, lty=rep(1,10), seriesF="Animal.Treatment",
+                     title.size=1.5, legend.title.size=10, legend.text.size=10,
+                     legend.key.size=15)

Fetching feature identifier(s) annotated to TNIP3 ...
Unique gene id found for TNIP3
Plotting ENSBTAG00000047107
```

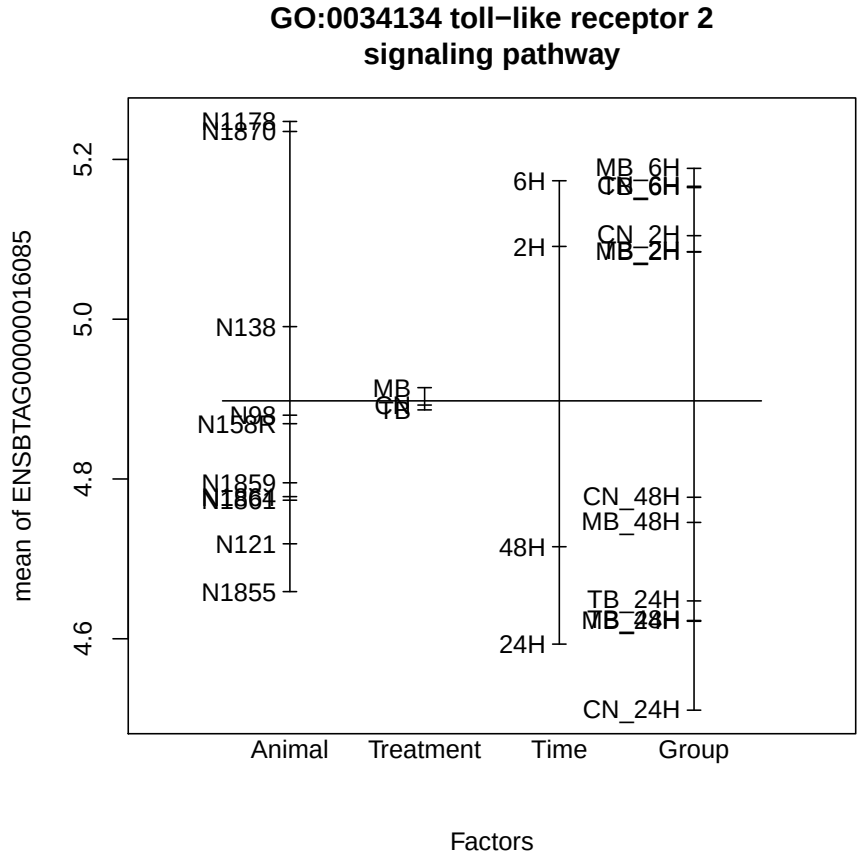


3.9 Comparison of univariate effects on gene expression

While the analysis is restricted to the evaluation of a single factor, it can be helpful to compare the relative impact of all known factors present in the the accompanying `phenoData` on the gene expression in the different groups of samples.

In other words, given a GO term identifier this feature will generate a plot for each associated gene, where the mean (default; can be changed) expression level will be computed for each level of each factor and compared to one another:

```
> plot_design(go_id = "GO:0034134", result = BP.5, eSet=AlvMac,
+     ask = FALSE, factors = c("Animal", "Treatment", "Time", "Group"),
+     main.Lsplit=30)
```



4 Additional controls and advanced functions

4.1 Subsetting an ExpressionSet to specific sample groups

It is straightforward to subset an `ExpressionSet` by extracting given columns (i.e. samples) and rows (i.e. gene features). Nevertheless, the `randomForest` package is quite sensitive to the definition of R factors; for instance, the `randomForest` package will crash if a factor is declared to have 3 levels (e.g. "A", "B", and "C"), while the `ExpressionSet` only contains samples for two of them (e.g. "A" and "B"). A simple fix is to update the known levels of the factor after having subsetted the `ExpressionSet`:

```
> AlvMac$Time <- factor(AlvMac$Time)
```

Note that this operation preservers the order of ordered factors.

However, subsetting an `ExpressionSet` by rows and columns does not automatically update the known levels of each factor to the remaining levels. For this task, the method `subEset` takes a named list, where item names must be column names from the `phenoData` slots and item values must be present in the corresponding columns, to return a subset of the original `ExpressionSet` and return only the samples which match those values:

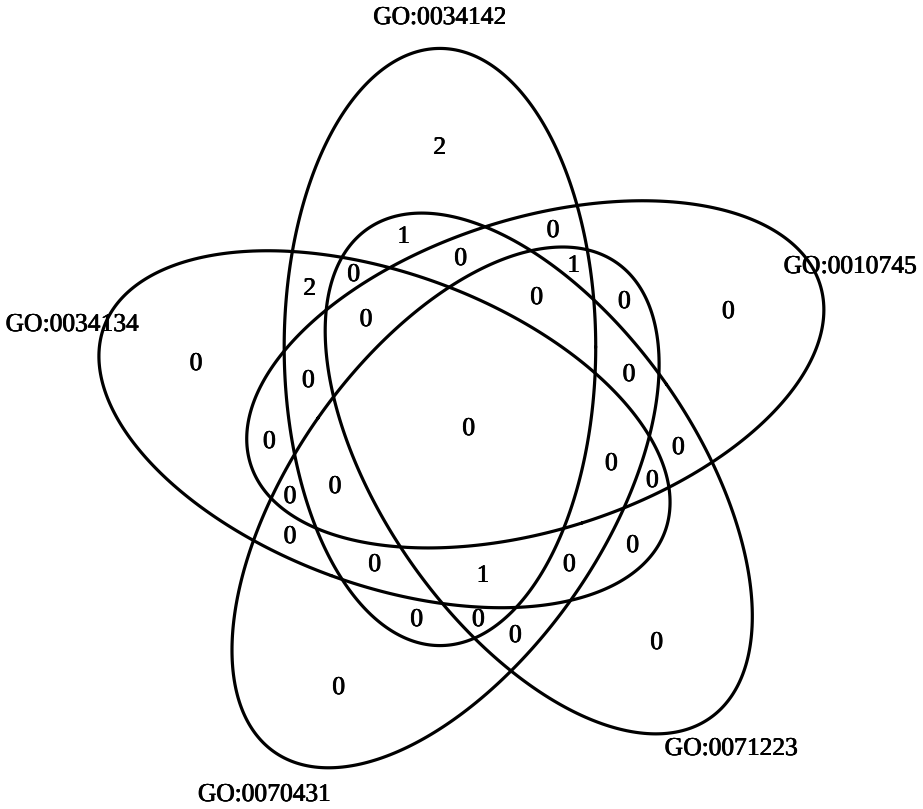
```
> subEset(eSet=AlvMac, subset=list(Time=c("2H","6H","24H"),
+                                  Treatment=c("CN","MB")))

ExpressionSet (storageMode: lockedEnvironment)
assayData: 100 features, 60 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: N1178_CN_24H N1178_CN_2H ... N98_MB_6H (60 total)
  varLabels: File Sample ... Animal.Treatment (8 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
```

4.2 Overlapping genes between GO terms

Often, the top-ranked cellular functions identified by pathway analysis and gene-set enrichment analysis tools may significantly overlap each other in terms of genes, which limits the number of genes identified by those approaches and blurrs the precise cellular function represented by the core gene-set among a list of closely related yet different functions.

Typically, overlapping sets are representated as Venn diagrams. The example below will produce a Venn diagram of the gene-sets associated with the five top-ranked "Biological Process" ontologies, and display it on screen, as the filename argument is left to the default `NULL` value (if a filename is given, the diagram will be printed to a TIFF file , no matter the extension specified):

```
> overlap_GO(go_ids = head(BP.5$GO$go_id, n=5), result = BP.5, filename=NULL)
```

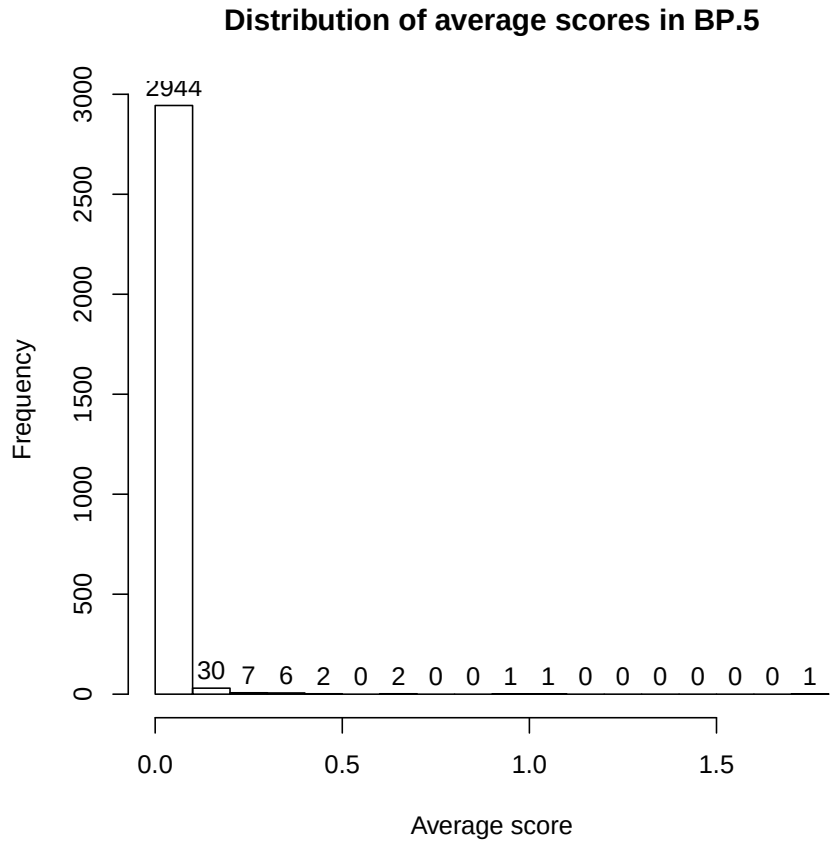


This method calls the `venn.diagram` method of the `VennDiagram` package, offering a high level of control on the resulting Venn diagram. A detailed description of the available arguments are accessible in the corresponding R documentation file.

4.3 Distribution of scores

Users might be interested in the general distribution of score and rank statistics produced by `GOexpress`. The distribution of scores may be represented as a histogram:

```
> hist_scores(result = BP.5, labels = TRUE)
```



Alternatively, quantile values can be returned for default or customised percentiles:

```
> quantiles_scores(result = BP.5)

       90%        95%        99%      99.9%     99.99%
0.00000000 0.03116737 0.15979435 0.68636057 1.54865250
```

4.4 Reordering by score

While scores are more prone to extreme outlier values and may slightly fluctuate between multiple runs of the random forest algorithm; ranks of genes and subsequently average ranks of GO terms are less prone to such variations and may be more reliable estimators of the importance of genes and cellular functions. Therefore, the default behaviour of `GOexpress` is to use the rank and average rank metrics to order genes and GO terms, respectively, in the returned score tables.

It is however possible to re-order the tables in the output variable according to the score metric (or revert back to the original one) as in the example below:

```
> BP.5.byScore <- rerank(result = BP.5, rank.by = "score")
```

5 Statistics

5.1 Overview

`GOexpress` was initially created from a set of gene-based, and later ontology-based, visualisation functions. Following the integration of those various plotting functions at the core of the `GOexpress` package, the need for a ranking of genes and GO terms soon became apparent in order to rapidly identify those that best cluster the samples according to the experimental factor studied, resulting in the clearest dendrograms and heatmaps. A two-fold procedure was implemented:

1. Using the available expression data, each gene present in the dataset is scored, evaluating its ability to cluster the predefined groups of samples. Genes are ranked according to their respective score; ties are resolved by assigning the lowest rank R to all G genes, giving the rank G+n to the next gene(s). The genewise scoring functions implemented are described in the following subsections, and all were validated on in-house datasets cross-checked with widely validated analytic pipelines (e.g. Ingenuity®Pathway Analysis, SIGORA)

2. Using the above ranks and scores, each GO term in the Ensembl BioMart is assigned the mean score and the mean rank of all the genes associated with it in the BioMart. Genes present in the BioMart but absent from the given dataset are assigned a score of 0 (minimum valid score; indicates no power to discriminates the predefined groups of sample) and a rank equal to the number of genes present in the entire dataset plus one (worst rank, while preserving discrete continuity of the ranking).

Importantly, the statistics performed to rank GO terms and genes do *not* influence the behaviour of the subsequent plotting functions; heatmaps, dendrograms and gene expression profiles are purely driven by the expression data, sample phenotype annotations provided and GO terms ontologies imported from the Ensembl BioMart server, without any transformation applied to the data. Therefore, users are encouraged to use and suggest alternative relevant scoring and ranking strategies, which could prioritise GO terms and genes in different ways. A current acknowledged bias is the higher scoring of GO terms associated with fewer genes, which shows some advantages as discussed in section 3.3.

5.2 Random Forest

We implemented the Random Forest framework to answer the question: "How well does each gene in the dataset cluster predefined groups of samples?". The random forest consists of multiple decision trees. Each tree is built based on a bootstrap sample (sample with replacement) of observations and a random sample of variables. The `randomForest` package first calculates the Gini index (Breiman et al, 1984) for each node in each tree. The Gini index is a measure of homogeneity from 0 (homogeneous) to 1 (heterogeneous). The decrease in the Gini index resulting from a split on a variable is then calculated for each node and averaged for each variable over all the trees in the model. The variable with the biggest mean decrease in the Gini index is then considered the most

important. Technically, GOexpress extracts the MeanDecreaseGini value from the importance slot of the randomForest output and uses this value as the score for each gene.

A key feature of the Random Forest framework is the implicit handling of interactions between genes, given a sufficient number of trees generated and genes sampled. Indeed, at each node in the decision tree, genes are sampled from the expression data and tested for their individual capacity to improve the partitioning reached in the previous node. The larger the number of trees and genes sampled, the more complete the coverage of interactions will be.

5.3 One-way Analysis of Variance (ANOVA)

We implemented the ANOVA statistical framework to answer the question: "How different is the expression level of each gene in the dataset in the different groups of samples?". Given the expression level of a gene in all the samples, the one-way ANOVA determines the ratio of the variance between the groups compared to the variance within the groups, summarised as an F statistic that GOexpress uses as a score for each gene. Simply put, if samples from the same groups show gene expression values similar to each other while samples from different groups show different levels of expression, those genes will produce a higher score. This score cannot be less than 0 (variance between groups insignificant to variance within groups), while very large ratios can easily be reached for genes markedly different between groups.

Contrary to the random forest framework, the Analysis of Variance makes important assumptions on the data: namely, the independence of observations, the normality of residuals, and the equality of variances in all groups. While the former two are the responsibility of the user to verify, the latter is taken care of by GOexpress. Indeed, the oneway.test method of the package stats is used with parameter var.equal set to to FALSE. While this reduces the sensitivity of the test, all genes are affected by this correction based on the relative amount of variance in the different predefined groups of samples. Finally, it is once again important to note that the ANOVA only evaluates univariate changes, while the random forest framework implicitely allows for interactions between genes.

6 Notes

6.1 Authors' contributions

Conception and development of the GOexpress package was carried out by KR-A with contributions by PAM, under the supervision of SVG and DEM. Experimental data used for testing was generated and analysed with the help of DAM and NC. Integration of the random forest statistical frameworks was advised by BH and AP. This User's Guide was prepared by KR-A, and edited by PAM, BH, DAM, NCN, AP, SVG, and DEM.

6.2 Acknowledgments

Special thanks to Dr. Kate Killick for testing and feedback. My thanks to the Bioconductor staff and in particular to Hervé Pagès for the helpful feedback which improved the standards of the code and documentation. Last but not

least, thanks to the University College Dublin "OpenSequencing" and the "Virtual Institute of Bioinformatics and Evolution" (VIBE) communities where I first presented raw versions of `GOexpress` and got valuable feedback and advice in return, underlying a significant number of updated and new features.