Babraham Bioinformatics

Exercises:
Gene List Analysis in R

Version 2025-7

Licence

.

Introduction

In these exercises we will replicate some of the analyses we previously ran on various web sites, but this time we'll run it in R using the analysis implementations in the ClusterProfiler package from Bioconductor.

We'll be using different gene lists this time – they come from this paper:

"Nucleosomal asymmetry shapes histone mark binding and promotes poising at bivalent domains"

We'll try both a categorical and a quantitative (GSEA) enrichment analysis and look at various options for visualising and exporting the results.

The analysis will be run on a cloud based RStudio Server instance into which we have installed the various packages which we'll be using for the exercises. Details of the packages we installed can be seen on our github repository containing all of our training image build instructions.

Exercise 1: Categorical Enrichment Analysis

Depending on your degree of familiarity with R you can either do these exercises in a plain R script (`File > New File > RScript`), an R Notebook (`File > New File > R Notebook`) or a Quarto Document (`File > New File > Quarto Document`). If you're fairly new to R then a plain RScript text file will be the simplest way to do this.

Loading required packages

Our first code will simply be loading in the packages we're going to use. These will be the packages containing the analysis and plotting code for the gene set analysis, but also the generic tidyverse packages which are useful for general data loading and manipulation. We'll also load in the mouse genome since the data we're using this time comes from mouse.

```
library(clusterProfiler)
library(enrichplot)
library(org.Mm.eg.db)
library(tidyverse)
```

Loading categorical enrichment data

The categorical list of hits is going to come from the file Programatic_Analysis/KAT6AB_knockout.txt in the course data folder.

If you are using a Notebook or Quarto document then just save your document into that folder to set the working directory. If you're using an R script then select Session > Set Working Directory > Choose Directory and then use the file browser to select this directory (not file).

Once you have done this you can load the data into a variable in R.

```
read_delim("KAT6AB_knockout.txt") -> kat6ab
```

You can look at the data by clicking on the kat6ab variable in the Environment window (top right)

Gene	Chromosome	Start	End	Strand	ID	Description	Pvalue	FDR	Log2FC	ShrunkLFC
Xkr4	1	3276124	3741721	-	ENSMUSG00000051951	X-linked Kx blood group related 4 [Source:MGI Symbol;Acc:...	1.948578e-03	1.569322e-02	-1.1502171755	-9.793486e-01
Rp1	1	4069780	4479464	-	ENSMUSG00000025900	retinitis pigmentosa 1 (human) [Source:MGI Symbol;Acc:MG...	4.288777e-01	6.808761e-01	-0.7388939857	-3.504721e-01
Sox17	1	4561154	4567577	-	ENSMUSG00000025902	SRY (sex determining region Y)-box 17 [Source:MGI Symbol;...	8.341584e-05	1.352577e-03	0.7363540530	7.051282e-01
Mrpl15	1	4843429	4855962	-	ENSMUSG00000033845	mitochondrial ribosomal protein L15 [Source:MGI Symbol;A...	3.937597e-01	6.500896e-01	-0.1394474804	-1.348887e-01
Lypla1	1	4878011	4918633	+	ENSMUSG00000025903	lysophospholipase 1 [Source:MGI Symbol;Acc:MGI:1344588]	6.196536e-01	8.009368e-01	-0.1147968173	-1.075258e-01
Tcea1	1	4928037	4968132	+	ENSMUSG00000033813	transcription elongation factor A (SII) 1 [Source:MGI Symbol;...	3.819878e-01	6.391129e-01	-0.1599865854	-1.534898e-01
Rgs20	1	4979799	5140508	-	ENSMUSG00000002459	regulator of G-protein signaling 20 [Source:MGI Symbol;Acc...	3.167324e-10	3.638275e-08	-1.7807973623	-1.616850e+00
Atp6v1h	1	5140241	5232752	+	ENSMUSG00000033793	ATPase, H+ transporting, lysosomal V1 subunit H [Source:M...	3.817989e-01	6.389396e-01	-0.1414234489	-1.368975e-01
Oprk1	1	5658689	5676354	+	ENSMUSG00000025905	opioid receptor, kappa 1 [Source:MGI Symbol;Acc:MGI:97439]	1.327109e-02	6.581613e-02	-1.4095416069	-1.000554e+00
Npbwr1	1	5983926	5987617	-	ENSMUSG00000033774	neuropeptides B/W receptor 1 [Source:MGI Symbol;Acc:MGI...	8.279712e-01	9.200836e-01	-0.1616381109	-9.525006e-02

Making Gene Lists

We need to make two lists of genes

1. The full list of measured genes in the data (background list)
2. The list of genes which are upregulated (adjusted p < 0.05 and log2FC > 1)

These will be the input to our gene set analysis.

```
kat6ab |>
  pull(Gene) -> background_genes

kat6ab |>
  filter(FDR < 0.05) |>
  filter(Log2FC > 1) |>
  pull(Gene) -> up_genes
```

Running the Gene Set Analysis

For this analysis we're going to use the enrichGO function from clusterProfiler. Here, we'll just use the BP (Biological Process) subset of the Gene Ontology, but you can try the MF (Molecular Function) or CC (Cellular Component) sections too if you'd like.

We're using mouse genes, and we're using gene symbols as input.

You can run

```
keytypes(org.Mm.eg.db)
```

..to see the list of available keys. In our case we're going to use SYMBOL as the key type since we have gene symbols for input.

We're restricting the gene set size to between 10 and 100 genes so we exclude the large non-specific gene sets at the top of the ontology, and the small sets with very few genes. These numbers aren't set in stone so you can always try playing with the cutoffs to see what effect it has.

```
enrichGO(
  up_genes,
  OrgDb = org.Mm.eg.db,
  keyType = "SYMBOL",
  universe = background_genes,
  ont = "BP",
  minGSSize = 10,
  maxGSSize = 100,
  readable = TRUE
) -> enrichgo_results
```

Viewing tabular results

The initial output is a data structure specifically tied to enrichment results, however it can trivially be converted to a standard tibble which will make it easier to look at. You can run:

```
enrichgo_results |> as_tibble()
```

..to see the results in the document / console, or you can run:

```
enrichgo_results |> as_tibble() |> view()
```

..to open them in the graphical viewer in RStudio

If you wanted to save these results to a tab delimited text file so you can create a table of results then you can do this with.

```
enrichgo_results |>
  as_tibble() |>
  write_tsv("categorical_enrichment_results.tsv")
```

We can also see how many interesting hits we have – these would have an adjusted p-value below 0.05 and an enrichment of at least 2X

```
enrichgo_results |>
  filter(p.adjust<0.05) |>
  filter(FoldEnrichment > 2) |>
  nrow()
```

SimpleGraphical Views

We can use various plots to look at the results. You can try changing the number of categories shown and the metric to plot on the x axis. With so many hits you can only ever show a selection of the top hits though.

```
enrichgo_results |>
  barplot(showCategory = 20)
```

```
enrichgo_results |>
  dotplot(
    showCategory = 20,
    x="Count"
  )
```

Summary Views

You can try a category network plot to look at connections between terms. As before you can only show a small number of categories here, and there likely isn't space to show the gene names (though you can try if you like). With smaller numbers of hits this works nicely to summarise both the categories and the genes involved.

```
enrichgo_results |>
  cnetplot(
    showCategory=15,
    node_label = "category"
  )
```

You can also try the treeview. This will accommodate a somewhat larger number of categories. You can play around with the number of categories shown and the number of clusters into which they are divided

```
enrichgo_results |>
    pairwise_termsim() |>
    treeplot(
        showCategory=50,
        nCluster=10
    )
```

Finally from the standard plots we'll do an upset plot. We'll also take this opportunity to illustrate filtering the data before sending it to a plot. This same filtering would work for any of the plot types and you can use any standard `dplyr` filters on the data to select just the categories you want to show.

```
enrichgo_results |>
  filter(str_detect(Description,"negative regulation")) |>
  upsetplot(
    n=6
  )
```

Custom Plot

Personally, I like showing the results as a graph of enrichment vs significance. It's not easy to label the categories, but it helps to illustrate the degree of difference in how interesting different hits are.

```
enrichgo_results |>
  as_tibble() |>
  ggplot(aes(x=FoldEnrichment, y=-10*log10(p.adjust), label=Description)) +
  geom_point() +
  ggrepel::geom_text_repel(
    data = . %>% filter(p.adjust<0.001, FoldEnrichment > 5),
    size=3
  )
```

For the plot we transform the p value into a Phred score (`-10 * log10(p)`) to put it on a nicer scale.

Exercise 2: Quantitative Enrichment Analysis

ClusterProfiler also has an implementation of the GSEA algorithm so we can run an analysis using this. We can use the same data we used for the categorical analysis, but instead of filtering based on a statistical test we can just use the shrunken Log2FoldChange from DESeq as the quantitative metric on which to analyse gene sets.

Creating the named vector

For this analysis we need a named vector containing our quantitative values. We'll use the same data as before. If you don't have it you can reload it as before.

```
read_delim("KAT6AB_knockout.txt") -> kat6ab
```

We need to order the data by descending fold change and then extract the fold change values and the associated gene names.

First we sort.

```
kat6ab |>
  arrange(desc(ShrunkLFC)) -> kat6ab
```

We can view the distribution of values we have

```
kat6ab |>
  mutate(index=1:n()) |>
  ggplot(aes(x=index, y=ShrunkLFC)) +
  geom_line() +
  geom_hline(yintercept = 0)
```

Now we extract the ShrunkLFC values into a vector, and then name those values with the gene names.

```
kat6ab |>
  pull(ShrunkLFC) -> kat6ab_slfc

kat6ab |>
  pull(Gene) -> names(kat6ab_slfc)
```

We can look at the first few items in the vector to check we can see the names and that the values look right.

```
kat6ab_slfc[1:10]
```

Running GSEA

Now we have the data we can run GSEA. As before we're just going to analyse the Biological Process subset of Gene Ontology for speed but in a more complete analysis you could look at the other categories as well.

```
gseGO(
  kat6ab_slfc,
  ont="BP",
  OrgDb = "org.Mm.eg.db",
  keyType = "SYMBOL",
  minGSSize = 10,
  maxGSSize = 100
) -> gsego_result
```

Viewing tabular results

The result can be directly converted to a tibble for viewing, and you could easily save this using `write_tsv`.

```
gsego_result |>
  as_tibble() |>
  head(n=30)
```

Note that most of the strongest hits have negative NES (normalised enrichment score) values, meaning they are skewed to low fold changes, not high. As before we can filter the result to look at the positive values separately from the negative.

```
gsego_result |>
  as_tibble() |>
  filter(NES > 0) |>
  head(n=30)
```

Plotting GSEA plots

You can try a few of these, using either `gseaplot` or `gseaplot2` to look at why GSEA picked these hits out.

```
gsego_result |>
  gseaplot(
    geneSetID = 1,
    title=gsego_result$Description[1]
  )

gsego_result |>
  gseaplot2(
    geneSetID = 7,
    title=gsego_result$Description[7]
  )
```

Dotplots

We'll try something a bit cleverer with the `dotplot`. We could use the simple formulation we saw before, but here we'll split the results by those which go up and down and then select the top 20 hits from each to plot.

```
gsego_result |>
  mutate(direction=if_else(NES>0, "Positive","Negative")) |>
  group_by(direction) |>
  slice(1:20) |>
  dotplot(showCategory=40, x="NES") +
  facet_grid(rows=vars(direction), scale="free_y") +
  geom_vline(xintercept = 0)
```

Enrichment Map Plots

We can construct an enrichment map plot, but the problem is there will be too much information on there to show all of the labels. We can show the whole thing to start with to get an idea of how much structure is in there.

We'll limit ourselves to the positive NES values.

```
gsego_result |>
  filter(NES > 0) |>
  pairwise_termsim() |>
  emapplot(
    min_edge=0.5,
    showCategory=100,
    node_label="none"
)
```

If we wanted a more interactive view then we can turn this into a `plotly` plot which will then give us labels which tell us what each category is when we mouse over it.

```
gsego_result |>
  filter(NES > 0) |>
  pairwise_termsim() |>
  emapplot(
    min_edge=0.5,
    showCategory=100,
    node_label="category"
  ) -> emap_plot

emap_plot$layers[[2]]$mapping$label = emap_plot$data$label

plotly::ggplotly(emap_plot, width = 800, height=800)
```