GOsummaries basics

Raivo Kolde raivo.kolde@eesti.ee

April 3, 2016

Contents

1 Introduction

GOsummaries is a package to visualise Gene Ontology (GO) enrichment analysis results on gene lists arising from different analyses such clustering or PCA. The significant GO categories are visualised as word clouds that can be combined with different plots summarising the underlying data.

2 Elements of a GOsummaries plot

Figure **??** shows an example of plot generated with *GOsummaries* package. All the *GOsummaries* plots have more or less the same layout, however, the elements are adjusted corresponding to the underlying analysis. The plot is built of components (Fig **??**A) that represent a gene list or a pair of gene lists as in Fig **??**. Each component is composed of two parts the word cloud(s) (Fig **??**E), representing the GO annotations of the gene lists, and a panel (Fig **??**C) that displays the underlying data experimental data. In this case the panel shows the expression values of the corresponding genes. There are also slots for the component title (Fig **??**B) and some additional information about the gene lists (Fig **??**D).

In the word clouds the sizes of the GO categories indicate the strength of enrichment, relative to the other results of the same query. To make global comparison of the strength of enrichment possible we use different shades of grey.

3 Usage of GOsummaries

In most cases the *GOsummaries* figures can be created using only two commands: `gosummaries` to create the object that has all the necessary information for drawing the plot and `plot.gosummaries` to actually draw the plot.

The `gosummaries` function requires a set of gene lists as an input. It applies GO enrichment analysis to these gene lists using g:Profiler (http://biit.cs.ut.ee/gprofiler/) web toolkit and saves the results into a `gosummaries` object. Then one can add experimental data and configure the slots for additional information.

However, this can be somewhat complicated. Therefore, we have provided several convenience functions to that generate the `gosummaries` objects based on the output of the most common analyses. We have functions `gosummaries.kmeans`,`gosummaries.prcomp` and `gosummaries.MArrayLM`, for k-means clustering, principal component analysis (PCA) and linear models with *limma*. These functions extract the gene lists right from the corresponding objects, run the GO enrichment and optionally add the experimental data in the right format.

The `gosummaries` can be plotted using the `plot` function. The figures might not fit into the plotting window, since the plot has to have rather strict layout to be readable. Therefore, it is advisable to write it into a file (file name can be given as a parameter).

Creating a simplest *GOsummaries* plot, starting from the gene lists goes as follows (example taken from `?GOsummaries`):

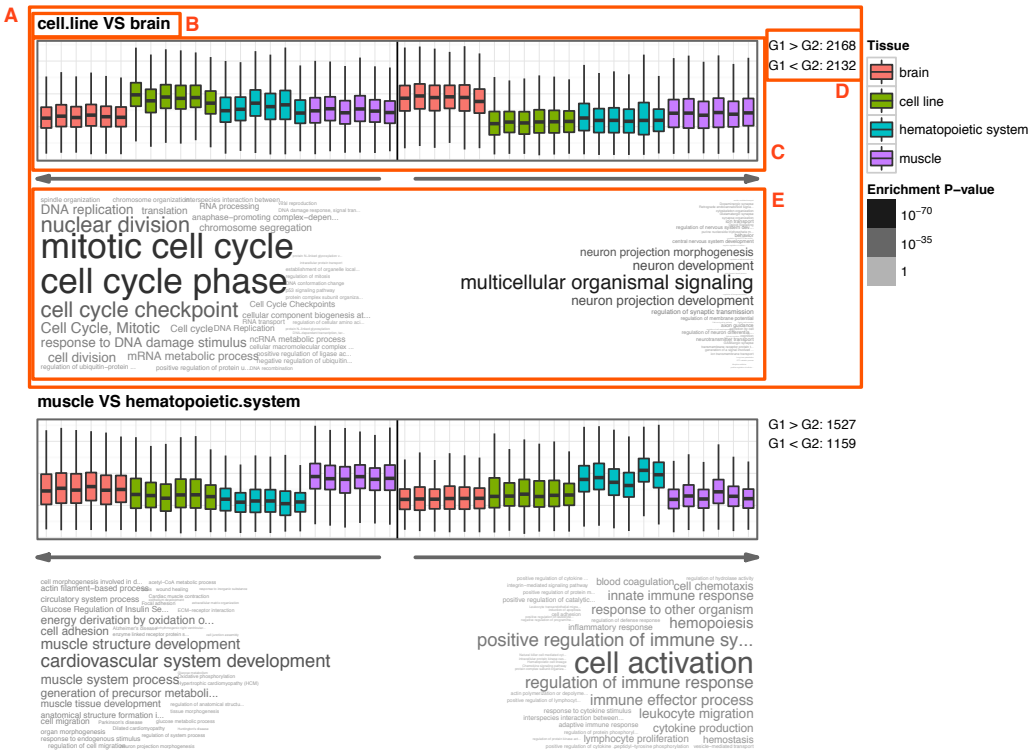

Figure 1: Elements of a GO summaries figure

```
> # Define gene lists
> genes1 = c("203485_at", "209469_at", "209470_s_at", "203999_at",
+          "205358_at", "203130_s_at", "210222_s_at", "202508_s_at", "203001_s_at",
+          "207957_s_at", "203540_at", "203000_at", "219619_at","221805_at",
+           "214046_at", "213135_at", "203889_at", "209990_s_at", "210016_at",
+          "202507_s_at","209839_at", "204953_at", "209167_at", "209685_s_at",
+          "211276_at", "202391_at", "205591_at","201313_at")
> genes2 = c("201890_at", "202503_s_at", "204170_s_at", "201291_s_at",
+          "202589_at", "218499_at", "209773_s_at", "204026_s_at", "216237_s_at",
+          "202546_at", "218883_s_at", "204285_s_at", "208659_at", "201292_at",
+          "201664_at")
> gl = list(List = list(genes1, genes2)) # Two lists per component
> # Construct gosummaries objects
> gs = gosummaries(gl)
> plot(gs, fontsize = 8, filename = "figure2.pdf")
```

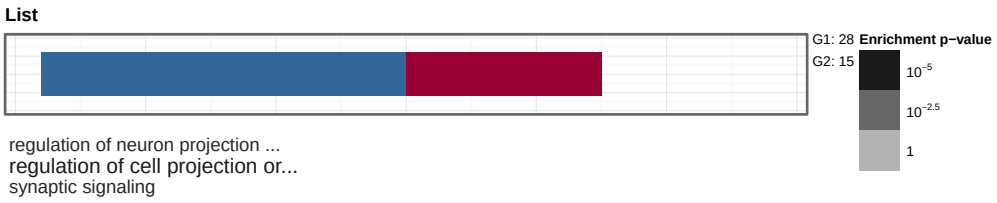

Figure 2: Simplest GO summaries figure.

In this example we had only the gene lists and no additional data to display in panel. In these situations *GOsummaries* displays by default just the number of genes.

These gene lists can be also displayed as separate components if the input gene list would have been constructed a bit differently.

```
> gl = list(List1 = genes1, List2 = genes2)
```

3.1 Configuring the GO analysis

Main task for the `gosummaries` function is to perform the GO enrichment analysis. To be able to fit the GO enrichment results into the word cloud, we have to reduce their number quite a bit. We have defined some default parameters for this. Still, there might be a need to adjust those parameters. These parameters apply to all versions of the `gosummaries` function.

In the first step we throw out results from less interesting GO branches. For example, by default we throw out results from Molecular Function and Cellular Component branch, since the results are often not as interesting. But this behaviour can be changed using the `go_branches` parameter.

Then we throw out categories that are either too big or too small, since very small categories might not describe the gene list as a whole and very large categories on contrary can be too generic. The exact values for these parameters can be controlled by parameters `min_set_size` and `max_set_size`

Finally we have set an upper limit for the number of categories to display, this can be changed using `max_signif` parameter. Of course, one can change also the p-value threshold with `max_p_value`.

It is also important to note that we assume, that the gene lists are ordered. If they are not then the option `ordered_query` should be set to FALSE.

3.2 Adding expression data

In case of clustering and differential expression there is an option to display expression data alongside the word clouds (see Figure **??**). In there, each boxplot represents the distribution of expression values of the genes in the current list in one particular sample. If samples correspond to different classes, tissues or treatments then it can be shown with different colours.

In `gosummaries.kmeans` and `gosummaries.MArrayLM` we have special parameters to add the expression data and its annotations: `exp` and `annotation`. The `exp` variable takes in an expression matrix, where rows correspond to genes and columns to samples. The correct expression values are extracted, based on the row names. Therefore, gene names in the gene list have to be present in the expression matrix. The `annotation` parameter accepts a `data.frame` where each row describes one sample. Therefore, the column names of `exp` have to be present in the row names of `annotation`.

Here is an example of adding the expression data:

```
> data(tissue_example)
> # Filter genes and perform k-means
> sd = apply(tissue_example$exp, 1, sd)
> exp2 = tissue_example$exp[sd > 0.75,]
> exp2 = exp2 - apply(exp2, 1, mean)
> kmr = kmeans(exp2, centers = 6, iter.max = 100)
> # Create gosummaries object
> exp2[1:6, 1:5]
```

	GSM123197.CEL	356362160.CEL	GSM123234.CEL	356360072.CEL	356362624.CEL
1294_at	-0.4670833	-0.7170833	-0.9470833	-0.5270833	-0.6870833
1405_i_at	-0.8975000	-1.1775000	-0.5875000	-0.9275000	-1.4975000
200002_at	-0.3045833	-0.8645833	-0.6345833	-0.7145833	-0.9745833
200003_s_at	-0.5841667	-1.0541667	-0.8141667	-0.8741667	-0.8241667
200005_at	-1.0491667	-0.3991667	-0.7791667	-1.0291667	-0.6591667
200008_s_at	-0.5475000	-0.5275000	-0.8775000	-0.1675000	-0.5675000

```
> head(tissue_example$annot)
```

```
              Tissue                                Meta15   Meta4
GSM123197.CEL  brain solid tissue non neoplastic disease disease
356362160.CEL  brain solid tissue non neoplastic disease disease
GSM123234.CEL  brain                  normal solid tissue  normal
356360072.CEL  brain solid tissue non neoplastic disease disease
356362624.CEL  brain                  normal solid tissue  normal
356367950.CEL  brain                  normal solid tissue  normal
```

```
> gs_kmeans = gosummaries(kmr, components = 1:2, exp = exp2, annotation = tissue_example$annot)
> plot(gs_kmeans, fontsize = 8, classes = "Tissue", filename = "figure3.pdf")
```

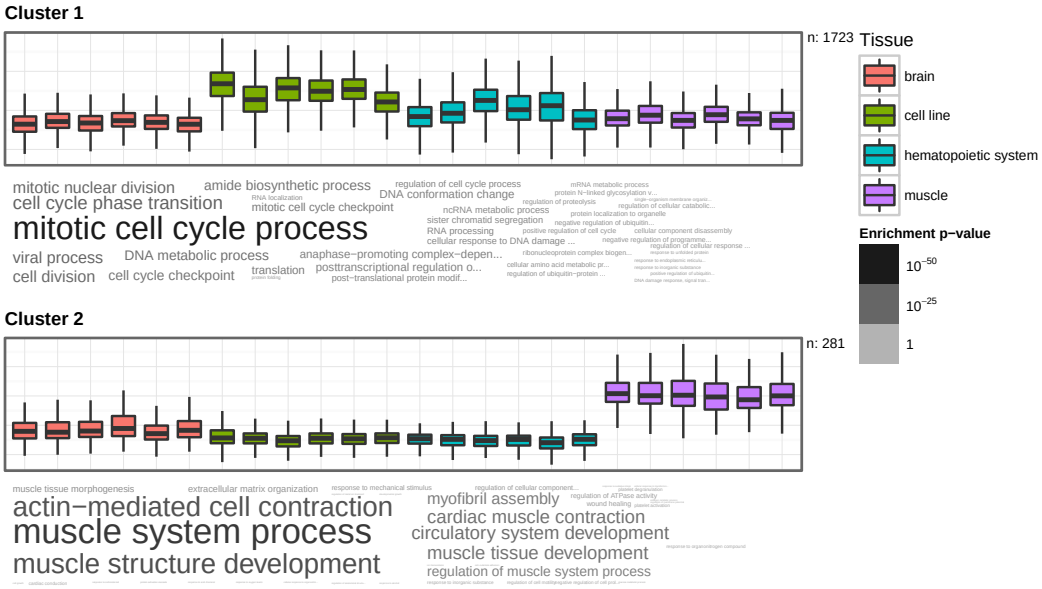

Figure 3: K-means plot with added expression data.

If one wants to add expression data to a custom `gosummaries` object then it is possible to use a function `add_expression.gosummar` that adds the expression data to an existing `gosummaries` object. The other parameters `exp` and `annotation` work as described above. For example, if we want to add expression data to the `gosummaries` object from the first example, we can write.

```
> data(tissue_example)
> gs_exp = add_expression.gosummaries(gs, exp = tissue_example$exp,
annotation = tissue_example$annot)
Using  as id variables
Using  as id variables
```

3.3 Configuring the plot appearance

The layout of the plot is fixed. However, it is still possible to configure some parameters. For example, the proportions of the panel area with parameters `panel_height` and `panel_width`. The unit for these measures is lines of text. Using these units keeps the proportions of the plot similar even if we change the `fontsize`. The panel height parameter is most useful if one wants to omit the panel area completely. Then one can set the `panel_height` to 0.

The content of the panels is drawn by the function that is specified in the `panel_plot` parameter. If one uses the built-in functions, such as `gosummaries.prcomp`, `gosummaries.kmeans`, etc. then the most suitable panel drawing function is selected automatically. Without any expression data, only the number of genes is displayed in there. In case of PCA, we display projection of the values to the principal component as histogram. For clustering and differential expression we show the boxplots of the expression in different samples. Instead of boxplot, one can use also the violin plot (`panel_violin`) or combination of boxplot and violin plot (`panel_violin_box`).

All the panel drawing functions basically generate a *ggplot2* plot based on the `Data` slot in a component of `gosummaires` object. From there we extract the plot area to display in panel and also the legend. In principle it is possible to define your own functions, as long as its input and output are match our functions and it conforms to the data in the `Data` slot in the components of `gosummaires` object. See the help of `panel_boxplot` and the source of these functions for more information.

If one wants to make smaller changes to the panels, such as, change the colour scheme, then for this we have easier means than defining new panel function. With the parameter `panel_customize` one can specify a function that modifies the plot created with the `panel_plot` function. For example the default function `customize` looks like this.

```
function(p, par){
p = p + ggplot2::scale_fill_discrete(par$classes)
return(p)
}
```

To select a different colour scheme one can modify that function and and give it to the `plot.gosummaires` function.

```
> cust = function(p, par){
+         p = p + scale_fill_brewer(par$classes, type = "qual", palette = 2)
+         return(p)
+ }
> plot(gs_kmeans, panel_plot = panel_violin, panel_customize = cust,
+ classes = "Tissue", components = 1:2, filename = "ex3.pdf")
```

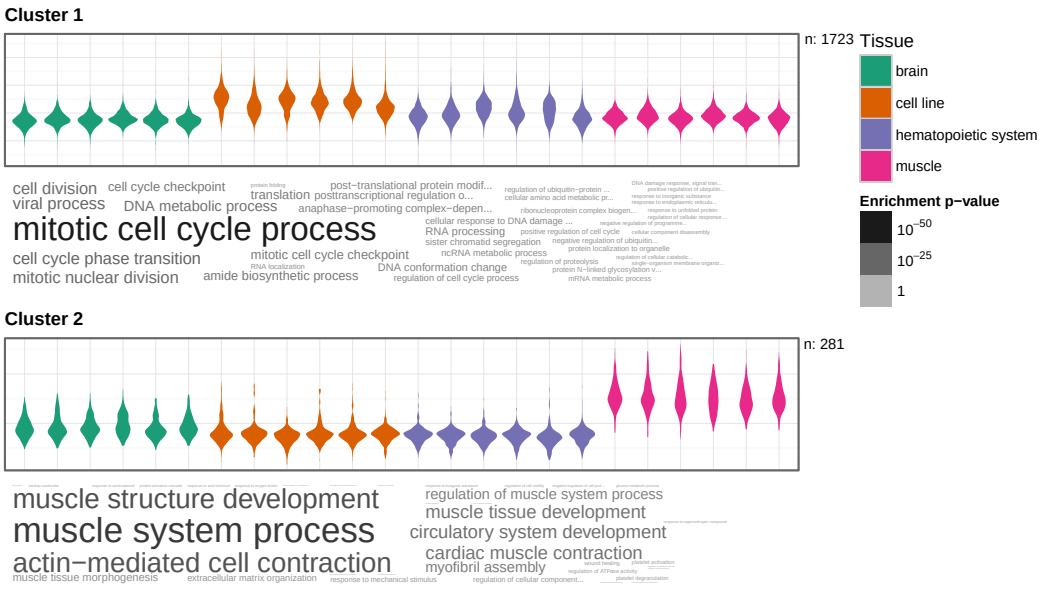

Figure 4: K-means plot with modified color scheme and violin plots instead of boxplots.

4 Using other annotation sources for word clouds

Right now the default pipeline always runs a g:Profiler query on the given genes and displays the results as word clouds. However, there are several situations where it would be reasonable to use data from some other source in the word clouds. For example, other GO enrichment tools might give more reasonable results.

For such cases there is a parameter `wc_data` in `gosummaries.default` where one can enter arbitrary data that will be shown on word clouds. The input structure is similar to the gene list input , only instead of vectors with gene names it requires data frames with two columns: "Term" and "Score". Where Term is the text that is being drawn and Score determines its size.

```
> wcd1 = data.frame(Term = c("KLF1", "KLF2", "POU5F1"), Score = c(0.05, 0.001, 0.0001))
> wcd2 = data.frame(Term = c("CD8", "CD248", "CCL5"), Score = c(0.02, 0.005, 0.00001))
```

To get one word cloud per block use flat list.

```
> gs = gosummaries(wc_data = list(Results1 = wcd1, Results2 = wcd2))
> plot(gs, filename = "figure5.pdf")
```

Figure 5: User supplied wordcloud data as two components

To get two word clouds per block use neted lists.

```
> # To get two word clouds per block use neted lists
> gs = gosummaries(wc_data = list(Results = list(wcd1, wcd2)))
> plot(gs, filename = "figure6.pdf")
```

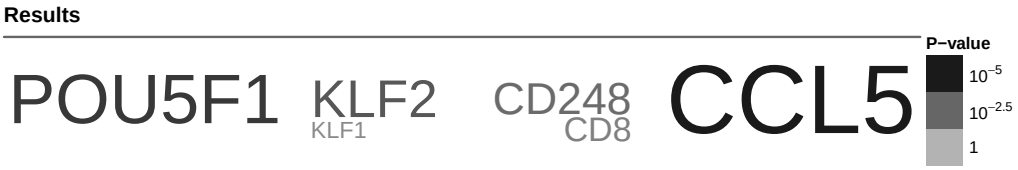

Figure 6: User supplied wordcloud data as one components

One can also add the gene lists when specifying `wc_data`, but they can be in many cases omitted. This option makes it easy to incorporate the GO enrichment results from other tools.

4.1 Attributes

Several general properties of the plot are stored in the attributes of `gosummaries` object. If needed, these can be changed using the `attr` function.

- `score_type`: Specifies how to handle the scores associated to words in word clouds. In case of "count", the score is expected to be positive and the word sizes are directly proportional to the scores. In case of "p-value", the word sizes are proportional to $-\log_{10}$ of the score.
- `wc_algorithm` Specifies the word cloud layout algorithm. In case of "top", the word placement starts from the top corner, in case of "middle" from the centre of left or right side of the box.
- `wordcloud_legend_title` Gives the title of the word cloud.

4.2 Displaying gene names instead of GO categories

If the gene lists or the whole dataset is very small then the GO analysis might not give many significant results. In these cases it would be more reasonable to show the names of genes instead. It is possible to add the gene lists as described above with `wc_data`. However, more convenient means are implemented for PCA, limma and MDS results in `gosummaries.prcomp`, `gosummaries.MArrayLM` and `gosummaries.matrix` respectively.

In each case there is a parameter `show_genes` that toggles if gene names or GO categories are shown. The parameters to decide the importance for different genes vary between these functions. For PCA we use the size of the component

loadings, for limma we use the adjusted p-value and for MDS we use the p-values of Spearman rank correlation with the components.

As gene identifiers in expression matrices can be unintelligible then by default these functions convert the identifiers into gene names using `gconvert` function from *gProfileR* package. It is possible to turn this function off as well by setting parameter `gconvert_target` to `NULL`.

These options are especially important when use *GOsummaries* on data that is not describing genes. For example, PCA analysis is often used for other high throughput experiments, such as metabolomics and metagenomics. Using the *GOsummaries* approach on these datasets can be very revealing.

4.2.1 Example: metagenomics

Principal Coordinate Analysis is very common on metagenomics data. To conveniently visualise these results with *GOsummaries*, there is a function `gosummaries.matrix`. Since the rows represent taxa instead of genes we cannot use GO enrichment analysis, but we can show the names of taxons.

```
> data(metagenomic_example)
> # Run Principal Coordinate Analysis on Bray-Curtis dissimilarity matrix
> pcoa = cmdscale(vegdist(t(metagenomic_example$otu), "bray"), k = 3)
> # By turning off the GO analysis we can show the names of taxa
> gs = gosummaries(pcoa, metagenomic_example$otu, metagenomic_example$annot,
+                  show_genes = T, gconvert_target = NULL, n_genes = 30)
> plot(gs, class = "BodySite", fontsize = 8, file = "figure7.pdf")
```

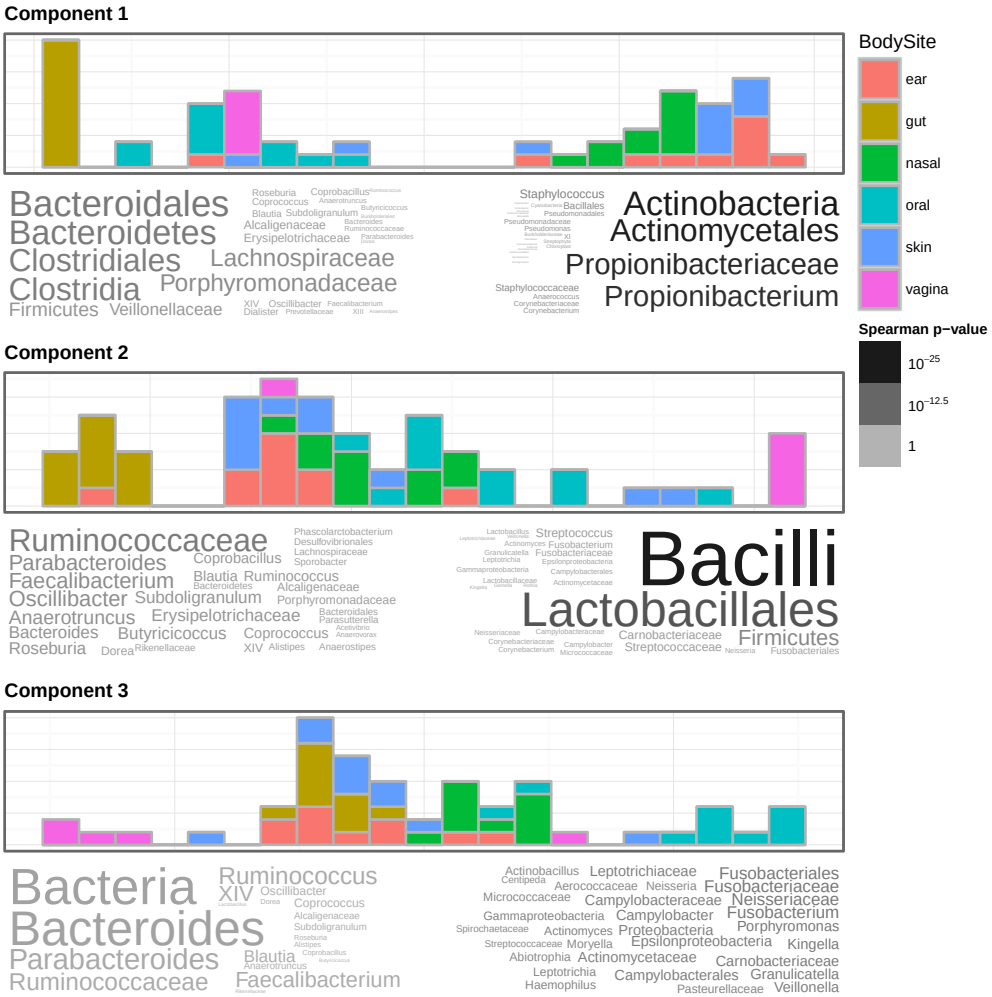

Figure 7: Visualisation of PCoA results on metagenomic data

5 Session info

```
> sessionInfo()
```

```
R version 3.2.4 (2016-03-10)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] ggplot2_2.1.0     vegan_2.3-4       lattice_0.20-33   permute_0.9-0
[5] GOsummaries_2.4.7 Rcpp_0.12.4

loaded via a namespace (and not attached):
 [1] cluster_2.0.3       magrittr_1.5        MASS_7.3-45         munsell_0.4.3
 [5] colorspace_1.2-6    gProfileR_0.5.3     stringr_1.0.0       plyr_1.8.3
 [9] tools_3.2.4         parallel_3.2.4      grid_3.2.4          gtable_0.2.0
```

```
[13] nlme_3.1-126       mgcv_1.8-12        digest_0.6.9       Matrix_1.2-4
[17] RColorBrewer_1.1-2 reshape2_1.4.1     bitops_1.0-6       RCurl_1.95-4.8
[21] labeling_0.3       limma_3.26.9       stringi_1.0-1      scales_0.4.0
[25] BiocStyle_1.8.0
```