Gene set enrichment analysis with tmod

January Weiner

2020-06-18

Abstract

The package `tmod` provides blood transcriptional modules described by Chaussabel et al. (2008) and by Li et al. (2014) as well as metabolic profiling clusters from Weiner et al. (2012). Furthermore, the package includes tools for testing the significance of enrichment of the modules as well as visualisation of the features (genes, metabolites etc.) and modules. This vignette is a tutorial for the package.

Contents

Important note on terminology

tmod is a package for *feature set / module* enrichment. How does that differ from *gene set* enrichment?

Firstly, the statistical tests and algorithms for gene set enrichment analysis work just as well on any other type of data. Whenever you can assign some feature (genes, proteins, metabolites) into distinct classes, you can use one or all feature set enrichment algorithms.

Secondly, Subramanian et al. (2005) introduced a fair amount of confusion by calling their particular algorithm "GSEA", short for "gene set enrichment analysis".

Thirdly, *tmod* was first created to allow an analysis of *transcriptional modules*.

For these reasons, in the following, I refer to *modules* (that is, sets of features), which can be gene sets, metabolite sets or other feature sets. Also, I am referring to *features* rather than *genes* in most cases.

Basic data analysis

The Gambia data set

In the following, we will use the Egambia data set included in the package. The data set has been generated by Maertzdorf et al. (2011) and has the GEO ID GSE28623. The data is already background corrected and normalized, so we can proceed with a differential gene expression analysis. Note that only a bit over 5000 genes from the original set of over 45000 probes is included.

```r
library(limma)
library(tmod)
data(Egambia)
design <- cbind(Intercept=rep(1, 30), TB=rep(c(0,1), each= 15))
E <- as.matrix(Egambia[,-c(1:3)])
fit <- eBayes( lmFit(E, design))
tt <- topTable(fit, coef=2, number=Inf,
  genelist=Egambia[,1:3] )

head(tt, 10)
```

```
##       GENE_SYMBOL                                                       GENE_NAME     EG      logFC   A
## 4178      FAM20A                    family with sequence similarity 20, member A"  54757   2.955829   4.0
## 20799     FCGR1B        Fc fragment of IgG, high affinity Ib, receptor (CD64)"      2210   2.391490  13.4
## 4122      BATF2        basic leucine zipper transcription factor, ATF-like 2     116071   2.680837  10.3
## 23567    ANKRD22                                      ankyrin repeat domain 22  118932   2.763908   8.0
## 20498     SEPT4                                                     septin 4    5414   3.286528   4.2
## 20360     CD274                                               CD274 molecule   29126   2.377399   7.3
## 2513       AIM2                                         absent in melanoma 2    9447   1.966342   9.9
## 24032    GOLSYN                                      Golgi-localized protein   55638  -2.534812   2.2
## 1337       ETV7                                               ets variant 7   51513   2.844012   8.0
## 467     SERPING1 serpin peptidase inhibitor, clade G (C1 inhibitor), member 1"      710   2.639069   7.7
```

OK, we see some of the genes known to be prominent in the human host response to TB. We can display one of these using tmod's showGene function (it's just a boxplot combined with a beeswarm, nothing special):

```r
group <- rep( c("CTRL", "TB"), each=15)
showGene(E["20799",], group,
  main=Egambia["20799", "GENE_SYMBOL"])
```

FCGR1B
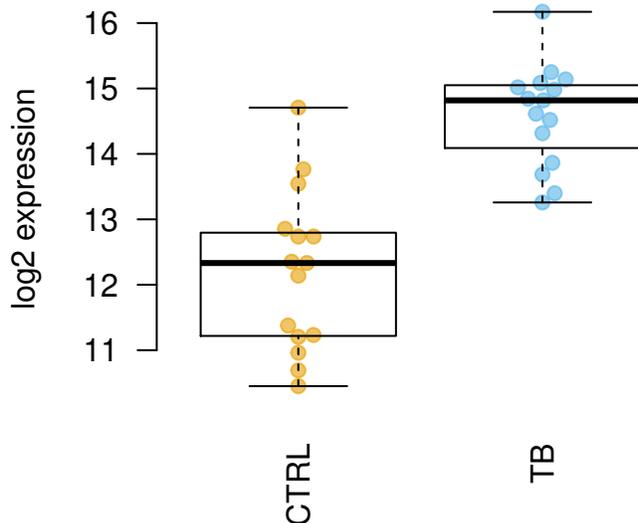
Fine, but what about the modules?

Transcriptional module analysis

There are two main functions in tmod to understand which modules are significantly enriched[1]

The first one, tmodHGtest, is simply a hypergeometric test on two groups of genes: 'foreground' (fg), or the list of differentially expressed genes, and 'background' (bg) – the gene universe, i.e., all genes present in the analysis. The gene identifiers used currently by tmod are HGNC identifiers, and we will use the `GENE_SYMBOL` field from the Egambia data set.

In this particular example, however, we have almost no genes which are significantly differentially expressed after correction for multiple testing: the power of the test with 10 individuals in each group is too low. For the sake of the example, we will therefore relax our selection. Normally, I'd use a q-value threshold of at least 0.001.

```
fg <- tt$GENE_SYMBOL[tt$adj.P.Val < 0.05 & abs( tt$logFC ) > 1]
res <- tmodHGtest(fg=fg, bg=tt$GENE_SYMBOL)
res
```

```
##                 ID                                          Title b  B  n    N        E        P.Val
## LI.M112.0 LI.M112.0                     complement activation (I) 4 11 47 4826 37.33849 2.480096e
## LI.M11.0   LI.M11.0                       enriched in monocytes (II) 4 20 47 4826 20.53617 3.414323e
## LI.M75       LI.M75                        antiviral IFN signature 3 10 47 4826 30.80426 9.906126e
## LI.S4         LI.S4                     Monocyte surface signature 3 10 47 4826 30.80426 9.906126e
## LI.S5         LI.S5                           DC surface signature 4 34 47 4826 12.08010 2.957367e
## LI.M165     LI.M165     enriched in activated dendritic cells (II) 3 19 47 4826 16.21277 7.521410e
## LI.M4.3     LI.M4.3 myeloid cell enriched receptors and transporters 2  5 47 4826 41.07234 9.112727e
## LI.M16       LI.M16               TLR and inflammatory signaling 2  5 47 4826 41.07234 9.112727e
```

The columns in the above table contain the following:

- **ID** The module ID. IDs starting with "LI" come from Li et al. (S. Li et al. 2014), while IDs starting with "DC" have been defined by Chaussabel et al. (Chaussabel et al. 2008).

---

[1]If you work with limma, there are other, more efficient and simpler to use functions. See "Working with limma" below.

- **Title** The module description
- **b** Number of genes from the given module in the fg set
- **B** Number of genes from the module in the bg set
- **n** Size of the fg set
- **N** Size of the bg set
- **E** Enrichment, calcualted as (b/n)/(B/N)
- **P.Value** P-value from the hypergeometric test
- **adj.P.Val** P-value adjusted for multiple testing using the Benjamini-Hochberg correction

Well, IFN signature in TB is well known. However, the numbers of genes are not high: n is the size of the foreground, and b the number of genes in fg that belong to the given module. N and B are the respective totals – size of bg+fg and number of genes that belong to the module that are found in this totality of the analysed genes. If we were using the full Gambia data set (with all its genes), we would have a different situation.

Another approach is to sort all the genes (for example, by the respective p-value) and perform a U-test on the ranks of (i) genes belonging to the module and (ii) genes that do not belong to the module. This is a bit slower, but often works even in the case if the power of the statistical test for differential expression is low. That is, even if only a few genes or none at all are significant at acceptable thresholds, sorting them by the p-value or another similar metric can nonetheless allow to get meaningful enrichments[2].

Moreover, we do not need to set arbitrary thresholds, like p-value or logFC cutoff.

```
l    <- tt$GENE_SYMBOL
res2 <- tmodUtest(l)
head(res2)
```

```
##               ID                                Title      U  N1       AUC      P.Value    adj.P.Val
## LI.M37.0 LI.M37.0 immune activation - generic cluster 352659 100 0.7462103 1.597067e-17 5.525852e-15
## LI.M37.1 LI.M37.1           enriched in neutrophils (I)  50280  12 0.8703781 4.530577e-06 6.569127e-04
## LI.S4       LI.S4            Monocyte surface signature  43220  10 0.8974252 6.853638e-06 6.569127e-04
## LI.M75     LI.M75               antiviral IFN signature  42996  10 0.8927741 8.632649e-06 6.569127e-04
## LI.M11.0 LI.M11.0           enriched in monocytes (II)  74652  20 0.7766542 9.492958e-06 6.569127e-04
## LI.M67     LI.M67            activated dendritic cells  28095   6 0.9714730 3.200305e-05 1.811391e-03
```

```
nrow(res2)
```

```
## [1] 25
```

This list makes a lot of sense, and also is more stable than the other one: it does not depend on modules that contain just a few genes. Since the statistics is different, the b, B, n, N and E columns in the output have been replaced by the following:

- **U** The Mann-Whitney U statistics
- **N1** Number of genes in the module
- **AUC** Area under curve – a measure of the effect size

There are two tests in tmod which both operate on an ordered list of genes: `tmodUtest` and `tmodCERNOtest`. The U test is simple, however has two main issues. Firstly, it detects enrichments as well as depletions – that is, modules which are enriched at the bottom of the list (e.g. modules which are never, ever regulated in a particular comparison) will be detected as well. This is often undesirable. Secondly, large modules will be reported as significant even if the actual effect size (i.e., AUC) is modest or very small, just because of the sheer number of genes in a module. Unfortunately, also the reverse is true: modules with a small number of genes, even if they consist of highly up- or down-regulated genes from the top of the list will not be detected.

The CERNO test, described by Yamaguchi et al. (Yamaguchi et al. 2008), is based on Fisher's method of combining probabilities. In summary, for a given module, the ranks of genes from the module are logarithmized,

---

[2]The rationale is that the non-significant p-values are not associated with the test that we are actually performing, but merely used to sort the gene list. Thus, it does not matter whether they are significant or not.

summed and multiplied by -2:

$$f_{CERNO} = -2 \cdot \sum_{i=1}^{N} \ln \frac{R_i}{N_{tot}}$$

This statitic has the χ^2 distribution with $2 \cdot N$ degrees of freedom, where N is the number of genes in a given module and N_{tot} is the total number of genes (Yamaguchi et al. 2008).

The CERNO test is actually much more practical than the U test for most purposes.

```
l    <- tt$GENE_SYMBOL
res2 <- tmodCERNOtest(l)
head( res2 )
```

```
##                ID                                Title     cerno  N1       AUC      cES      P.Valu
## LI.M37.0  LI.M37.0 immune activation - generic cluster 426.35781 100 0.7462103 2.131789 1.824844e-1
## LI.M11.0  LI.M11.0            enriched in monocytes (II) 113.80864  20 0.7766542 2.845216 5.255069e-0
## LI.S4         LI.S4         Monocyte surface signature  76.37298  10 0.8974252 3.818649 1.606057e-0
## LI.M112.0 LI.M112.0            complement activation (I)  73.67987  11 0.8455773 3.349085 1.722322e-0
## LI.M75       LI.M75              antiviral IFN signature  65.29854  10 0.8927741 3.264927 1.045914e-0
## LI.M16       LI.M16    TLR and inflammatory signaling  46.33475   5 0.9790500 4.633475 1.247201e-0
```

Here, the results are similar, however CERNO test was able to detect another module – "TLR and inflammatory signaling". Although only 5 genes are in this module (which is why U test could not detect it), the genes are all on the top of the list of the differentially regulated genes.

Let us now investigate in more detail the module LI.M75, the antiviral interferon signature. We can use the `evidencePlot` function to see how the module is enriched in the list `l`.

```
evidencePlot(l, "LI.M75")
```
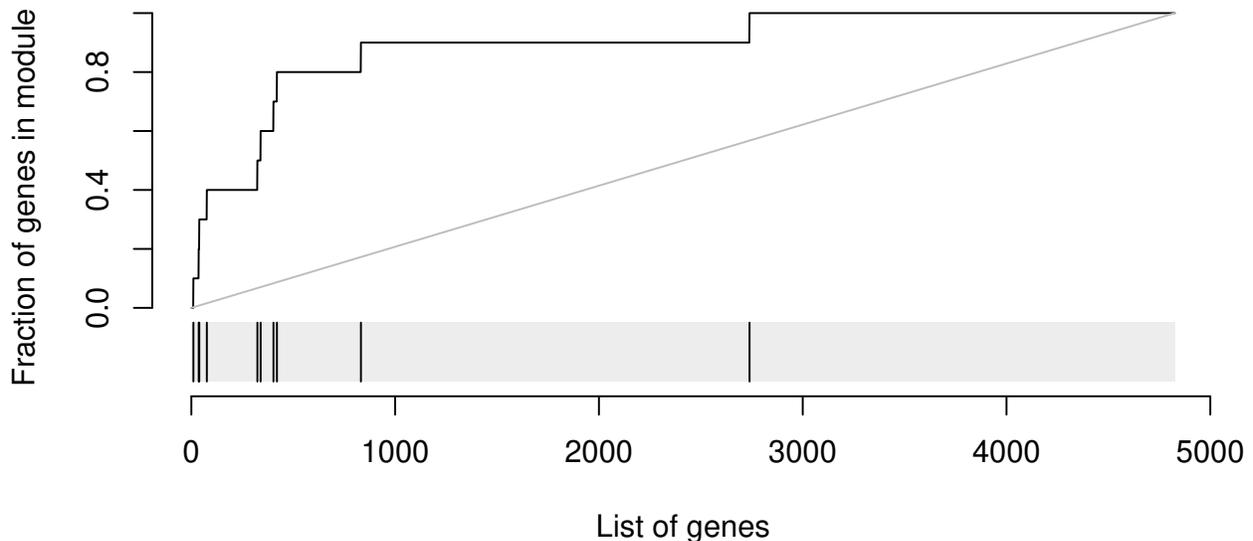

In essence, this is a receiver-operator characteristic (ROC) curve, and the area under the curve (AUC) is related to the U-statistic, from which the P-value in the tmodUtest is calculated, as $\text{AUC} = \frac{U}{n_1 \cdot n_2}$. Both the U statistic and the AUC are reported. Moreover, the AUC can be used to calculate effect size according to the Wendt's formula(Wendt 1972) for rank-biserial correlation coefficient:

$$r = 1 - \frac{2 \cdot U}{n_1 \cdot n_2} = 1 - 2 \cdot \text{AUC}$$

5

In the above diagram, we see that nine out of the 10 genes that belong to the LI.M75 module and which are present in the Egambia data set are ranked among the top 1000 genes (as sorted by p-value).

Working with multiple sets of comparisons

Working with limma

Given the popularity of the limma package, tmod includes functions to easily integrate with limma. In fact, if you fit a design / contrast with limma function lmFit and calculate the p-values with eBayes(), you can directly use the resulting object in tmodLimmaTest and tmodLimmaDecideTests[3].

```
res.l <- tmodLimmaTest(fit, Egambia$GENE_SYMBOL)
length(res.l)
```

```
## [1] 2
```

```
names(res.l)
```

```
## [1] "Intercept" "TB"
```

```
head(res.l$TB)
```

```
##               ID                                Title   cerno N1       AUC      cES      P.Valu
## LI.M37.0   LI.M37.0 immune activation - generic cluster 414.27395 100 0.7255121 2.071370 4.568772e-1
## LI.M11.0   LI.M11.0           enriched in monocytes (II) 105.61794  20 0.7862464 2.640449 7.921155e-08
## LI.M112.0 LI.M112.0            complement activation (I)  75.62229  11 0.8667988 3.437377 8.385947e-08
## LI.S4           LI.S4        Monocyte surface signature  69.97439  10 0.8836794 3.498719 1.838992e-0
## LI.M75         LI.M75            antiviral IFN signature  66.10214  10 0.8645349 3.305107 7.780282e-0
## LI.M67         LI.M67          activated dendritic cells  50.35750   6 0.9712310 4.196458 1.208877e-0
```

The tmodLimmaTest function uses coefficients and p-values from the limma object to order the genes. By default, the genes are ordered by MSD (Minimum Significant Difference), rather than p-value or log fold change.

The MSD is defined as follows:

$$\text{MSD} = \begin{cases} CI.L & \text{if logFC} > 0 \\ -CI.R & \text{if logFC} < 0 \end{cases}$$

Where logFC is the log fold change, CI.L is the left boundary of the 95% confidence interval of logFC and CI.R is the right boundary. MSD is always greater than zero and is equivalent to the absolute distance between the confidence interval and the x axis. For example, if the logFC is 0.7 with 95% CI = [0.5, 0.9], then MSD=0.5; if logFC is -2.5 with 95% CI = [-3.0, -2.0], then MSD = 2.0.

The idea behind MSD is as follows. Ordering genes by decreasing absolute log fold change will include on the top of the list some genes close to background, for which log fold changes are grand, but so are the errors and confidence intervals, just because measuring genes with low expression is loaded with errors. Ordering genes by decreasing absolute log fold change should be avoided.

On the other hand, in a list ordered by p-values, many of the genes on the top of the list will have strong signals and high expression, which results in better statistical power and ultimately with lower p-values – even though the actual fold changes might not be very impressive.

However, by using MSD and using the boundary of the confidence interval to order the genes, the genes on the top of the list are those for which we can *confidently* that the actual log fold change is large. That is

---

[3]The function tmodLimmaDecideTests is described in the next section

because the 95% confidence intervals tells us that in 95% cases, the real log fold change will be anywhere within that interval. Using its bountary closer to the x-axis (zero log fold change), we say that in 95% of the cases the log fold change will have this or larger magnitude (hence, "minimal significant difference").

This can be visualized as follows, using the drop-in replacement for limma's topTable function, tmodLimmaTopTable, which calculates msd as well as confidence intervals. We will consider only genes with positive log fold changes and we will show top 50 genes as ordered by the three different measures:

```r
plotCI <- function(x, ci.l, ci.r, title="") {
  n <- length(x)
  plot(x,
    ylab="logFC", xlab="Index",
    pch=19, ylim=c( min(x-ci.l), max(x+ci.r)),
    main=title)
  segments(1:n, ci.l, 1:n, ci.r, lwd=5, col="#33333333")
}

par(mfrow=c(1,3))

x <- tmodLimmaTopTable(fit, coef="TB")
print(head(x))
```

```
##       logFC.TB        t.TB      msd.TB      SE.TB         d.TB      ciL.TB      ciR.TB     qval.TB
## 34  0.02819016  0.07556852 -0.7277810 0.3730410  0.02879949 -0.7277810 0.7841613 0.99538447
## 36  1.52416640  3.87981294  0.7280616 0.3928453  1.63977633  0.7280616 2.3202712 0.04393162
## 41  0.07888294  0.17834978 -0.8174289 0.4422934  0.09554857 -0.8174289 0.9751948 0.99504430
## 44  0.15321399  0.32385505 -0.8055162 0.4730944  0.19850746 -0.8055162 1.1119442 0.99504430
## 52 -0.23501607 -0.61703279 -0.5368429 0.3808810 -0.24514187 -1.0068750 0.5368429 0.99504430
## 62 -0.31952987 -0.55848332 -0.8399144 0.5721386 -0.50066053 -1.4789741 0.8399144 0.99504430
```

```r
x <- x[ x$logFC.TB > 0, ] # only to simplify the output!
x2 <- x[ order(abs(x$logFC.TB), decreasing=T),][1:50,]
plotCI(x2$logFC.TB, x2$ciL.TB, x2$ciR.TB, "logFC")

x2 <- x[ order(x$qval.TB),][1:50,]
plotCI(x2$logFC.TB, x2$ciL.TB, x2$ciR.TB, "q-value")

x2 <- x[ order(x$msd.TB, decreasing=T),][1:50,]
plotCI(x2$logFC.TB, x2$ciL.TB, x2$ciR.TB, "MSD")
```
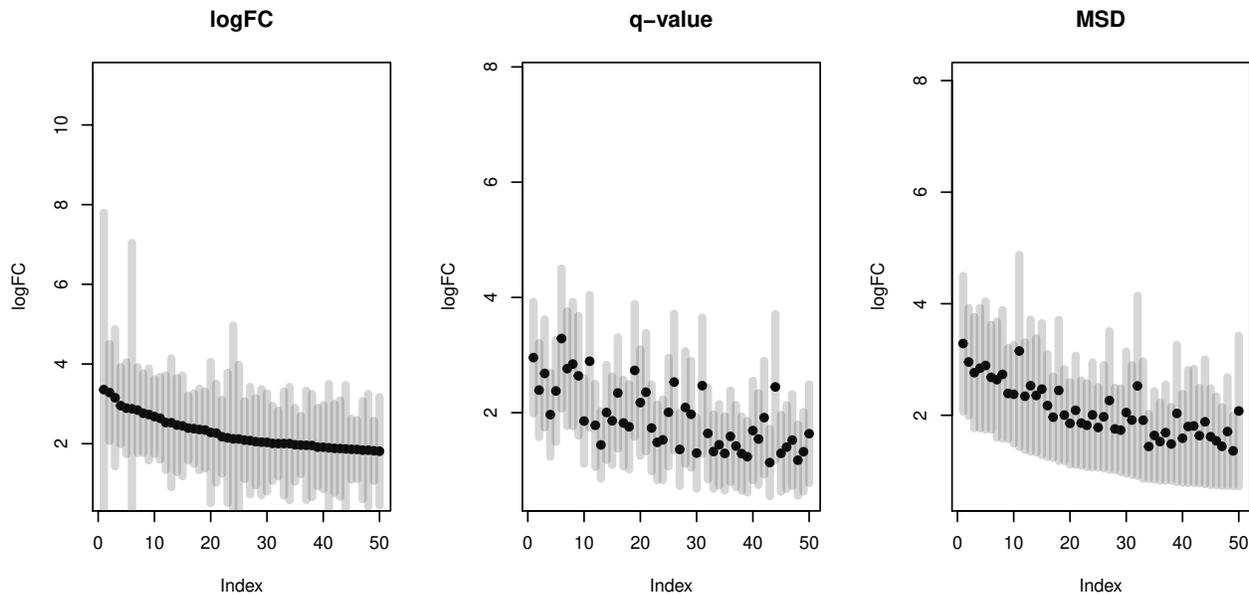
Black dots are logFCs, and grey bars denote 95% confidence intervals. On the left panel, the top 50 genes ordered by the fold change include several genes with broad confidence intervals, which, despite having a large log fold change, are not significantly up- or down-regulated.

On the middle panel the genes are ordered by p-value. It is clear that the log fold changes of the genes vary considerably, and that the list includes genes which are more and less strongly regulated in TB.

The third panel shows genes ordered by decreasing MSD. There is less variation in the logFC than on the second panel, but at the same time the fallacy of the first panel is avoided. MSD is a compromise between considering the effect size and the statistical significance.

What about enrichments?

```
x <- tmodLimmaTopTable(fit, coef="TB", genelist=Egambia[,1:3])
x.lfc  <- x[ order(abs(x$logFC.TB), decreasing=T),]
x.qval <- x[ order(x$qval.TB),]
x.msd  <- x[ order(x$msd.TB, decreasing=T),]

head(tmodCERNOtest(x.lfc$GENE_SYMBOL))
```

```
##                 ID                                        Title    cerno  N1       AUC      cES
## LI.M37.0   LI.M37.0            immune activation - generic cluster 381.20058 100 0.7345239 1.906003 1.990
## LI.M112.0 LI.M112.0                     complement activation (I)  67.57425  11 0.8340036 3.071557 1.583
## LI.M75       LI.M75                       antiviral IFN signature  59.49036  10 0.8714493 2.974518 8.537
## LI.S4         LI.S4                   Monocyte surface signature  58.93059  10 0.8759759 2.946529 1.041
## LI.M67       LI.M67                      activated dendritic cells  44.54543   6 0.9406639 3.712119 1.231
## LI.M165     LI.M165 enriched in activated dendritic cells (II)  84.06285  19 0.7057362 2.212180 2.481
```

```
head(tmodCERNOtest(x.qval$GENE_SYMBOL))
```

```
##                 ID                                       Title    cerno  N1       AUC      cES       P.Value
## LI.M37.0   LI.M37.0 immune activation - generic cluster 427.03180 100 0.7523212 2.135159 1.521279e-18
## LI.M11.0   LI.M11.0            enriched in monocytes (II) 114.85395  20 0.7910112 2.871349 3.691442e-09
## LI.S4         LI.S4          Monocyte surface signature  77.27812  10 0.9164037 3.863906 1.131929e-08
## LI.M112.0 LI.M112.0              complement activation (I)  74.30410  11 0.8595676 3.377459 1.367620e-07
## LI.M75       LI.M75                 antiviral IFN signature  65.41723  10 0.8932932 3.270861 1.001270e-06
## LI.M16       LI.M16       TLR and inflammatory signaling  46.32281   5 0.9790085 4.632281 1.253436e-06
```

```
head(tmodCERNOtest(x.msd$GENE_SYMBOL))
```

```
##                ID                             Title      cerno  N1       AUC      cES     P.Valu
## LI.M37.0  LI.M37.0 immune activation - generic cluster 414.27395 100 0.7255121 2.071370 4.568772e-1
## LI.M11.0  LI.M11.0           enriched in monocytes (II) 105.61794  20 0.7862464 2.640449 7.921155e-08
## LI.M112.0 LI.M112.0            complement activation (I)  75.62229  11 0.8667988 3.437377 8.385947e-08
## LI.S4          LI.S4          Monocyte surface signature  69.97439  10 0.8836794 3.498719 1.838992e-07
## LI.M75        LI.M75              antiviral IFN signature  66.10214  10 0.8645349 3.305107 7.780282e-07
## LI.M67        LI.M67             activated dendritic cells  50.35750   6 0.9712310 4.196458 1.208877e-0
```

In this case, the results of p-value and msd-ordering are very similar.

Comparing tests across experimental conditions

In the above example with the Gambian data set there were only two coefficients calculated in limma, the intercept and the TB. However, often there are several coefficients or contrasts which are analysed simultaneously, for example different experimental conditions or different time points. tmod includes several functions which make it easy to visualize such sets of enrichments.

The object res.l created above using the tmod function tmodLimmaTest is a list of tmod results. Any such list can be directly passed on to functions tmodSummary and tmodPanelPlot, as long as each element of the list has been created with tmodCERNOtest or a similar function. tmodSummary creates a table summarizing module information in each of the comparisons made. The values for modules which are not found in a result object (i.e., which were not found to be significantly enriched in a given comparison) are shown as NA's:

```
head(tmodSummary(res.l), 5)
```

```
## A tmodSummary object
##                ID                        Title AUC.Intercept  q.Intercept    AUC.TB         q.TB
## LI.M11.0  LI.M11.0     enriched in monocytes (II)     0.8145651 0.0001137611 0.7862464 9.671792e-06
## LI.M112.0 LI.M112.0      complement activation (I)            NA           NA 0.8667988 9.671792e-06
## LI.M118.0 LI.M118.0     enriched in monocytes (IV)            NA           NA 0.8377967 2.850219e-03
## LI.M124     LI.M124 enriched in membrane proteins     0.8807517 0.0114869572        NA           NA
## LI.M127     LI.M127    type I interferon response            NA           NA 0.9448247 1.043621e-02
```

We can neatly visualize the above information on a heatmap-like representation:

```
tmodPanelPlot(res.l, text.cex=0.8)
```
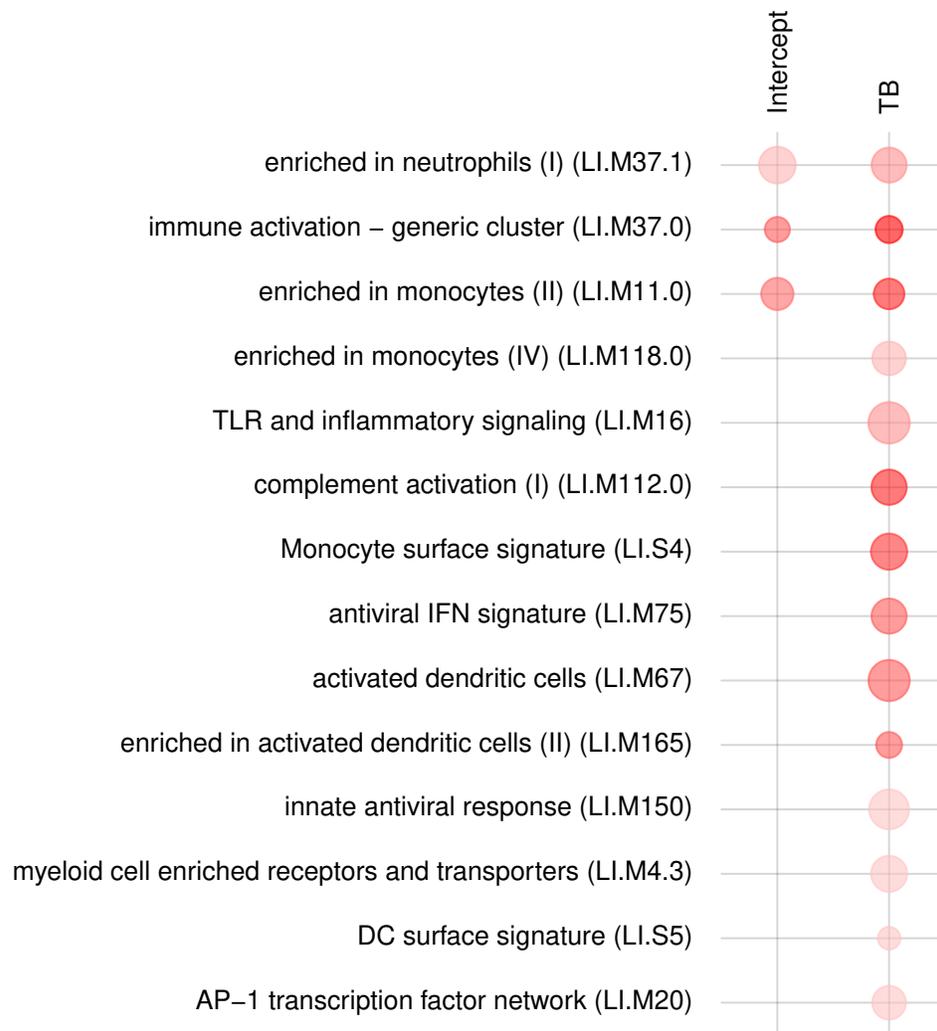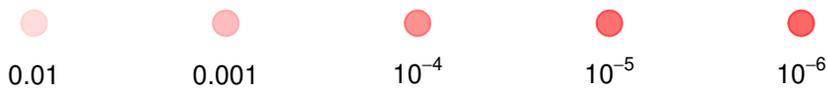
The sizes of the red blobs on the figure correspond to the effect size, that is, the AUC, while the intensity of the color reflects the q-value from the module enrichment test. We can see that also the intercept term is enriched for genes found in monocytes and neutrophils. Note that by default, tmodPanelPlot only shows enrichments with $p < 0.01$, hence a slight difference from the tmodSummary output.

The function tmodPanelPlot has many optional arguments for customization, including options for label sizes, p value thresholds and custom functions for plotting the test results instead of just red blobs.

It is often of interest to see which enriched modules go up, and which go down? Specifically, we would like to see, for each module, how many genes are up-, and how many genes are down-regulated. tmodPanelPlot takes an optional argument, pie, which contains information on significantly regulated genes in modules. We can conveniently generate it from a limma linear fit object with the tmodLimmaDecideTests function:

```
pie <- tmodLimmaDecideTests(fit, genes=Egambia$GENE_SYMBOL)
head(pie$TB[ order( pie$TB[,"up"], decreasing=T), ])
```

```
##           down   N up
## DC.M3.4      0  11  9
## DC.M4.2      0  16  7
## LI.M11.0     0  16  4
## LI.M37.0     0 110  4
## LI.M112.0    0   9  4
## LI.M165      0  24  4
```

```
data(tmod)
tmod$MODULES["DC.M3.4",]
```

```
##              ID     Title Category Annotated
## DC.M3.4 DC.M3.4 Interferon   DC.M3       Yes http://www.biir.net/public_wikis/module_annotation/V2_'
```

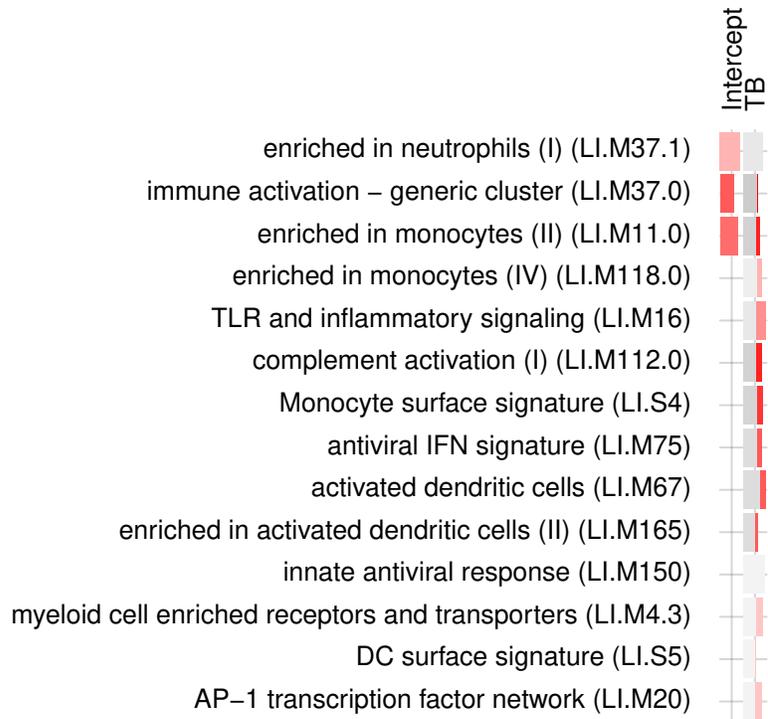The pie object is a list. Each element of the list corresponds to one coefficient and is a data frame with the columns "down", "N" and "up" (in that order). Importantly, all names of the "res.l" list must correspond to an item in the pie list.

```
all(names(pie) %in% names(res.l))
```

```
## [1] TRUE
```

We can now use this information in tmodPanelPlot:

```
tmodPanelPlot(res.l, pie=pie, text.cex=0.8)
```

11

Intercept TB

- enriched in neutrophils (I) (LI.M37.1)
- immune activation – generic cluster (LI.M37.0)
- enriched in monocytes (II) (LI.M11.0)
- enriched in monocytes (IV) (LI.M118.0)
- TLR and inflammatory signaling (LI.M16)
- complement activation (I) (LI.M112.0)
- Monocyte surface signature (LI.S4)
- antiviral IFN signature (LI.M75)
- activated dendritic cells (LI.M67)
- enriched in activated dendritic cells (II) (LI.M165)
- innate antiviral response (LI.M150)
- myeloid cell enriched receptors and transporters (LI.M4.3)
- DC surface signature (LI.S5)
- AP−1 transcription factor network (LI.M20)

P value:

0.01   0.001 10^{-4} 10^{-5} 10^{-6}

Effect size:

0.5   1

A rug-like plot can be also generated:

```
tmodPanelPlot(res.l,
  pie=pie, pie.style="rug",
  grid="between")
```

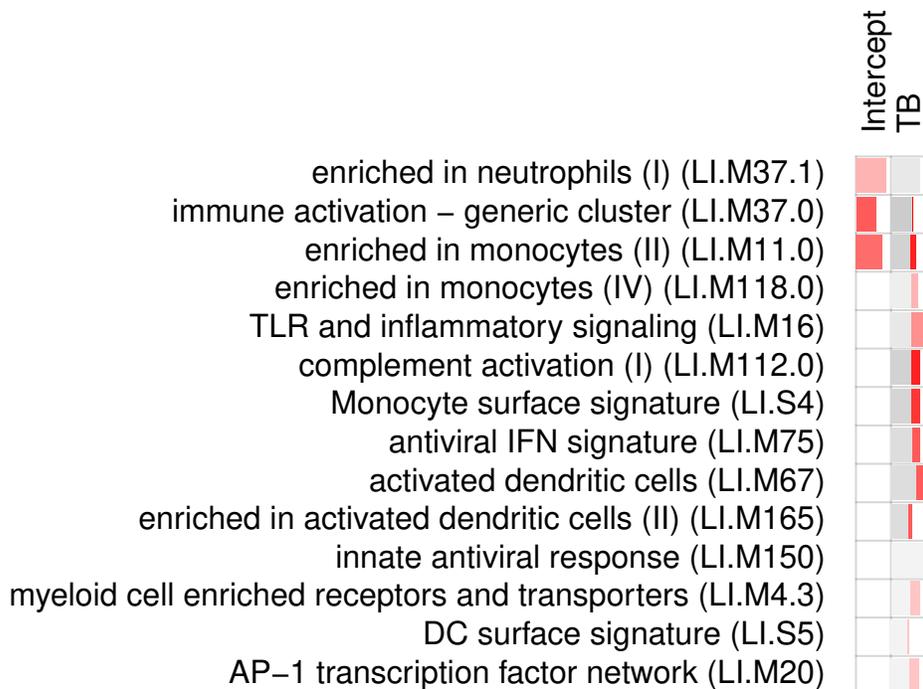|	Intercept	TB
enriched in neutrophils (I) (LI.M37.1)		
immune activation – generic cluster (LI.M37.0)		
enriched in monocytes (II) (LI.M11.0)		
enriched in monocytes (IV) (LI.M118.0)		
TLR and inflammatory signaling (LI.M16)		
complement activation (I) (LI.M112.0)		
Monocyte surface signature (LI.S4)		
antiviral IFN signature (LI.M75)		
activated dendritic cells (LI.M67)		
enriched in activated dendritic cells (II) (LI.M165)		
innate antiviral response (LI.M150)		
myeloid cell enriched receptors and transporters (LI.M4.3)		
DC surface signature (LI.S5)		
AP−1 transcription factor network (LI.M20)		

P value:

| 0.01 | 0.001 | 10^{-4} | 10^{-5} | 10^{-6} |

Effect size:

| 0.5 | | | | 1 |

There is also a more general function, tmodDecideTests that also produces a tmodPanelPlot-compatible object, a list of data frames with gene counts. However, instead of taking a limma object, it requires (i) a gene name, (ii) a vector or a matrix of log fold changes, and (iii) a vector or a matrix of p-values. We can replicate the result of tmodLimmaDecideTests above with the following commands:

```
tt.I <-
  topTable(fit, coef="Intercept", number=Inf, sort.by="n")
tt.TB <- topTable(fit, coef="TB", number=Inf, sort.by="n")
pie2 <- tmodDecideTests(Egambia$GENE_SYMBOL,
  lfc=cbind(tt.I$logFC, tt.TB$logFC),
  pval=cbind(tt.I$adj.P.Val, tt.TB$adj.P.Val))
identical(pie[[1]], pie2[[1]])
```

```
## [1] TRUE
```

Using other sets of modules

By default, tmod uses the modules published by Li et al. (S. Li et al. 2014) (LI). A second set of modules was published by Chaussabel et al. (Chaussabel et al. 2008) (DC); new module definitions were described by Banchereau et al. (Banchereau et al. 2012) and can be found on a public website[4].

---

[4]http://www.biir.net/public_wikis/module_annotation/G2_Trial_8_Modules

Depending on the `mset` parameter to the test functions, either the LI or DC sets are used, or both, if the `mset=all` has been specified.

```
l    <- tt$GENE_SYMBOL
res2 <- tmodUtest(l, mset="all")
head( res2 )
```

```
##                ID                                 Title      U  N1       AUC      P.Value     adj.P.Val
## LI.M37.0 LI.M37.0 immune activation - generic cluster 352659 100 0.7462103 1.597067e-17 9.678227e-15
## DC.M4.2   DC.M4.2                           Inflammation  91352  20 0.9503953 1.674762e-12 5.074530e-10
## DC.M1.2   DC.M1.2                             Interferon  73612  17 0.9004196 5.703006e-09 9.623646e-07
## DC.M3.2   DC.M3.2                           Inflammation  96366  24 0.8361620 6.352241e-09 9.623646e-07
## DC.M5.15 DC.M5.15                            Neutrophils  65289  16 0.8483498 7.240084e-07 8.774982e-05
## DC.M7.29 DC.M7.29                           Undetermined  77738  20 0.8087599 9.084521e-07 9.175366e-05
```

As you can see, the information contained in both module sets is partially redundant.

Functional multivariate analysis

Transcriptional modules can help to understand the biological meaning of the calculated multivariate transformations. For example, consider a principal component analysis (PCA), visualised using the pca3d package (Weiner 2013):

```
mypal <- c("#E69F00", "#56B4E9")
pca <- prcomp(t(E), scale.=TRUE)

col <- mypal[ factor(group) ]
par(mfrow=c(1, 2))
l<-pcaplot(pca, group=group, col=col)

legend("topleft", as.character(l$groups),
       pch=l$pch,
       col=l$colors, bty="n")
l<-pcaplot(pca, group=group, col=col, components=3:4)
legend("topleft", as.character(l$groups),
       pch=l$pch,
       col=l$colors, bty="n")
```
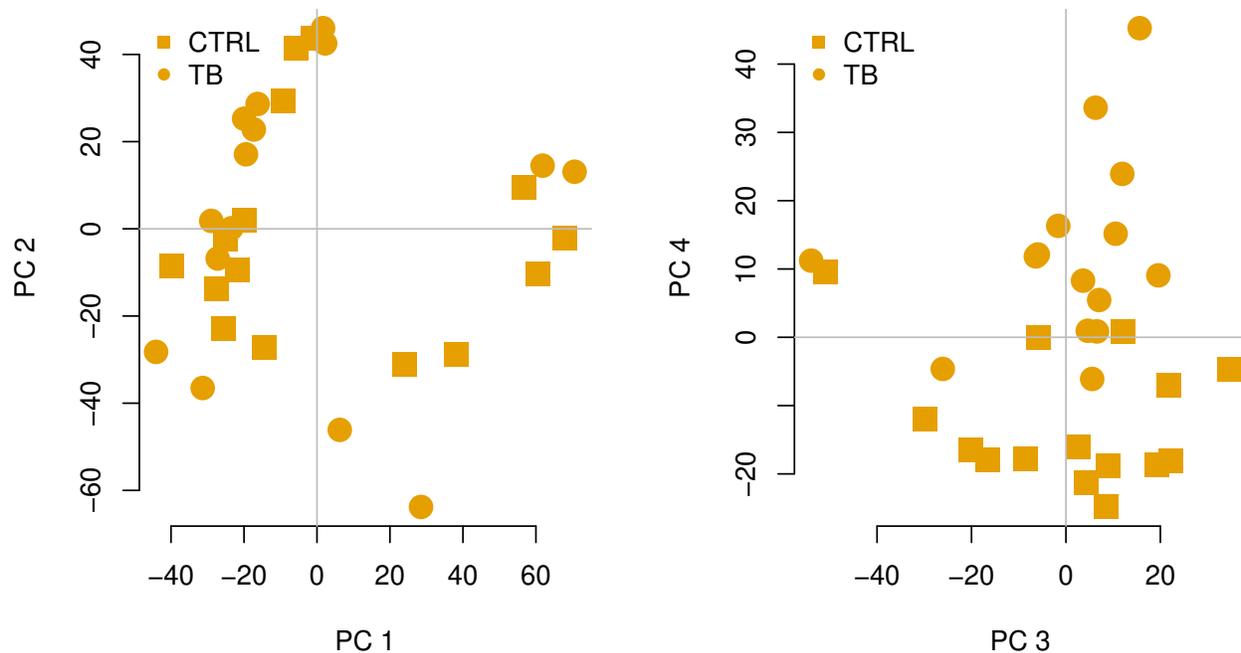
The fourth component looks really interesting. Does it correspond to the modules which we have found before? Each principal component is, after all, a linear combination of gene expression values multiplied by weights (or scores) which are constant for a given component. The i-th principal component for sample j is given by

$$PC_{i,j} = \sum_k w_{i,k} \cdot x_{k,j}$$

where k is the index of the variables (genes in our case), $w_{i,k}$ is the weight associated with the i-th component and the k-th variable (gene), and $x_{k,j}$ is the value of the variable k for the sample j; that is, the gene expression of gene k in the sample j. Genes influence the position of a sample along a given component the more the larger their absolute weight for that component.

For example, on the right-hand figure above, we see that samples which were taken from TB patients have a high value of the principal component 4; the opposite is true for the healthy controls. The genes that allow us to differentiate between these two groups will have very large, positive weights for genes highly expressed in TB patients, and very large, negative weights for genes which are highly expressed in NID, but not TB.

We can sort the genes by their weight in the given component, since the weights are stored in the pca object in the "rotation" slot, and use the tmodUtest function to test for enrichment of the modules.

```
o <- order(abs(pca$rotation[,4]), decreasing=TRUE)
l <- Egambia$GENE_SYMBOL[o]
res <- tmodUtest(l)
head(res)
```

```
##                 ID                                 Title      U  N1       AUC      P.Value   adj.P.Val
## LI.M37.0 LI.M37.0 immune activation – generic cluster 339742 100 0.7188785 3.133111e-14 1.084056e-11
## LI.M37.1 LI.M37.1            enriched in neutrophils (I)  50096  12 0.8671929 5.405722e-06 6.700097e-04
## LI.M75     LI.M75             antiviral IFN signature  43379  10 0.9007267 5.809333e-06 6.700097e-04
## LI.M11.0 LI.M11.0         enriched in monocytes (II)  74343  20 0.7734395 1.185187e-05 1.025187e-03
## LI.S5         LI.S5                 DC surface signature 115007  34 0.7058762 1.711493e-05 1.184353e-03
## LI.M67       LI.M67             activated dendritic cells  28291   6 0.9782503 2.506730e-05 1.445548e-03
```
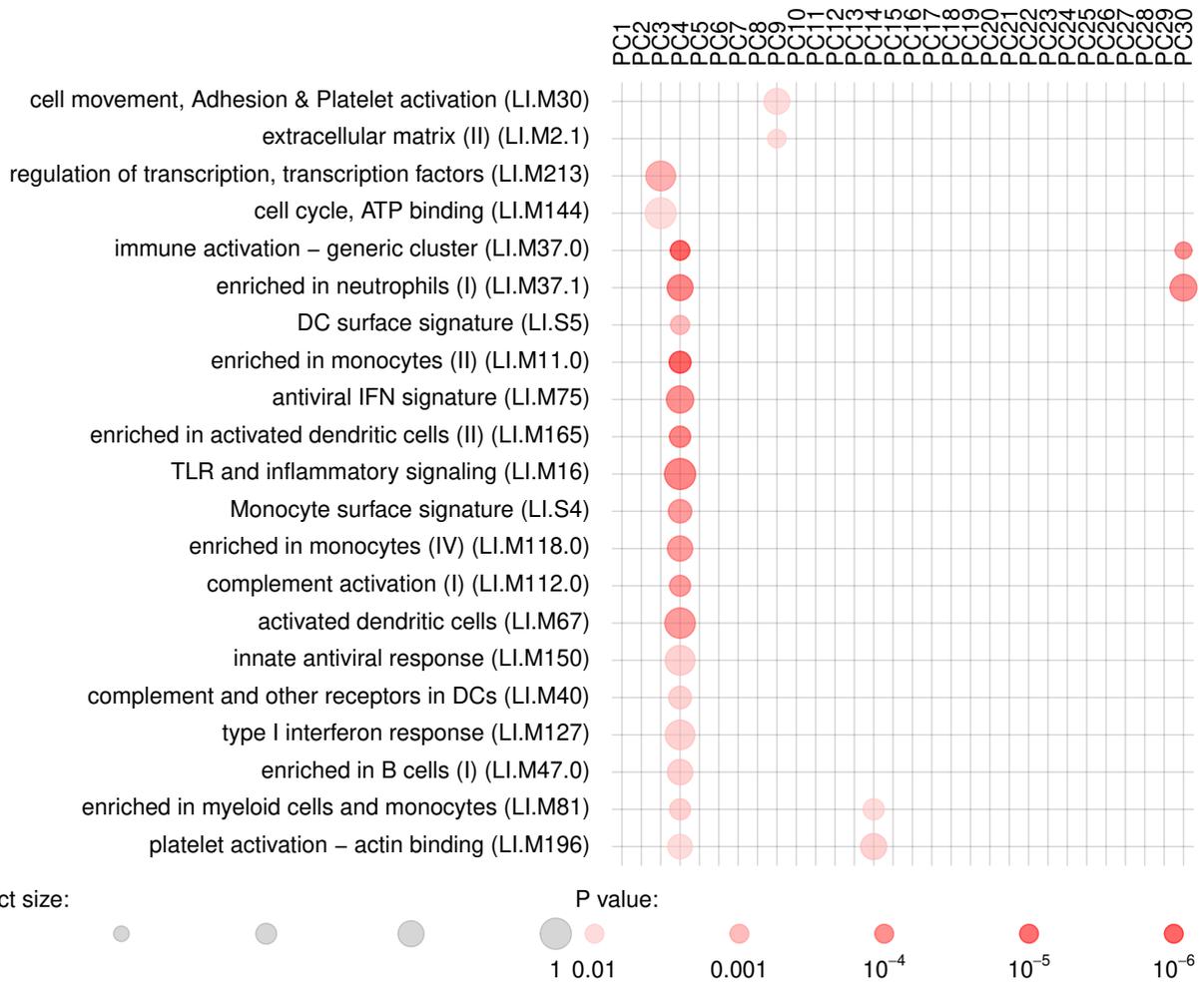
Perfect, this is what we expected: we see that the neutrophil / interferon signature which is the hallmark of the TB biosignature. What about other components? We can run the enrichment for each component and visualise the results using tmod's functions tmodSummary and tmodPanelPlot. Below, we use the filter.empty option to omit the principal components which show no enrichment at all.

```
# Calculate enrichment for each component
gs    <- Egambia$GENE_SYMBOL
# function calculating the enrichment of a PC
gn.f <- function(r) {
    tmodCERNOtest(gs[order(abs(r), decreasing=T)],
              qval=0.01)
}
x <- apply(pca$rotation, 2, gn.f)
tmodSummary(x, filter.empty=TRUE)[1:5,]
```

```
## A tmodSummary object
##                  ID                         Title    AUC.PC3       q.PC3    AUC.PC4       q.PC4 AUC.PC9
## LI.M11.0   LI.M11.0 enriched in monocytes (II)         NA          NA 0.7734395 2.136524e-07      NA
## LI.M112.0 LI.M112.0  complement activation (I)          NA          NA 0.7509865 4.910746e-05      NA
## LI.M118.0 LI.M118.0 enriched in monocytes (IV)         NA          NA 0.8528591 5.027869e-05      NA
## LI.M127     LI.M127 type I interferon response          NA          NA 0.9593030 3.706095e-03      NA
## LI.M144     LI.M144    cell cycle, ATP binding 0.9894257 0.006051848        NA          NA      NA
```

The following plot shows the same information in a visual form. The size of the blobs corresponds to the effect size (AUC value), and their color – to the q-value.
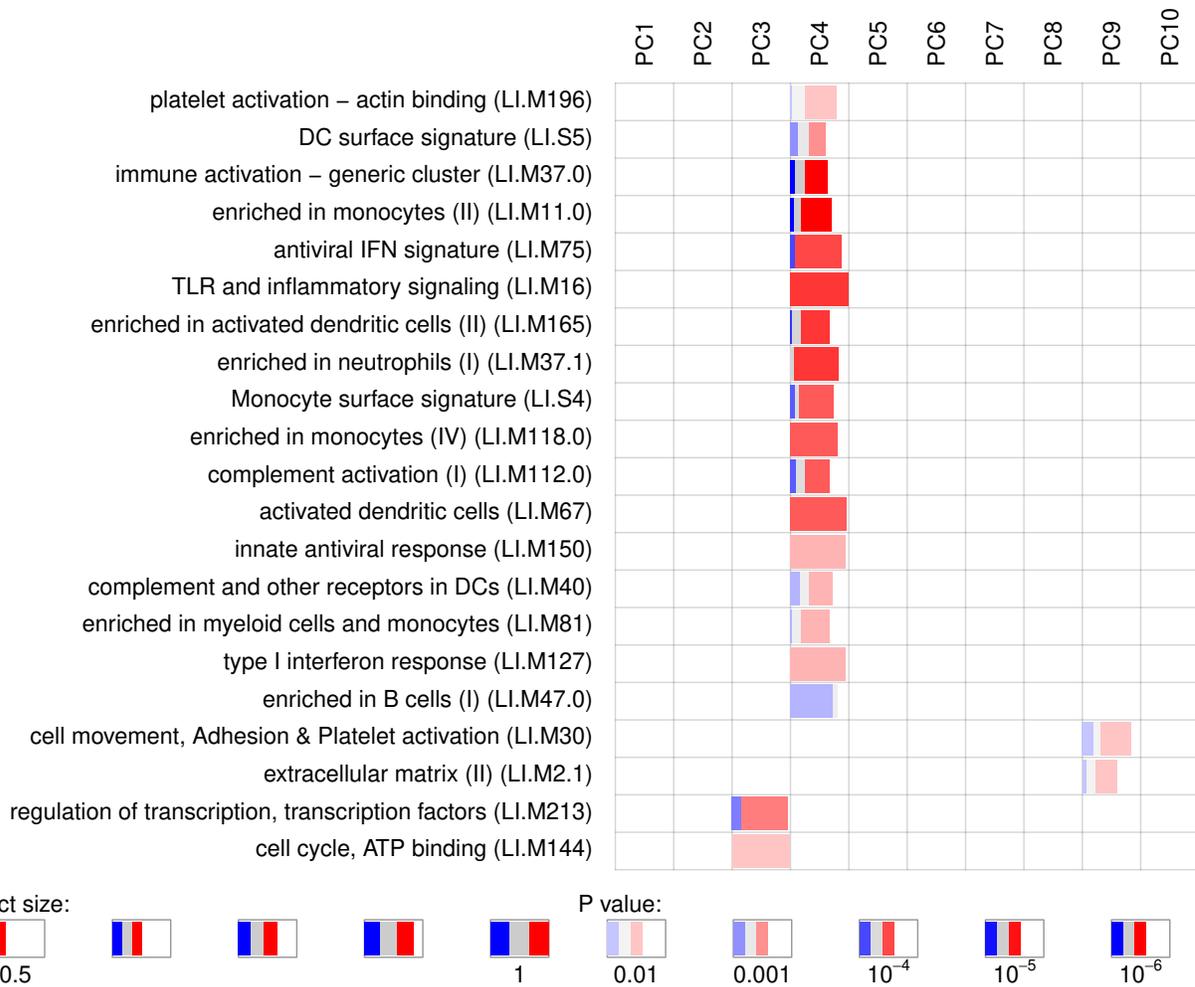
```
tmodPanelPlot(x)
```

cell movement, Adhesion & Platelet activation (LI.M30)
extracellular matrix (II) (LI.M2.1)
regulation of transcription, transcription factors (LI.M213)
cell cycle, ATP binding (LI.M144)
immune activation – generic cluster (LI.M37.0)
enriched in neutrophils (I) (LI.M37.1)
DC surface signature (LI.S5)
enriched in monocytes (II) (LI.M11.0)
antiviral IFN signature (LI.M75)
enriched in activated dendritic cells (II) (LI.M165)
TLR and inflammatory signaling (LI.M16)
Monocyte surface signature (LI.S4)
enriched in monocytes (IV) (LI.M118.0)
complement activation (I) (LI.M112.0)
activated dendritic cells (LI.M67)
innate antiviral response (LI.M150)
complement and other receptors in DCs (LI.M40)
type I interferon response (LI.M127)
enriched in B cells (I) (LI.M47.0)
enriched in myeloid cells and monocytes (LI.M81)
platelet activation – actin binding (LI.M196)

Effect size: 0.5 ... 1

P value: 0.01 0.001 10^{-4} 10^{-5} 10^{-6}

However, we might want to ask, for each module, how many of the genes in that module have a negative, and how many have a positive weight? We can use the function tmodDecideTests for that. For each principal component shown, we want to know how many genes have very large (in absolute terms) weights – we can use the "lfc" parameter of tmodDecideTests for that. We define here "large" as being in the top 25% of all weights in the given component. For this, we need first to calculate the 3rd quartile (top 25% threshold). We will show only 10 components:

```
qfnc <- function(r) quantile(r, 0.75)
qqs <- apply(pca$rotation[,1:10], 2, qfnc)
pie <- tmodDecideTests(gs, lfc=pca$rotation[,1:10], lfc.thr=qqs)
tmodPanelPlot(x[1:10], pie=pie,
  pie.style="rug", grid="between")
```

PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10

platelet activation – actin binding (LI.M196)

DC surface signature (LI.S5)

immune activation – generic cluster (LI.M37.0)

enriched in monocytes (II) (LI.M11.0)

antiviral IFN signature (LI.M75)

TLR and inflammatory signaling (LI.M16)

enriched in activated dendritic cells (II) (LI.M165)

enriched in neutrophils (I) (LI.M37.1)

Monocyte surface signature (LI.S4)

enriched in monocytes (IV) (LI.M118.0)

complement activation (I) (LI.M112.0)

activated dendritic cells (LI.M67)

innate antiviral response (LI.M150)

complement and other receptors in DCs (LI.M40)

enriched in myeloid cells and monocytes (LI.M81)

type I interferon response (LI.M127)

enriched in B cells (I) (LI.M47.0)

cell movement, Adhesion & Platelet activation (LI.M30)

extracellular matrix (II) (LI.M2.1)

regulation of transcription, transcription factors (LI.M213)

cell cycle, ATP binding (LI.M144)

Effect size:

0.5   1

P value:

0.01   0.001 10^{-4} 10^{-5} 10^{-6}

PCA and tag clouds

For another way of visualizing enrichment, we can use the tagcloud package (Weiner 2014). P-Values will be represented by the size of the tags, while AUC – which is a proxy for the effect size – will be shown by the color of the tag, from grey (AUC=0.5, random) to black (1):

```r
library(tagcloud)
w <- -log10(res$P.Value)
c <- smoothPalette(res$AUC, min=0.5)
tags <- strmultline(res$Title)
tagcloud(tags, weights=w, col=c)
```

enriched in
neutrophils (I)

complement and other
receptors in DCs

enriched in
monocytes (IV)

platelet activation
– actin binding

enriched in
monocytes (II)

complement
activation (I)

type I interferon
response

TLR and inflammatory
signaling

TBA

antiviral
IFN signature

immune activation
– generic cluster

enriched in myeloid
cells and monocytes

activated
dendritic cells

TBA

Monocyte surface
signature

enriched in activated
dendritic cells (II)

innate antiviral
response

DC surface
signature

myeloid cell enriched
receptors and transporters

enriched in
B cells (I)

transmembrane and
ion transporters (I)

enriched in
B cells (II)

We can now annotate the PCA axes using the tag clouds; however, see below for a shortcut in tmod.

```r
par(mar=c(1,1,1,1))
o3 <- order(abs(pca$rotation[,3]), decreasing=TRUE)
l3 <- Egambia$GENE_SYMBOL[o3]
res3 <- tmodUtest(l3)
layout(matrix(c(3,1,0,2),2,2,byrow=TRUE),
  widths=c(1/3, 2/3), heights=c(2/3, 1/3))
col <- mypal[ factor(group) ]

# note -- PC4 is now x axis!!
l <- pcaplot(pca, group=group, components=4:3,
  col=col, cex=1.8)

legend("topleft",
  as.character(l$groups),
  pch=l$pch,
  col=l$colors, bty="n")

tagcloud(tags, weights=w, col=c, fvert= 0)
tagcloud(strmultline(res3$Title),
  weights=-log10(res3$P.Value),
  col=smoothPalette(res3$AUC, min=0.5),
  fvert=1)
```
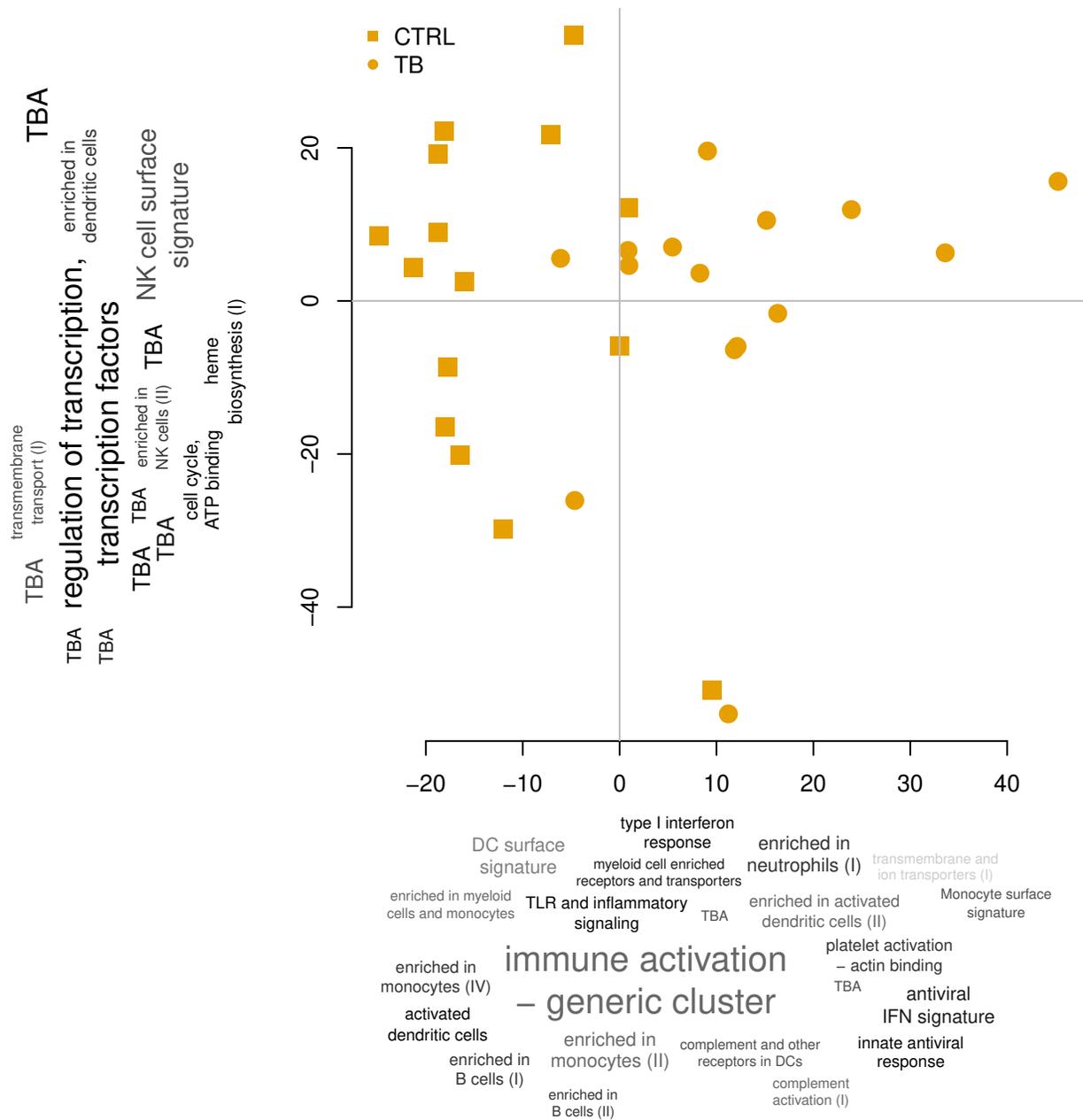
As mentioned previously, there is a way of doing it all with tmod much more quickly, in just a few lines of code:

Note that `plot.params` are just parameters which will be passed to the pca2d function. However, remember that is must be a list.

To plot the PCA, tmod uses the function pcaplot(), but you can actually do it yourself by providing tmodPCA with a suitable function. The only requirement is that the function takes named parameters "pca" and "components":
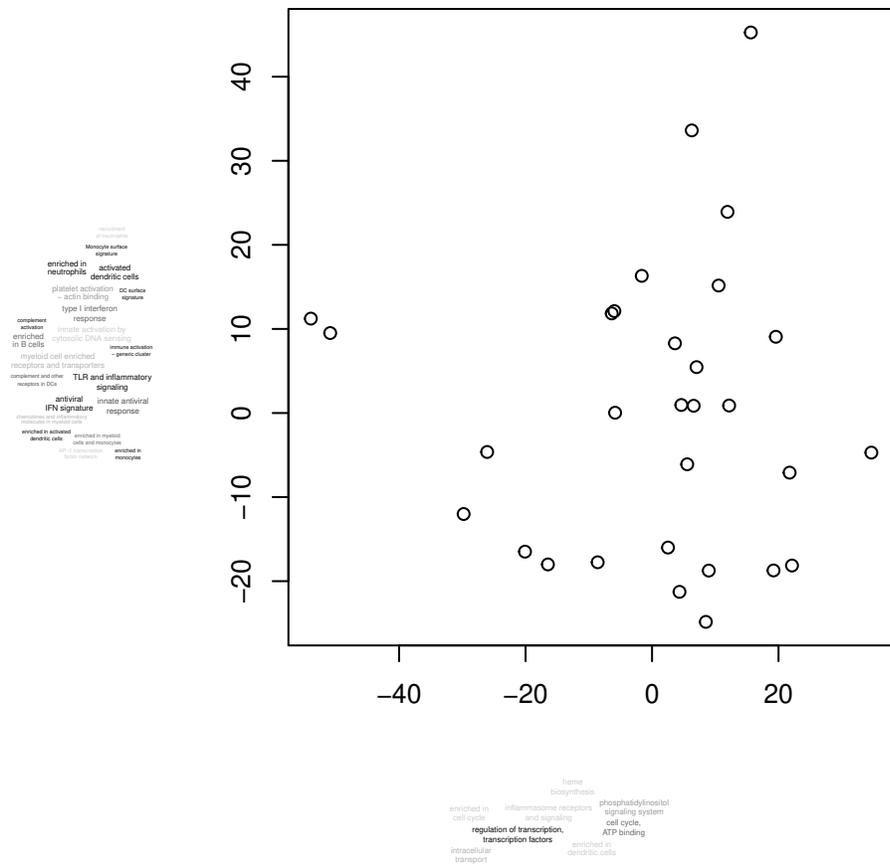
```
plotf <- function(pca, components) {
  id1 <- components[1]
  id2 <- components[2]
  print(id1)
  print(id2)
```

```
  plot(pca$x[,id1], pca$x[,id2])
}

ret <- tmodPCA(pca, genes=Egambia$GENE_SYMBOL,
  components=3:4, plotfunc=plotf)
```

```
## [1] 3
## [1] 4
```

Alternatively, you can use the function "pca2d" from the pca3d package:

```
if(require(pca3d)) plotf <- pca2d
ret <- tmodPCA(pca, genes=Egambia$GENE_SYMBOL,
  components=3:4, plotfunc=plotf, plot.params=list(group=group))
```
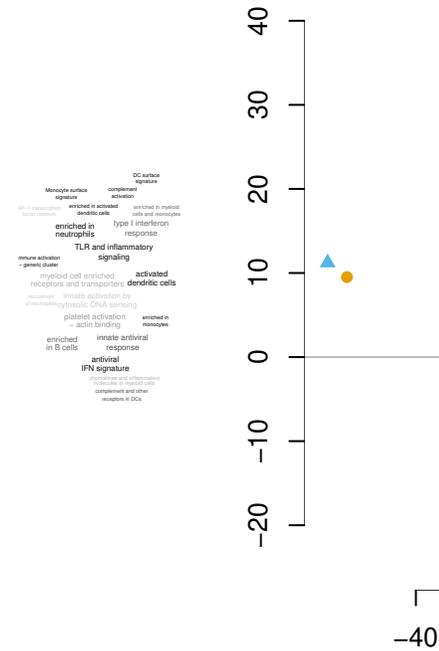
Permutation tests

The GSEA approach (Subramanian et al. 2005) is based on similar premises as the other approaches described here. In principle, GSEA is a combination of an arbitrary scoring of a sorted list of genes and a permutation test. Although the GSEA approach has been criticized from statistical standpoint (Damian and Gorfine 2004), it remains one of the most popular tools to analyze gene sets amongst biologists. In the following, it will be shown how to use a permutation-based test with tmod.

A permutation test is based on a simple principle. The labels of observations (that is, their group assignments) are permutated and a statistic s_i is calculated for each i-th permutation. Then, the same statistic s_o is calculated for the original data set. The proportion of the permutated sets that yielded a statistic s_i equal to or higher than s_o is the p-value for a statistical hypothesis test.

First, we will set up a function that creates a permutation of the Egambia data set and repeats the limma procedure for this permutation, returning the ordering of the genes.

```
permset <- function(data, design) {
  require(limma)
  data <- data[, sample(1:ncol(data)) ]
  fit  <- eBayes(lmFit(data, design))
  tt   <- topTable(fit, coef=2, number=Inf, sort.by="n")
  order(tt$P.Value)
```

```
}
```

In the next step, we will generate 100 random permutations. The `sapply` function will return a matrix with a column for each permutation and a row for each gene. The values indicate the order of the genes in each permutation. We then use the tmod function `tmodAUC` to calculate the enrichment of each module for each permutation.

```
# same design as before
design <- cbind(Intercept=rep(1, 30),
  TB=rep(c(0,1), each= 15))
E      <- as.matrix(Egambia[,-c(1:3)])
N      <- 250  # small number for the sake of example
set.seed(54321)
perms  <- sapply(1:N, function(x) permset(E, design))
pauc   <- tmodAUC(Egambia$GENE_SYMBOL, perms)
dim(perms)
```

```
## [1] 5547  250
```

We can now calculate the true values of the AUC for each module and compare them to the results of the permutation. The parameters "order.by" and "qval" ensure that we will calculate the values for all the modules (even those without any genes in our gene list!) and in the same order as in the perms variable.

```
fit <- eBayes(lmFit(E, design))
tt  <- topTable(fit, coef=2, number=Inf,
  genelist=Egambia[,1:3])
res <- tmodCERNOtest(tt$GENE_SYMBOL, qval=Inf, order.by="n")
all(res$ID == rownames(perms))
```

```
## [1] TRUE
```

```
fnsum <- function(m) sum(pauc[m,] >= res[m,"AUC"])
sums <- sapply(res$ID, fnsum)
res$perm.P.Val <- sums / N
res$perm.P.Val.adj <- p.adjust(res$perm.P.Val)
res <- res[order(res$AUC, decreasing=T),]
head(res[order(res$perm.P.Val),
  c("ID", "Title", "AUC", "adj.P.Val", "perm.P.Val.adj") ])
```

```
##               ID                       Title       AUC      adj.P.Val perm.P.Val.adj
## LI.M16    LI.M16 TLR and inflammatory signaling 0.9790500 7.192190e-05              0
## LI.M59    LI.M59    CCR1, 7 and cell signaling 0.9771973 5.751429e-02              0
## LI.M67    LI.M67       activated dendritic cells 0.9714730 8.363690e-05              0
## LI.M150  LI.M150       innate antiviral response 0.9498859 9.956972e-03              0
## LI.M127  LI.M127       type I interferon response 0.9455715 1.163487e-02              0
## LI.S4      LI.S4    Monocyte surface signature 0.8974252 1.852319e-06              0
```

Although the results are based on a small number of permutations, the results are nonetheless strikingly similar. For more permutations, they improve further. The table below is a result of calculating 100,000 permutations.

```
ID        Title                                           AUC       adj.P.Val
LI.M37.0           immune activation - generic cluster 0.7462103   0.00000
LI.M11.0                     enriched in monocytes (II) 0.7766542   0.00000
LI.M112.0                     complement activation (I) 0.8455773   0.00000
LI.M37.1                   enriched in neutrophils (I) 0.8703781   0.00000
LI.M105                                            TBA 0.8949512   0.00000
LI.S4                     Monocyte surface signature 0.8974252   0.00000
```

```
LI.M150                          innate antiviral response 0.9498859  0.00000
LI.M67                            activated dendritic cells 0.9714730  0.00000
LI.M16                     TLR and inflammatory signaling 0.9790500  0.00000
LI.M118.0                        enriched in monocytes (IV) 0.8774710  0.00295
LI.M75                            antiviral IFN signature 0.8927741  0.00295
LI.M127                        type I interferon response 0.9455715  0.00295
LI.S5                                DC surface signature 0.6833387  0.02336
LI.M188                                               TBA 0.8684647  0.09894
LI.M165        enriched in activated dendritic cells (II) 0.7197180  0.11600
LI.M240                              chromosome Y linked 0.8157171  0.11849
LI.M20                    AP-1 transcription factor network 0.8763327  0.12672
LI.M81            enriched in myeloid cells and monocytes 0.7562851  0.13202
LI.M3                        regulation of signal transduction 0.7763995  0.14872
LI.M4.3   myeloid cell enriched receptors and transporters 0.8859573  0.15675
```

Unfortunately, the permutation approach has two main drawbacks. Firstly, it requires a sufficient number of samples – for example, with three samples in each group there are only $6! = 720$ possible permutations. Secondly, the computational load is substantial.

Accessing the tmod data

The `tmod` package stores its data in two data frames and two lists. This object is contained in a list called `tmod`, which is loaded with `data("tmod")`. The names mimick the various environments from Annotation.dbi packages, but currently the objects are just two lists and two data frames.

- `tmod$MODULES` is a data frame which contains general module information as defined in the supplementary materials for Li et al. (S. Li et al. 2014) and Chaussabel et al. (Chaussabel et al. 2008)
- `tmod$GENES` is a data frame which contains general gene information, including columns with HGNC ("primary"), as well as ENTREZ and REFSEQ identifiers.
- `tmod$MODULES2GENES` is a list with module IDs (same as in the "ID" column of `tmod$MODULES`) as names. Every element of the list is a character vector with IDs ("primary" column of `tmod$GENES`) of the genes which are included in this module.
- `tmod$GENES2MODULES` is a list with gene IDs (same as in the "primary" column of `tmod$GENES`) as names. Every element of the list is a character vector with IDs of the modules in which the gene is found.

Using these variables, one can apply any other tool for the analysis of enriched module sets available, for example, the `geneSetTest` function from the limma package (Smyth et al. (Smyth 2005)). We will first run `tmodUtest` setting the `qval` to `Inf` to get p-values for all modules. Then, we apply the `geneSetTest` function to each module:

```
data(tmod)
res <- tmodUtest(tt$GENE_SYMBOL, qval=Inf)
gstest <- function(x) {
  sel <- tt$GENE_SYMBOL %in% tmod$MODULES2GENES[[x]]
  geneSetTest(sel, tt$logFC)
}
gst <- sapply(res$ID, gstest)
```
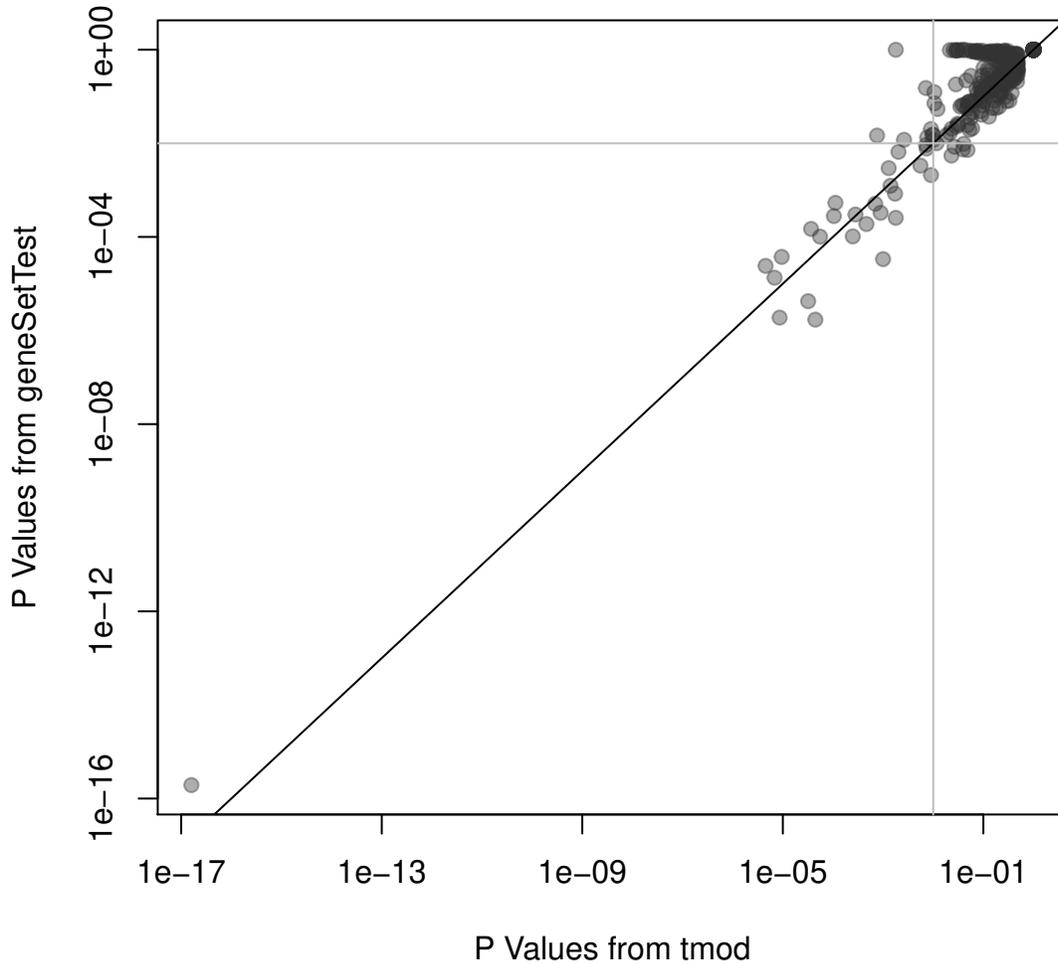
Are the results of both statistical approaches similar? `tmod` uses a very simple statistical test. The approach from `geneSetTest` is more complex, but similar in principle.

```
plot(res$P.Value, gst,
  log="xy", pch=19,
  col="#33333366",
```

```
  xlab="P Values from tmod",
  ylab="P Values from geneSetTest")
abline(0,1)
abline(h=0.01, col="grey")
abline(v=0.01, col="grey")
```


On the plot above, the p-values from `tmod` are plotted against the p-values from `geneSetTest`. As you can see, in this particular example, both methods give very similar results.

Using and creating custom sets of modules

It is possible to use any kind of arbitrary or custom gene set definitions. These custom definition of gene sets takes form of a list which is then provided as the `mset` parameter to the test functions. The list in question must have the following members:

- **MODULES** A data frame which contains at least the columns "ID" and "Title". The IDs must correspond to the names of the `MODULES2GENES` list.
- **GENES** (optional) A data frame which contains at least the column "ID". The gene IDs must correspond to the gene IDs used in `MODULES2GENES`.
- **MODULES2GENES** A list. The names of the list are the IDs from the MODULES data frame. The items in the list are character vectors with names of the genes that are associated with each module.

- **GENES2MODULES** (optional) A list with the reverse mapping from genes to modules. Names on that list must correspond to **GENES$ID**, and the character vector members of the list must correspond to **MODULES$ID**.

tmod will accept a simple list that contains the above fields. However, the function makeTmod can be used conveniently to create a tmod object.

Here is a minimal definition of such a set:

```r
mymset <- makeTmod(
  modules=data.frame(ID=c("A", "B"),
            Title=c("A title",
                    "B title")),
  modules2genes=list(
    A=c("G1", "G2"),
    B=c("G3", "G4"))
)
mymset
```

```
## An object of class "tmod"
##  2 modules, 4 genes
```

Both GENES and GENES2MODULES will be automatically created by makeTmod.

Whether the gene IDs are Entrez, or something else entirely does not matter, as long as they matched the provided input to the test functions.

MSigDB

The MSigDB database from the Broad institute is an interesting collection of gene sets (actually, multiple collections). Unfortunately, MSigDB cannot be distributed or even accessed without a free registration, which imposes a heavy limination on third party tools such as tmod. Use the following guide to download and parse the database such that you can use it with R and tmod.

First, you will need to download the MSigDB in XML format[5]. This file can be accessed at the URL http://software.broadinstitute.org/gsea/msigdb/download_file.jsp?filePath=/resources/msigdb/6.1/msigdb_v6.1.xml – follow the link, register and log in, and save the file on your disk (roughly 113 MB).

Importing MSigDB is easy – tmod has a function specifically for that purpose. Once you have downloaded the MSigDB file, you can create the tmod-compatible R object with one command[6]. However, the tmod function tmodImportMsigDB() can also use this format, look up the manual page:

```r
msig <- tmodImportMSigDB("msigdb_v5.0.xml")
msig
```

```
## An object of class "tmod"
##   8430 modules, 32233 genes
```

That's it – now you can use the full MSigDB for enrichment tests:

```r
res <- tmodCERNOtest(tt$GENE_SYMBOL, mset=msig )
head(res)
```

```
##           ID                                  Title
## M3408 M3408        GSE1432 ctrl vs ifng 24h microglia dn
## M3010 M3010                          Hecker ifnb1 targets
## M3286 M3286    GSE13485 ctrl vs day3 yf17d vaccine pbmc dn
```

---

[5]Note that even if you register with MSig, it is not possible to download the database directly from R in the XML format.
[6]MSigDB gene sets can be also downloaded as "GMT" files. This format contains less information and is therefore less usable.

```
## M3288 M3288    GSE13485 ctrl vs day7 yf17d vaccine pbmc dn
## M3311 M3311 GSE13485 pre vs post yf17d vaccination pbmc dn
## M3347 M3347              GSE14000 unstim vs 4h lps dc dn
##          cerno N1     AUC      cES      P.Value
## M3408 239.0983 39 0.8014227 3.065363 2.967858e-18
## M3010 244.1219 43 0.8459807 2.838626 4.555892e-17
## M3286 247.0915 45 0.7293732 2.745461 1.408943e-16
## M3288 272.2570 54 0.7222067 2.520898 3.626792e-16
## M3311 229.4948 41 0.7272625 2.798718 6.715323e-16
## M3347 272.0698 55 0.7334883 2.473362 9.792737e-16
##          adj.P.Val
## M3408 2.501904e-14
## M3010 1.920308e-13
## M3286 3.959129e-13
## M3288 7.643464e-13
## M3311 1.132204e-12
## M3347 1.375880e-12
```

The results are quite typical for MSigDB, which is quite abundant with similar or overlapping gene sets. As the first results, we see, again, interferon response, as well as sets of genes which are significantly upregulated after yellow fever vaccination – and which are also interferon related. We might want to limit our analysis only to the 50 "hallmark" module categories:

```
sel <- msig$MODULES$Category == "H"
tmodCERNOtest(tt$GENE_SYMBOL, mset=msig[sel] )
```

```
##          ID                                      Title
## M5913 M5913        Hallmark interferon gamma response
## M5921 M5921                        Hallmark complement
## M5911 M5911        Hallmark interferon alpha response
## M5946 M5946                        Hallmark coagulation
## M5890 M5890          Hallmark tnfa signaling via nfkb
## M5930 M5930 Hallmark epithelial mesenchymal transition
## M5932 M5932              Hallmark inflammatory response
## M5953 M5953                  Hallmark kras signaling up
## M5892 M5892          Hallmark cholesterol homeostasis
##          cerno N1     AUC      cES      P.Value
## M5913 221.68317 41 0.7786936 2.703453 8.505170e-15
## M5921 217.81028 56 0.6979148 1.944735 8.607634e-09
## M5911 108.39559 20 0.7563566 2.709890 3.192325e-08
## M5946 179.24580 50 0.6779481 1.792458 1.966824e-06
## M5890 148.95123 47 0.6484665 1.584588 2.657694e-04
## M5930 212.53461 73 0.6371808 1.455717 2.701053e-04
## M5932 184.53035 62 0.6206393 1.488148 3.457724e-04
## M5953 221.76208 82 0.6046637 1.352208 1.790956e-03
## M5892  49.14641 14 0.6138968 1.755229 8.040562e-03
##          adj.P.Val
## M5913 4.252585e-13
## M5921 2.151909e-07
## M5911 5.320542e-07
## M5946 2.458530e-05
## M5890 2.250878e-03
## M5930 2.250878e-03
## M5932 2.469803e-03
## M5953 1.119347e-02
```

```
## M5892 4.466979e-02
```

We see both – the prominent interferon response and the complement activation. Also, in addition, TNF-α signalling via NF-$\kappa\beta$.

Manual creation of tmod module objects: MSigDB

For the purposes of an example, the code below shows how to parse the XML MSigDB file using the R package XML. Essentially, this is the same code that tmodImportMsigDB is using:

```r
library(XML)
foo  <- xmlParse( "/home/january/Projects/R/pulemodule/vignette/msigdb_v5.0.xml" )
foo2 <- xmlToList(foo)
```

There are over 10,000 "gene sets" (equivalent to modules in tmod) defined. Each member of foo2 is a named character vector:

```r
path1 <- foo2[[1]]
class(path1)
```

```
## [1] "character"
```

```r
names(path1)
```

```
##  [1] "STANDARD_NAME"       "SYSTEMATIC_NAME"
##  [3] "HISTORICAL_NAMES"    "ORGANISM"
##  [5] "PMID"                "AUTHORS"
##  [7] "GEOID"               "EXACT_SOURCE"
##  [9] "GENESET_LISTING_URL" "EXTERNAL_DETAILS_URL"
## [11] "CHIP"                "CATEGORY_CODE"
## [13] "SUB_CATEGORY_CODE"   "CONTRIBUTOR"
## [15] "CONTRIBUTOR_ORG"     "DESCRIPTION_BRIEF"
## [17] "DESCRIPTION_FULL"    "TAGS"
## [19] "MEMBERS"             "MEMBERS_SYMBOLIZED"
## [21] "MEMBERS_EZID"        "MEMBERS_MAPPING"
## [23] "FOUNDER_NAMES"       "REFINEMENT_DATASETS"
## [25] "VALIDATION_DATASETS"
```

For our example analysis, we will use only human gene sets. We further need to make sure there are no NULLs in the list.

```r
orgs <- sapply(foo2, function(x) x["ORGANISM"])
unique(orgs)

foo3 <- foo2[ orgs == "Homo sapiens" ]
foo3 <- foo3[ ! sapply(foo3, is.null) ]
```

Next, construct the MODULES data frame. We will use four named fields for each vector, which contain the ID (systematic name), description, category and subcategory:

```r
modules <- t(sapply(foo3,
  function(x)
    x[ c("SYSTEMATIC_NAME", "STANDARD_NAME", "CATEGORY_CODE", "SUBCATEGORY_CODE") ]))
colnames(modules) <- c( "ID", "Title", "Category", "Subcategory" )
modules <- data.frame(modules, stringsAsFactors=FALSE)
```

Then, we create the modules to genes mapping and the GENES data frame. For this, we use the `MEMBERS_SYMBOLIZED` field, which is a comma separated list of gene symbols belonging to a particular

module:

```
m2g <- lapply(foo3,
  function(x) strsplit( x["MEMBERS_SYMBOLIZED"], "," )[[1]])
names(m2g) <- modules$ID

msig <- makeTmod(modules=modules, modules2genes=m2g)
```

From now on, you can use msig with tmod.

Note that it is not necessary to create the members GENES and GENES2MODULES manually. The reverse mapping from genes to modules, GENES2MODULES, will be automatically inferred from MODULES2GENES. If no meta-information on genes is provided in GENES, then a minimal data frame will be created with one column only (ID).

Case study: Metabolic profiling of TB patients

Introduction

One of the main objectives in writing tmod was the ability to analyse metabolic profiling data and other uncommon data sets. In 2012, we have analysed metabolic profiles of serum collected from patients suffering from tuberculosis (TB) and healthy controls (Weiner 3rd et al. 2012). It turned out that there are huge differences between these two groups of individuals, involving amino acid metabolism, lipid metabolism and many others. In the course of the analysis, we found correlations between the metabolites which are not explained fully by the metabolic pathways. For example, cortisol is correlated with kynurenine due to the immunoactive function of these molecules indicating an activation of the immune system, and not because these two molecules are linked by a synthesis process. Vice versa, kynurenine and tryptophan were not directly correlated, even though these molecules are clearly linked by a metabolic process, because tryptophan is not an immune signalling molecule, while kynurenine is.

The tmod package includes both, the data set used in the Weiner et al. paper and the cluster definitions (modules) published therein. In the following, we will use these modules to analyse the metabolic profiles[7].

First, we load the data modules and the data set to analyse.

```
data(modmetabo) ## modules
data(tbmprof)
ids <- rownames(tbmprof)
tb <- factor(gsub("\\..*", "", ids))
sex <- factor( gsub( ".*\\.([MF])\\..*", "\\1", ids))
table(tb, sex)
```

```
##          sex
## tb        F  M
##    HEALTHY 58 34
##    TB      25 19
```

Differential analysis

The metabolic profiling data has not exactly a normal distribution, but that varies from one compound to another. It is possible to normalize it by ranking, but we can simply use the wilcoxon test to see differences between males and females as well as TB patients and healthy individuals.

---

[7]Formally, this is not correct, as the modules were derived from the data set that we are going to analyse, however it serves for demonstration purposes

```r
wcx.tb <- apply(tbmprof, 2, function(x) wilcox.test(x ~ tb, conf.int=T))
wcx.tb <- t(sapply(wcx.tb, function(x) c(x$estimate, x$p.value)))

wcx.sex <- apply(tbmprof, 2, function(x) wilcox.test(x ~ sex, conf.int=T))
wcx.sex <- t(sapply(wcx.sex, function(x) c(x$estimate, x$p.value)))

wcx <- data.frame(ID=colnames(tbmprof),
                  E.tb=wcx.tb[,1], pval.tb=wcx.tb[,2],
                  E.sex=wcx.sex[,1], pval.sex=wcx.sex[,2],
                  row.names=colnames(tbmprof))
```

The data frame contains the results of all tests. We can now test both the healthy/tb comparison and the male/female comparison for enrichment in metabolic profiling modules. Instead ordering the feature identifiers, we use the option "input.order" to determine the sorting.

```r
ids <- wcx$ID
res <- list()
res$tb <- tmodCERNOtest(ids[order(wcx$pval.tb)], mset=modmetabo)
res$tb
```

```
##             ID                                           Title    cerno N1       AUC      cES          P
## ME.107 ME.107                             Amino acids cluster 104.64470 18 0.8824576 2.906797 1.28379
## ME.37   ME.37 Kynurenines, taurocholates and cortisol cluster 116.88147 25 0.8837093 2.337629 2.81609
## MP.2     MP.2                                      Amino Acid  99.16212 28 0.7060786 1.770752 3.35799
```

```r
res$sex <- tmodCERNOtest(ids[order(wcx$pval.sex)], mset=modmetabo)
res$sex
```

```
##             ID              Title    cerno N1       AUC      cES      P.Value    adj.P.Val
## ME.26 ME.26   Hormones cluster 62.47927 10 0.9198068 3.123963 2.923803e-06 0.0001227997
## MS.1   MS.1             Steroid 60.98935 11 0.8732115 2.772243 1.593507e-05 0.0003346365
## ME.69 ME.69 Cholesterol cluster 45.14144 11 0.8190623 2.051883 2.546330e-03 0.0356486262
```

Both these result tables are concordant with previous findings. The enriched modules in male vs. female comparison are what one would expect. In TB, a cluster consisting of kynurenine, bile acids and cortisol is up-regulated, while amino acids go down. We can take a closer look at it using the evidencePlot function.

Why is there a module called "Amino acid cluster" and another one called "Amino acid"? The "cluster" in the name of the module indicates that it has been build by clustering of the profiles, while the other module has been based on the biochemical classification of the molecules. This information is contained in the `Category` column of the `MODULES` data frame:

```r
modmetabo$MODULES[ c("ME.107", "MP.2"), ]
```
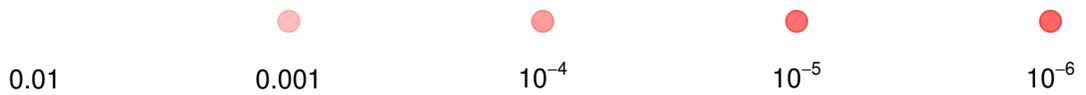
```
##             ID               Title Category
## ME.107 ME.107 Amino acids cluster  Cluster
## MP.2     MP.2          Amino Acid  Pathway
```

To get an overview for both of these comparisons at the same time, we can use the tmodPanelPlot function. The size of the blobs below corresponds to the AUC values from the tables above.

```r
tmodPanelPlot(res)
```
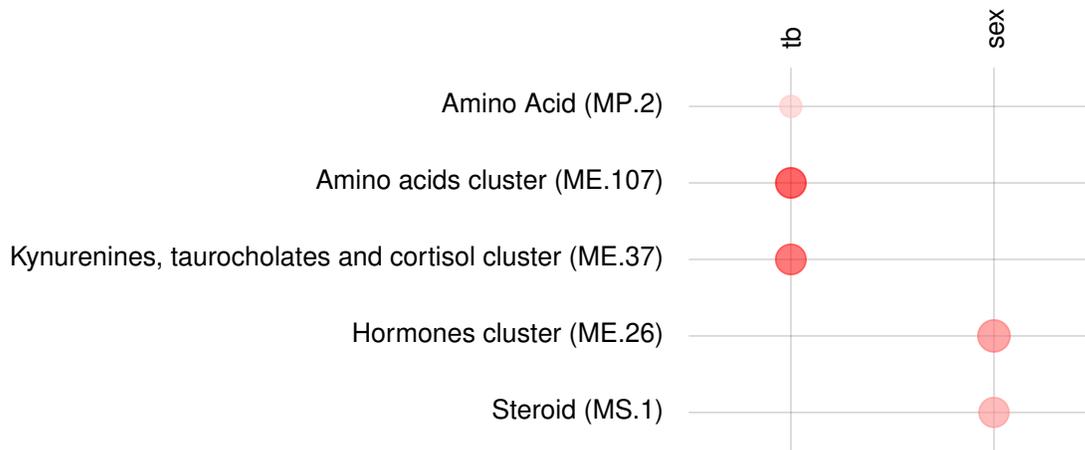
This, unfortunately, does not tell us in which group the metabolites from a given modules are higher. For this, we can use the "estimate" from the wilcox.test above and a parameter for tmodPanelPlot called "pie". To create the value for this parameter – a list that describes, for each condition and for each module, how many metabolites change in one direction, and how many change in the other.
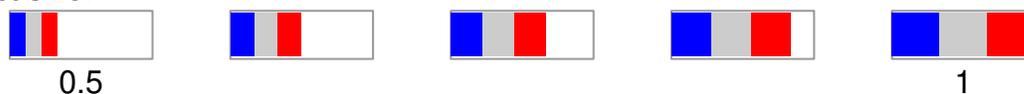
```
pie.data <- wcx[,c("E.sex", "E.tb")]
colnames(pie.data) <- c("sex", "tb")
pie <- tmodDecideTests(wcx$ID, lfc=pie.data, lfc.thr=0.2, mset=modmetabo)
tmodPanelPlot(res, pie=pie, pie.style="rug", grid="between")
```
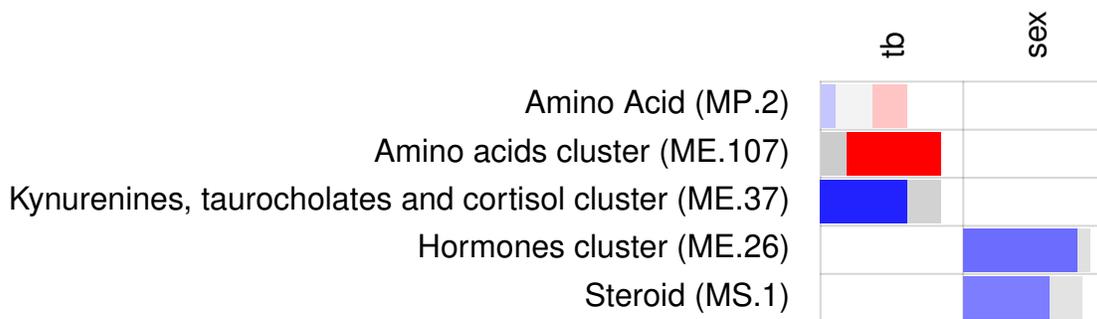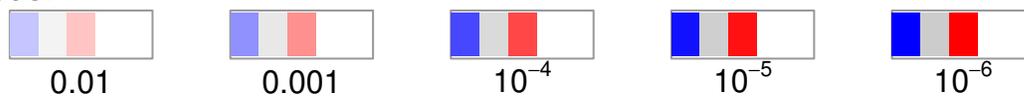
We see now that the cortisol cluster is higher in TB, while amino acids are found at lower concentration in the patients. Also, we see that most of the steroids found (cluster ME.26 and module MS.1) are lower in females. The latter is not surprising if we inspect it closely.

```
wcx <- wcx[order(wcx$pval.sex),]
showModule(wcx[,c("E.sex", "pval.sex")], wcx$ID, "MS.1", mset=modmetabo)
```

```
##                 E.sex      pval.sex         ID                                                   Name
## HMDB00493    -0.86999795 3.042745e-06   HMDB00493       5alpha-androstan-3beta,17beta-diol disulfate
## HMDB00365    -0.64001269 4.033730e-05   HMDB00365                               epiandrosterone sulfate
## HMDB02759    -0.61997366 1.072607e-04   HMDB02759                                 androsterone sulfate
## M.37186      -0.49999995 1.486517e-04     M.37186 5alpha-androstan-3alpha,17beta-diol monosulfate (1)
## HMDB03818.1  -0.38999920 1.541563e-04 HMDB03818.1         4-androsten-3beta,17beta-diol disulfate (2)
## M.32619      -0.35995962 3.418813e-04     M.32619                               pregn steroid monosulfate*
## HMDB03818    -0.45997259 4.349672e-03   HMDB03818         4-androsten-3beta,17beta-diol disulfate (1)
## HMDB01032    -0.27006303 5.284132e-03   HMDB01032         dehydroisoandrosterone sulfate (DHEA-S)
## HMDB02802    -0.10000226 8.852221e-02   HMDB02802                                             cortisone
## HMDB00063    -0.11997203 1.552120e-01   HMDB00063                                               cortisol
## HMDB04026    -0.07998115 3.350142e-01   HMDB04026             21-hydroxypregnenolone disulfate
```

```
i <- "HMDB00493" # what is it?
modmetabo$GENES[i,]
```

```
##                 ID                                          Name Pathway Subpathway      HMDB   KEGG
## HMDB00493 HMDB00493 5alpha-androstan-3beta,17beta-diol disulfate   Lipid     Steroid HMDB00493 C12525
```

```
par(mfrow=c(1,2))
showGene(tbmprof[,i], sex, main=modmetabo$GENES[i, "Name"],
  ylab="Relative abundance")

## now for cortisol cluster
i <- "HMDB00063"
wcx <- wcx[order(wcx$pval.tb),]
showModule(wcx[,c("E.tb", "pval.tb")], wcx$ID, "ME.37",
  mset=modmetabo)[1:10,] # only first 10!
```

```
##                   E.tb        pval.tb         ID                Name      Pathway
## M.47908    -6.999310e-01 2.665735e-14   M.47908             Unknown
## M.32599    -7.999872e-01 2.320130e-10   M.32599 glycocholenate sulfate*       Lipid          Secon
## HMDB00169  -6.299770e-01 5.118925e-09 HMDB00169             mannose Carbohydrate Fructose, Mannose
## Mx.22110   -6.448828e-05 1.379682e-08  Mx.22110     3-hydroxykynurenine   Amino acid
## HMDB00063  -5.399765e-01 1.990892e-08 HMDB00063             cortisol       Lipid
## HMDB00159  -2.900586e-01 2.491091e-08 HMDB00159             phenylalanine   Amino Acid       Phenylalanin
## M.32807    -1.219964e+00 3.577292e-08   M.32807   taurocholenate sulfate       Lipid          Secon
## M.46637    -1.030041e+00 6.660175e-08   M.46637             Unknown
## M.46652    -8.399503e-01 1.420077e-07   M.46652             Unknown
## HMDB00684  -3.100146e-01 1.788895e-07 HMDB00684             kynurenine   Amino Acid
```

```
showGene(tbmprof[,i], tb, main=modmetabo$GENES[i, "Name"],
  ylab="Relative abundance")
```
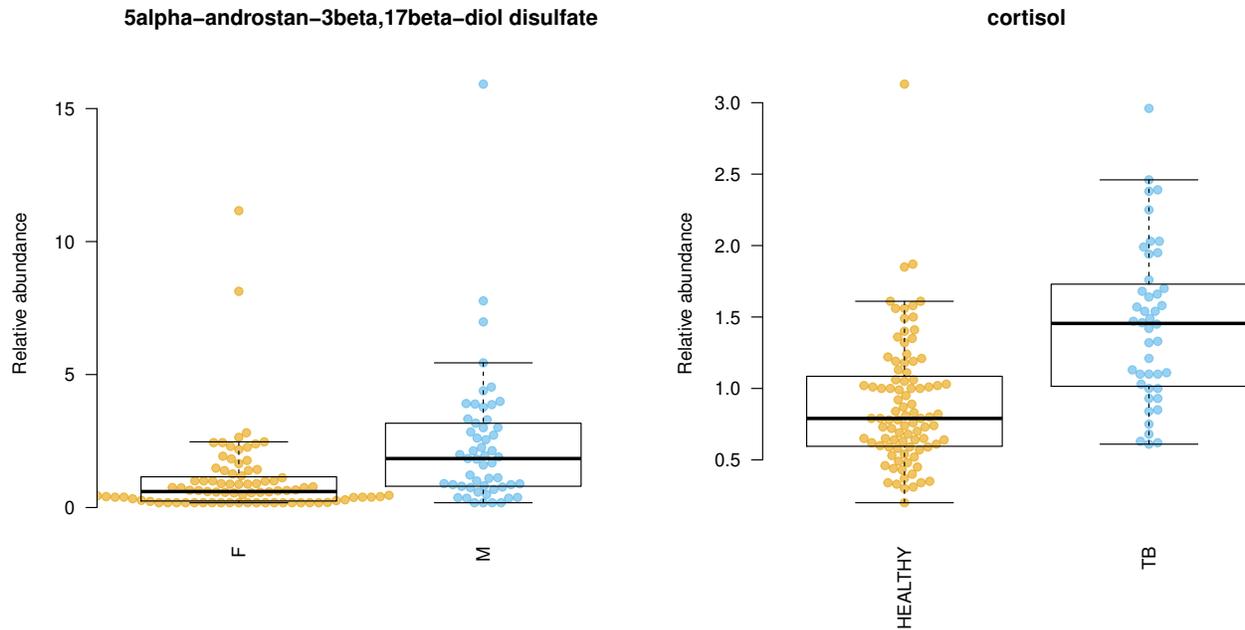
Functional multivariate analysis

We can practically circumvent a gene-by-gene analysis. In fact, we are rarely interested in the p-values associated with single genes or metabolites. There is too many of them, and the statistical power is limited by the sheer number of tests and the requirement of correction for multiple testing. In case you have not read the part on FMA above, "Functional multivariate analysis", in its simplest form, is simply combining a principal component analysis (PCA) with enrichment analysis. PCA lets us explore where the variance in the data is; enrichment analysis allows us to interpret the principal components in functional terms.
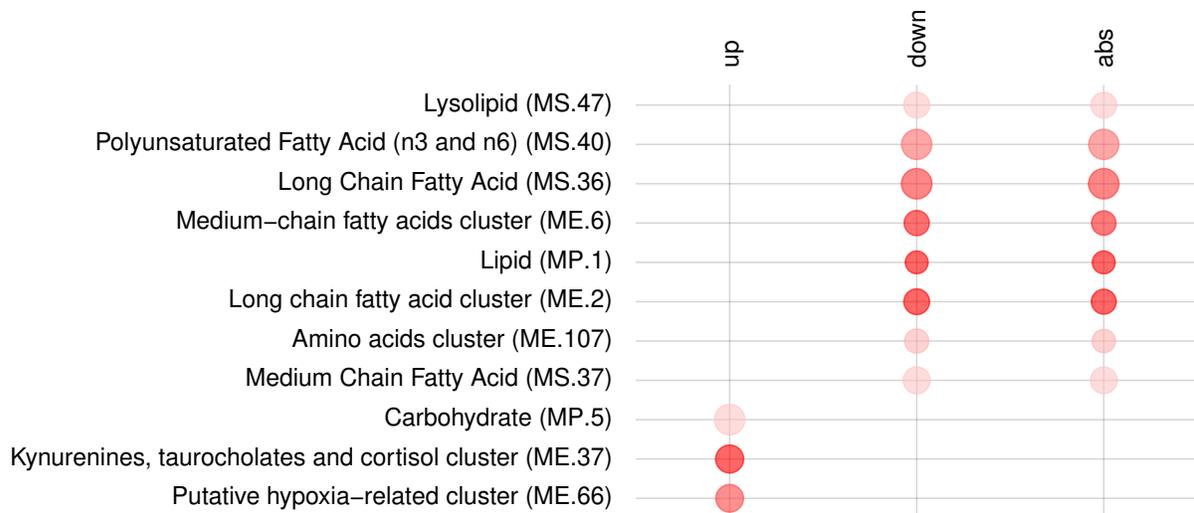
In tmod, it can be done in a few lines of code:

```
pca <- prcomp(tbmprof, scale.=T)
ret <- tmodPCA(pca, genes=colnames(tbmprof), mset=modmetabo,
  plot.params=list(group=tb, legend="topright"))
```
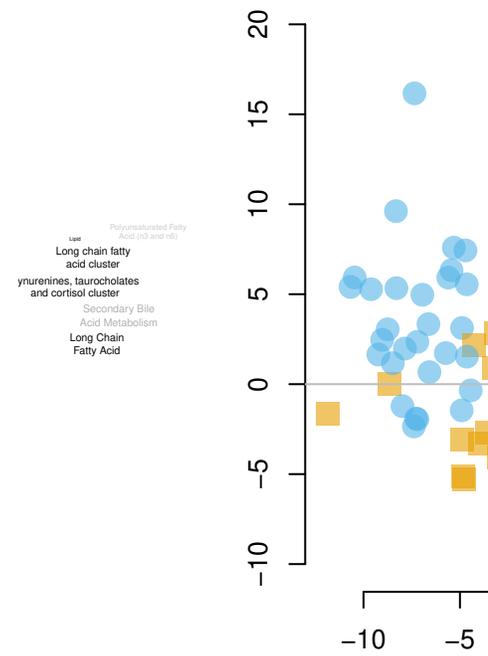
The `ret` object now contains the results of enrichments (in the `ret$enrichments member`). For each principal component, there is one element (e.g. `ret$enrichments$PC.1`) and we can directly throw it on a panel plot:

```
tmodPanelPlot(ret$enrichments$PC.1)
```

P value:

| 0.01 | 0.001 | 10^{-4} | 10^{-5} | 10^{-6} |

Effect size:

0.5   1

OK, but which of the terms are characteristic for TB patients? Which for the healthy controls? In the above, the enrichments were based on a list sorted by the absolute PCA weights. However, we can split it into a list ordered by signed weights ordered once from small to large values, and once from large to small values.

```r
pca <- prcomp(tbmprof, scale.=T)
ret <- tmodPCA(pca, genes=colnames(tbmprof), mset=modmetabo,
  plot.params=list(group=tb, legend="topright"),
  mode="cross")
```
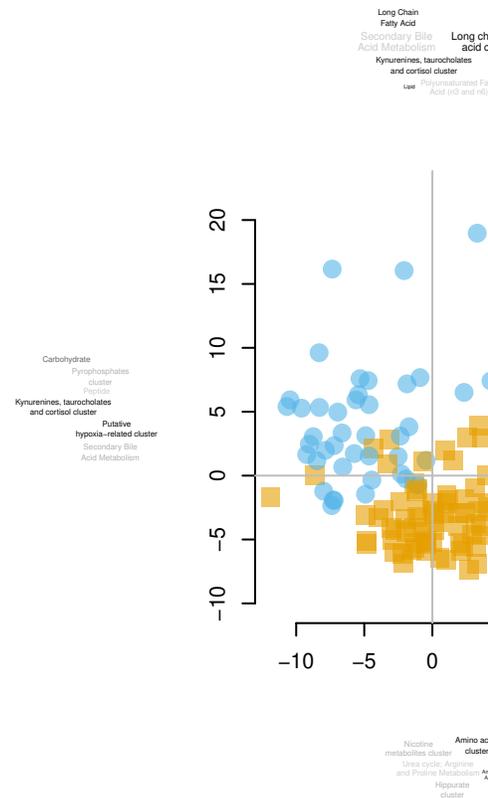
In essence, reading this plot is simple. First, note that this time the tag clouds on the top and the bottom correspond to the two ends of the vertical, y axis (second component); and the tag clouds at the left and right correspond to the two ends of the horizontal, x axis (first PCA component).

Now, take the amino acid cluster (bottom of the plot): it is enriched at the lower end of the y axis, which means, that features in that cluster are higher in the yellow points which are at the bottom of the plot (lower end of the y). In other words, amino acids are higher in healthy persons – a finding which corroborates the differential analysis above.

Similarly, "kynurenines" are at the left, lower side of the x axis, which means, that features from this cluster are at higher levels in TB patients.

What about the male-female differences? They probably can be found in other, less important[8] components. We could look for them manually, but we can also search which of the responses (turned to orthogonal PCA components) is best predicted by the `sex` factor.

```
foo <- summary(lm(pca$x ~ sex))
foo <- t(sapply(foo,
  function(x) c(r=x$r.squared, pval=x$coefficients[2,4])))
head(foo[ order(foo[,2]), ])
```

```
##                        r         pval
## Response PC5   0.24569651 8.485092e-10
## Response PC10  0.21456193 1.359253e-08
```

---

[8]That is, components which include a smaller fraction of the total variance in the data set

```
## Response PC7   0.03280924 3.482826e-02
## Response PC8   0.02213252 8.388780e-02
## Response PC107 0.01986978 1.016566e-01
## Response PC6   0.01919507 1.077110e-01
```

We can use the components 1 (which corresponds to TB/healthy) and components 5, which corresponds to male/female differences, as suggested by the above calculations.

```
ret <- tmodPCA(pca, genes=colnames(tbmprof), mset=modmetabo,
  plot.params=list(group=paste(sex, tb), legend="topright"),
  components=c(2,5))
```
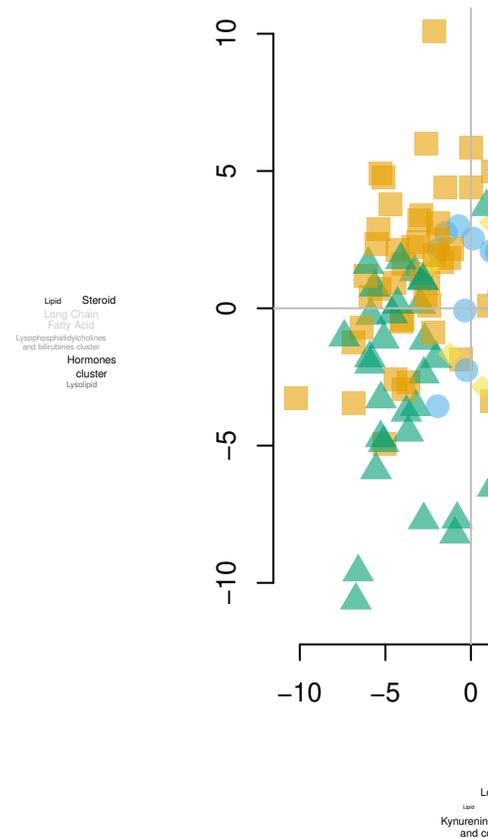

Orange circles and blue triangles are females, located mostly in Q1 and Q2 (top half); this corresponds to differences on the y axis and the tagcloud next to it (hormone cluster, steroids etc.). On the other hand, TB patients (blue triangles and yellow circles) are in Q1 and Q4 (right-hand side), which corresponds to the TB-specific tag cloud below the y axis.

Appendix: Mathematics behind CERNO test

The CERNO statistic is defined as

$$f_{CERNO} = -2 \cdot \sum_{i=1}^{N} \ln \frac{R_i}{N_{tot}}$$

It is straightforward to show that f_{CERNO} has a χ^2 distribution with $2 \cdot N$ degrees of freedom. First, note that a gene rank $\frac{R_i}{N_{tot}}$ is the probability that a random gene has equal or better rank and has uniform distribution.

$$\frac{R_i}{N_{tot}} \sim U(0,1)$$

Negative natural logarithm of a uniform distribution is an exponential distribution, so

$$-\ln \left(\frac{R_i}{N_{tot}} \right) \sim \text{Exp}(1)$$

and

$$-2 \cdot \ln \left(\frac{R_i}{N_{tot}} \right) \sim \text{Exp} \left(\frac{1}{2} \right) \sim \chi^2(2)$$

Sum of N independent variables each with a $\chi^2(2)$ distribution is a χ^2 distribution with $2 \cdot N$ degrees of freedom, and thus

$$f_{CERNO} = -2 \cdot \sum_{i=1}^{N} \ln \left(\frac{R_i}{N_{tot}} \right) \sim \chi^2(2 \cdot N)$$

The main problem with this approach is the assumption that we are summing independent variables, an assumption which is obviously violated by the fact that gene expression tends to be correlated.

References

Banchereau, Romain, Alejandro Jordan-Villegas, Monica Ardura, Asuncion Mejias, Nicole Baldwin, Hui Xu, Elizabeth Saye, et al. 2012. "Host Immune Transcriptional Profiles Reflect the Variability in Clinical Disease Manifestations in Patients with Staphylococcus Aureus Infections." *PLoS One* 7 (4). Public Library of Science: e34390.

Chaussabel, Damien, Charles Quinn, Jing Shen, Pinakeen Patel, Casey Glaser, Nicole Baldwin, Dorothee Stichweh, et al. 2008. "A Modular Analysis Framework for Blood Genomics Studies: Application to Systemic Lupus Erythematosus." *Immunity* 29 (1). Elsevier: 150–64.

Damian, Doris, and Malka Gorfine. 2004. "Statistical Concerns About the Gsea Procedure." *Nature Genetics* 36 (7). Nature Publishing Group: 663–63.

Li, Shuzhao, Nadine Rouphael, Sai Duraisingham, Sandra Romero-Steiner, Scott Presnell, Carl Davis, Daniel S Schmidt, et al. 2014. "Molecular Signatures of Antibody Responses Derived from a Systems Biology Study of Five Human Vaccines." *Nature Immunology* 15 (2). Nature Publishing Group: 195–204.

Maertzdorf, Jeroen, Martin Ota, Dirk Repsilber, Hans J Mollenkopf, January Weiner, Philip C Hill, and Stefan HE Kaufmann. 2011. "Functional Correlations of Pathogenesis-Driven Gene Expression Signatures in Tuberculosis." *PloS One* 6 (10). Public Library of Science: e26938.

Smyth, Gordon K. 2005. "Limma: Linear Models for Microarray Data." In *Bioinformatics and Computational Biology Solutions Using R and Bioconductor*, edited by R. Gentleman, V. Carey, S. Dudoit, R. Irizarry, and

W. Huber, 397–420. New York: Springer.

Subramanian, Aravind, Pablo Tamayo, Vamsi K Mootha, Sayan Mukherjee, Benjamin L Ebert, Michael A Gillette, Amanda Paulovich, et al. 2005. "Gene Set Enrichment Analysis: A Knowledge-Based Approach for Interpreting Genome-Wide Expression Profiles." *Proceedings of the National Academy of Sciences of the United States of America* 102 (43). National Acad Sciences: 15545–50.

Weiner 3rd, January, Shreemanta K Parida, Jeroen Maertzdorf, Gillian F Black, Dirk Repsilber, Anna Telaar, Robert P Mohney, et al. 2012. "Biomarkers of Inflammation, Immunosuppression and Stress with Active Disease Are Revealed by Metabolomic Profiling of Tuberculosis Patients." *PloS One* 7 (7). Public Library of Science: e40221.

Weiner, January. 2013. *Pca3d: Three Dimensional Pca Plots.*

———. 2014. *Tagcloud: Tag Clouds.*

Wendt, Hans W. 1972. "Dealing with a Common Problem in Social Science: A Simplified Rank-Biserial Coefficient of Correlation Based on the U Statistic." *European Journal of Social Psychology* 2 (4). Wiley Online Library: 463–65.

Yamaguchi, Ken D, Daniel L Ruderman, Ed Croze, T Charis Wagner, Sharlene Velichko, Anthony T Reder, and Hugh Salamon. 2008. "IFN-β-Regulated Genes Show Abnormal Expression in Therapy-Naïve Relapsing–remitting Ms Mononuclear Cells: Gene Expression Analysis Employing All Reported Protein–protein Interactions." *Journal of Neuroimmunology* 195 (1). Elsevier: 116–20.