GOsummaries basics

Raivo Kolde `rkolde@gmail.com`

April 4, 2013

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 3.2 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 3.2A) that represent a gene list or a pair of gene lists as in Fig 3.2. Each component is composed of two parts the word cloud(s) (Fig 3.2E), representing the GO annotations of the gene lists, and a panel (Fig 3.2C) 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 3.2B) and some additional information about the gene lists (Fig 3.2D).

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.

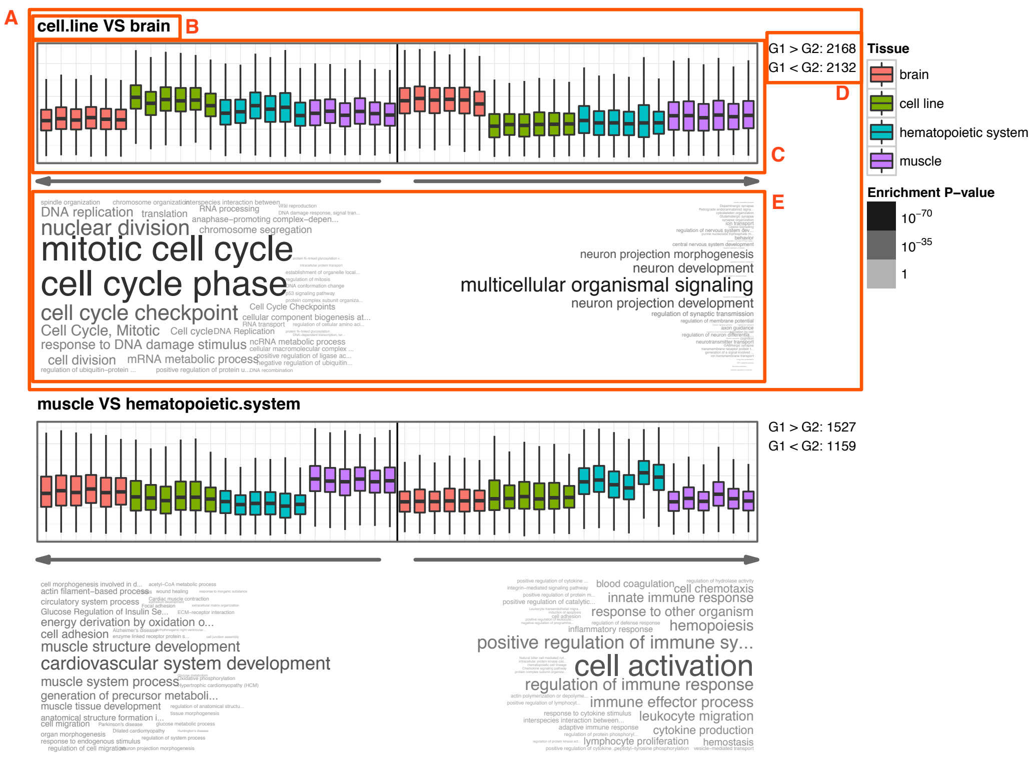

Figure 1: Elements of a GO summaries figure

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`):

```
> # 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)
```

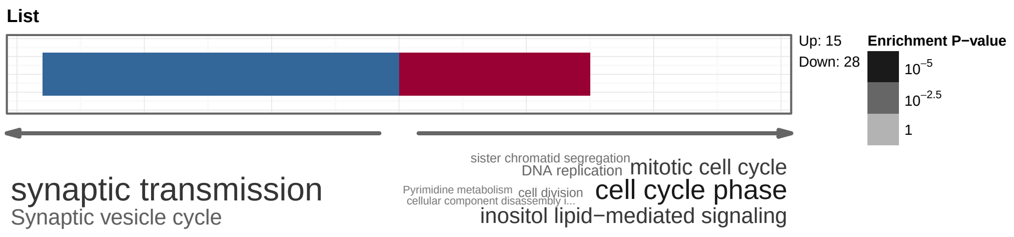

Figure 2: Simplest GO summaries figure.

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

```
200003_s_at    -0.8241667
200005_at      -0.6591667
200008_s_at    -0.5675000
> head(tissue_example$annot)
              Tissue                              Meta15   Meta4
GSM123197.CEL  brain solid tissue non neoplastic disease disease
356362160.CEL  brain solid tissue non neoplastic disease disease
GSM123234.CEL  brain                 normal solid tissue  normal
356360072.CEL  brain solid tissue non neoplastic disease disease
356362624.CEL  brain                 normal solid tissue  normal
356367950.CEL  brain                 normal solid tissue  normal
>
> gs_kmeans = gosummaries(kmr, exp = exp2, annotation = tissue_example$annot)
```

If one wants to add expression data to a custom `gosummaries` object then it is possible to use a function `add_expression.gosummaries` 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
> plot(gs_exp, classes = "Tissue")
```

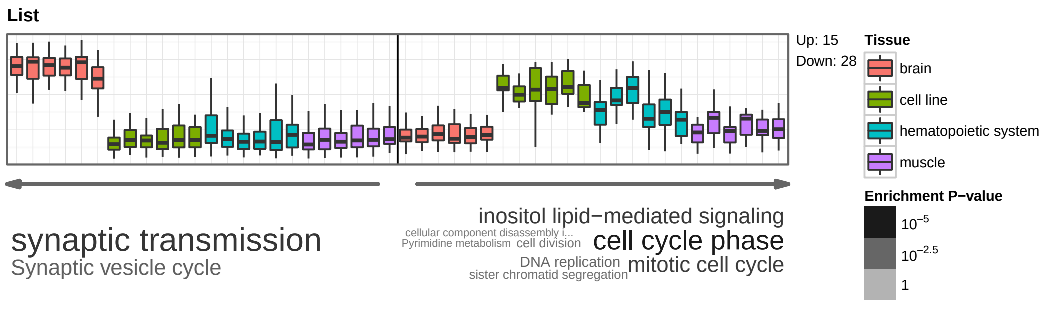

Figure 3: `GOsummaries` figure with added expression data.

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)
```