Using MLP

Nandini Raghavan, An De Bondt, Tobias Verbeke

November 24, 2026

Contents

1 Introduction **1**

2 Example Use **2**
2.1 Preliminaries . . . 2
2.2 Prepare P Values . . . 2
2.3 Prepare Gene Sets . . . 2
2.4 Run MLP . . . 4
2.5 Visualize MLP Results . . . 6
2.5.1 Quantile Curves . . . 6
2.5.2 Bar Plot . . . 7
2.5.3 Gene Ontology Graph . . . 8
2.6 Visualize Individual Genes in a Gene Set . . . 9

3 References **10**

1 Introduction

Profiling technologies like gene expression profiling made it possible to quantify and compare relative gene expression profiles across a series of conditions and this for thousands of genes at a time. In order to understand the biology behind the difference between e.g. treatment and control, one might look at the function of individual genes which are differentially expressed. Another approach is to test which biological processes are significantly affected. Genes can be grouped into gene sets e.g. based on the biological process they are involved in. Coordinated differential expression of a set of functionally related genes could be more relevant than differential expression of a few unrelated genes, scattered across multiple gene sets.
The idea of the MLP methodology is to test whether there are gene sets enriched in small p-values (MLP denotes mean minus log p-value). The method does not require a cut-off value for significance at gene level. Also a distinction between up- or downregulated genes is not needed. Both of these principles taken together makes it possible to find, in one go, affected gene sets consisting of e.g. 50% inhibitors and 50% inducers.
The MLP methodology involves the use of (a) a test statistic to quantify the extent of the differential expression and (b) a resampling scheme to judge whether the difference is possibly real or attributable to chance. This process can be repeated for all gene sets of interest. The starting point for the MLP methodology is a list of p-values, or any similar statistics, that can quantify the degree of differential expression for each gene measured. These can be generated by a variety of methods used for calculating p-values based on gene expression data, several of them have been incorporated in the limma package. Smyth et al., 2003 Smyth, 2005
In this vignette, we show an example to identify biological processes that are affected in the considered experiment. The Gene Ontology Consortium is dealing with the classification of genes based on 3 criteria:

- the biological process they are involved in
- the molecular function they have
- their cellular localisation

The analysis behind the results below is focussing on identifying affected gene sets based on biological processes.

2 Example Use

The example data is from an expression profiling experiment with 2 sample groups, comparing wild-type mice with animals of which 1 gene has been knocked out. Each of the groups consist of 6 mice. The expression array used is the Affymetrix' Mouse430_2. The gene expression measurements have been summarized using GC-RMA (Irizarry et al., 2003 and Wu et al., 2004) based on Entrez Gene probeset definitions Dai et al., 2005.

2.1 Preliminaries

```
> require(MLP)
> require(limma)
> library(AnnotationDbi)
```

Load the needed libraries and the preprocessed data.

```
>   pathExampleData <- system.file("exampleFiles", "expressionSetGcrma.rda", package = "MLP")
>   load(pathExampleData)
>   annotation(expressionSetGcrma) <- "mouse4302"
```

It is advisable to make use of the annotation packages of the BrainArray group[1] at the University of Michigan, but as these packages are unfortunately not officially part of BioConductor (and on the build servers used to build packages [and corresponding vignettes]), the code above allows for using the sub-optimal Affymetrix annotation.

2.2 Prepare P Values

Estimate the fold changes and standard errors by fitting a linear model for each gene.

```
>   ### calculation of the statistics values via limma
>   group <- as.numeric(factor(pData(expressionSetGcrma)$subGroup1, levels = c("WT", "KO")))-1
>   design <- model.matrix(~group)
>   fit <- lmFit(exprs(expressionSetGcrma), design)
>   fit2 <- eBayes(fit)
>   results <- limma:::topTable(fit2, coef = "group", adjust.method = "fdr", number = Inf)
>   pvalues <- results[,"P.Value"]
>   names(pvalues) <- rownames(results)
>   # since we moved towards using "_at", next step should be needed as well
>   names(pvalues) <- sub("_at", "", names(pvalues))
```

2.3 Prepare Gene Sets

Create an object with the overview of the groups of genes you would like to consider. This object is a list of class `geneSetMLP`, where the slot names correspond to the gene set identifier and the slot content is a character vector of Entrez Gene identifiers for those genes belonging to that gene set. This object can be created using the `getGeneSets` function. This function has 3 parameters:

- `species` = a string being 'Human', 'Mouse', 'Rat', 'Dog' or 'Rhesus'
- `geneSetSource` = a string or a data.frame (more info below)
- `entrezIdentifiers` = a character vector of Entrez Gene identifiers for which gene statistics are available

The `geneSetSource` can be a string, i.e. `'GOBP'`, `'GOMF'`, `'GOCC'`, `'KEGG'` or `'REACTOME'`. The downstream analysis in these cases will identify gene sets, publicly available, as defined by the Gene Ontology Consortium for the first 3 , the Kyoto Encyclopedia of Genes and Genomes and REACTOME. For the latter, `species` must be one of 'Human', 'Mouse', 'Rat' or 'Dog'. The `geneSetSource` can also be a data.frame with at least the following 4 columns:

- `PATHWAYID` = identifier of the gene set
- `PATHWAYNAME` = description of the gene set
- `TAXID` = taxonomy identifier (9606, 10090, 10116 or 9615 for respectively Human, Mouse, Rat or Dog)

[1] See here.

- `GENEID` = Entrez Gene identifier belonging to that gene set

As an example, submitting the GO biological processes information as a data.drame would look like the data.frame below. In such a data.frame, the details of each gene set (identifier and description), is repeated as many times as the number of genes in that gene set and this for each species enclosed in the database.

```
    PATHWAYID TAXID                      PATHWAYNAME GENEID
1  GO:0000002 10090 mitochondrial genome maintenance  18975
2  GO:0000002 10090 mitochondrial genome maintenance  19819
3  GO:0000002 10090 mitochondrial genome maintenance  27393
4  GO:0000002 10090 mitochondrial genome maintenance  27395
5  GO:0000002 10090 mitochondrial genome maintenance  27397
6  GO:0000002 10090 mitochondrial genome maintenance  57813
7  GO:0000002 10090 mitochondrial genome maintenance  83945
8  GO:0000002 10090 mitochondrial genome maintenance 226153
9  GO:0000002 10090 mitochondrial genome maintenance 382985
10 GO:0000002 10116 mitochondrial genome maintenance  83474
11 GO:0000002 10116 mitochondrial genome maintenance 291824
12 GO:0000002 10116 mitochondrial genome maintenance 298933
13 GO:0000002 10116 mitochondrial genome maintenance 309441
14 GO:0000002 10116 mitochondrial genome maintenance 309762
15 GO:0000002 10116 mitochondrial genome maintenance 360481
```

```
> geneSet <- getGeneSets(species = "Mouse",
+                  geneSetSource = "GOCC",
+                  entrezIdentifiers = names(pvalues)
+ )
> tail(geneSet, 3)

$`GO:0009841`
[1] "270166"

$`GO:1990325`
[1] "373864"

$`GO:0150017`
[1] "380684"
```

As mentioned above, the returned object is a list of class `geneSetMLP`. This object has attributes which are used for the downstream analysis.

```
> str(attributes(geneSet))

List of 5
 $ names        : chr [1:2079] "GO:0005575" "GO:0005576" "GO:0030312" "GO:0031012" ...
 $ species      : chr "Mouse"
 $ geneSetSource: chr "GOCC"
 $ descriptions : Named chr [1:2079] "cellular_component" "extracellular region" "external encapsulatin
  ..- attr(*, "names")= chr [1:2079] "GO:0005575" "GO:0005576" "GO:0030312" "GO:0031012" ...
 $ class        : chr [1:2] "geneSetMLP" "list"
```

2.4 Run MLP

Run the actual MLP. To retrieve exact reproducible results, the seed is set in advance. The MLP function has 10 parameters, many of them can remain on their default values:

- `geneSet` = object of class `geneSetMLP` created by `getGeneSets`
- `geneStatistic` = named numeric vector corresponding to a gene-specific statistic, such as a p-value. The names are the corresponding Entrez Gene identifiers. This vector has the same length as the character vector submitted as `entrezIdentifiers` to the getGeneSets function.
- `minGenes` = minimal number of genes for a gene set to be considered in the analysis, default 5
- `maxGenes` = maximal number of genes for a gene set to be considered in the analysis, default 100
- `rowPermutations` = logical indicating whether critical values for the `geneSet` are computed using a permutation of the `geneStatistics`, default TRUE. The alternative is column permutations which requires a matrix of p-values corresponding to permutations of the original samples to be input. This latter option has not been fully implemented.
- `nPermutations` = number of permutations to be used for calculating the critical value for a certain `geneSet`.
- `smoothPValues` = logical indicating whether smoothing is desirable or not, default TRUE
- `probabilityVector` = vector of probabilities at which critical value curves for the `geneSet` are to be calculated. Default is c(0.5, 0.9, 0.95, 0.99, 0.999, 0.9999, 0.99999) corresponding to p-values of respectively 0.5, 0.1, 0.01, 0.001 etc.
- `df` = degrees of freedom for the smoothing parameter used in `smoothPValues`. The higher, the more smooth, default 9.
- `addGeneSetDescription` = logical indicating whether adding gene sets annotation to the MLP output is desirable or not, default TRUE.

```
> set.seed(111)
> mlpOut <- MLP(
+                 geneSet = geneSet,
+                 geneStatistic = pvalues,
+                 minGenes = 5,
+                 maxGenes = 100,
+                 rowPermutations = TRUE,
+                 nPermutations = 50,
+                 smoothPValues = TRUE,
+     probabilityVector = c(0.5, 0.9, 0.95, 0.99, 0.999, 0.9999, 0.99999),
+                 df = 9)
> head(mlpOut)
```

```
           totalGeneSetSize testedGeneSetSize geneSetStatistic geneSetPValue
GO:0140095                5                 5        1.6953984   0.001052959
GO:0106139               23                17        0.9155116   0.007899240
GO:1990917                9                 8        1.0943857   0.008898807
GO:0005602                7                 5        1.1808740   0.014516891
GO:0042613               12                10        0.9461005   0.016062568
GO:0042611               29                15        0.8520148   0.017927235
                      geneSetDescription
GO:0140095             cytoplasmic lattice
GO:0106139           symbiont cell surface
GO:1990917                        ooplasm
GO:0005602 complement component C1 complex
GO:0042613    MHC class II protein complex
GO:0042611             MHC protein complex
```

Some properties of the MLP procedure , as well as some parts of the implemented procedure, assume that the `geneStatistic` has a uniform distribution between 0 and 1 under the null hypothesis for a given analysis. The returned object is a data.frame of class `MLP` with at least 4 columns:

- `totalGeneSetSize` = total number of genes in the corresponding gene set
- `testedGeneSetSize` = number of genes in the corresponding `geneSet` for which a gene statistic has been submitted
- `geneSetStatistic` = mean of the -log10 of the genes tested in that `geneSet`
- `geneSetPValue` = p-value associated with the geneSetStatistic

This object has attributes which are used for visualising the analysis results.

```
> str(attributes(mlpOut))
```

```
List of 6
 $ names                   : chr [1:5] "totalGeneSetSize" "testedGeneSetSize" "geneSetStatistic" "geneS
 $ class                   : chr [1:2] "MLP" "data.frame"
 $ row.names               : chr [1:945] "GO:0140095" "GO:0106139" "GO:1990917" "GO:0005602" ...
 $ species                 : chr "Mouse"
 $ geneSetSource           : chr "GOCC"
 $ quantileCurveInformation:List of 5
  ..$ x0 : Named num [1:945] 9.54 3.46 2.45 6.78 3.16 ...
  .. ..- attr(*, "names")= chr [1:945] "GO:0001750" "GO:0097381" "GO:0120202" "GO:0098533" ...
  ..$ y0 : Named num [1:945] 0.407 0.583 0.541 0.335 0.224 ...
  .. ..- attr(*, "names")= chr [1:945] "GO:0001750" "GO:0097381" "GO:0120202" "GO:0098533" ...
  ..$ xtp: num [1:945, 1:7] 0.441 0.432 0.422 0.44 0.431 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:945] "GO:0001750" "GO:0097381" "GO:0120202" "GO:0098533" ...
  .. .. ..$ : chr [1:7] "Curve0.5" "Curve0.9" "Curve0.95" "Curve0.99" ...
  ..$ qi : num [1:7] 0.5 0.9 0.95 0.99 0.999 ...
  ..$ lqi: NULL
```

2.5 Visualize MLP Results

Three different types of plots are made available. The type of the plot is indicated with the `type` argument which can be one of

```
plot(., type = "quantileCurves")
plot(., type = "barplot")
plot(., type = "GOgraph")
```

2.5.1 Quantile Curves

This visualisation shows the relationship between the `geneSetStatistic` and the size of the gene sets. It also indicates the quantiles of interest as specified as `probabilityVector` in the `MLP` function. The most significant gene sets are plotted above the smooth curve.

```
> pdf("mlpQuantileCurves.pdf", width = 10, height = 10)
> plot(mlpOut, type = "quantileCurves")
> tmp <- dev.off()
```

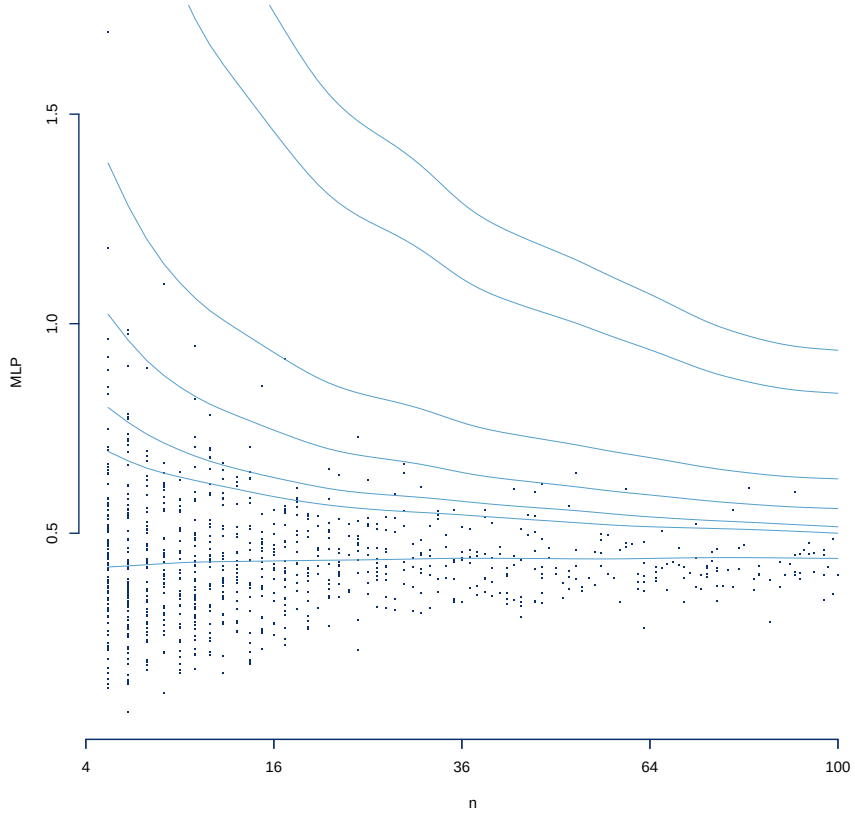

Figure 1: Example of a quantile curve for the MLP results. Every dot represents a gene set. Every line represents a smoothing of the quantile per gene set size.

2.5.2 Bar Plot

For this type of plot there are some extra parameters of interest:

- `nRow` = number of gene sets to include in the graph, default is 20
- `barColors` = vector of colors, default is a shade of grey per bar. The 3 possible shadings correspond to the % of genes in a gene set tested as compared to the total number of genes, the darker, the bigger the portion of genes tested.
- `ylab` =label for the y-axis

```
>   pdf("mlpBarplot.pdf", width = 10, height = 10)
>   op <- par(mar = c(30, 10, 6, 2))
>   plot(mlpOut, type = "barplot", ylab = "-log10(gene set p-value)")
>   par(op)
>   tmp <- dev.off()
```

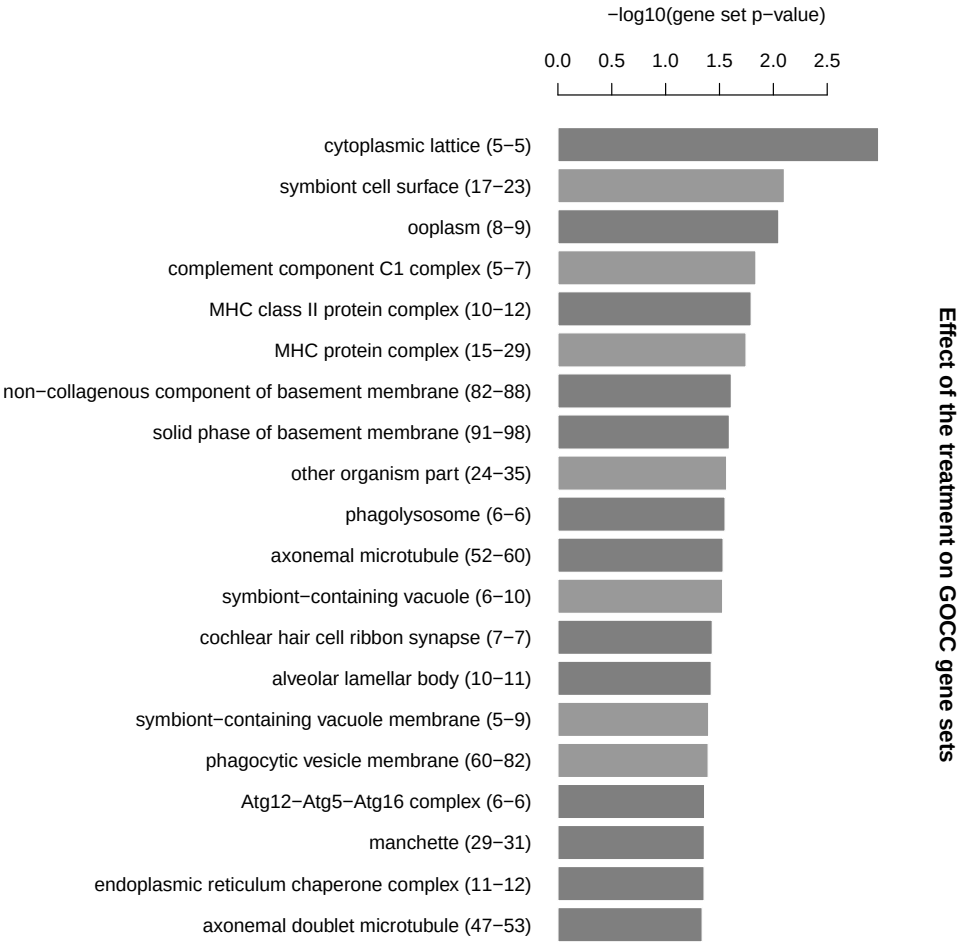

Figure 2: Example of a barplot for the MLP results. The height of a bar represents the significance (-log10(`geneSetPValue`)) of the gene set indicated on the x-axis. The number between brackets represent the number of genes within that gene set (number of genes for which a gene statistic has been submitted as well as the total number of genes).

2.5.3 Gene Ontology Graph

As the title indicates, this type of plot is only possible if the `geneSetSource` was 'GOBP', 'GOMF' or 'GOCC'. Also for this type of plot there is some extra parameters:

- `nRow` = number of gene sets as basis to create the graph, default is low, i.e. 5. The higher this number, the more populated the graph gets.
- `nCutDescPath` = number of characters at which the pathway description should be cut (depends on figure size)

```
>   pdf("mlpGOgraph.pdf", width = 8, height = 6)
>   op <- par(mar = c(0, 0, 0, 0))
>   plot(mlpOut, type = "GOgraph", nRow = 10, nCutDescPath = 15)
>   par(op)
>   tmp <- dev.off()
```

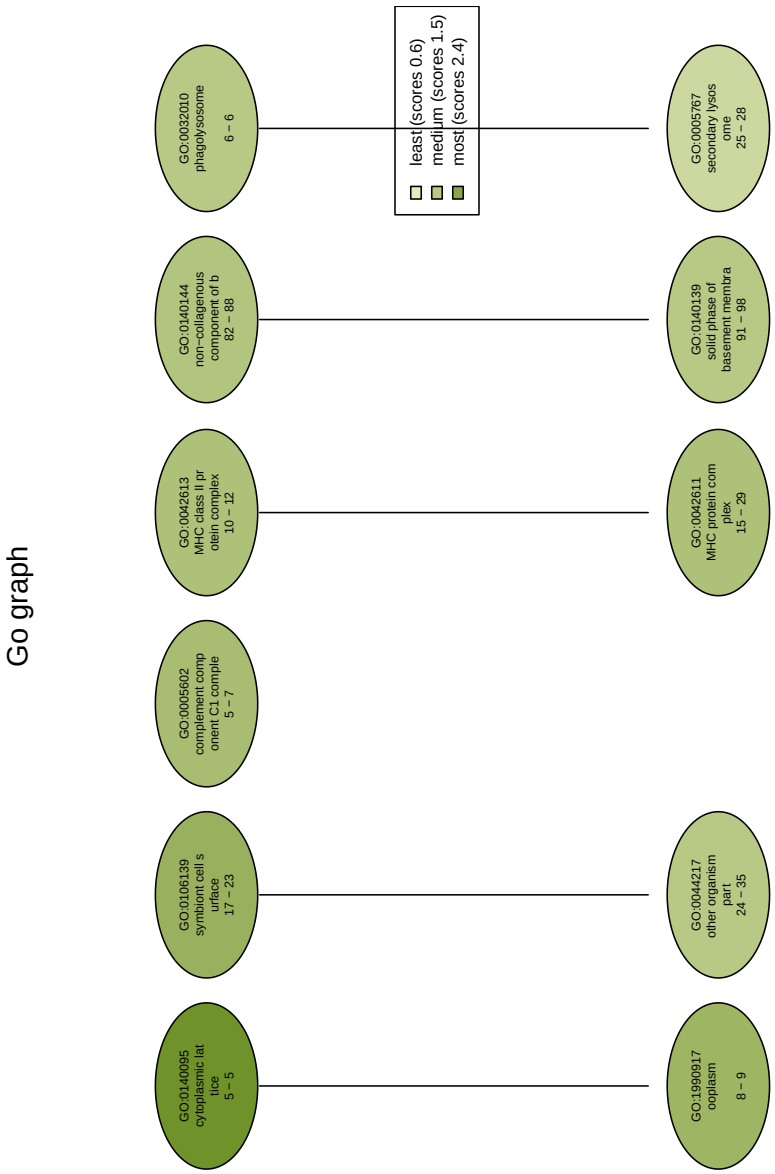

Figure 3: Example of a GOgraph for the MLP results. Every elipse represents a gene set. The color indicates the significance, the more green, the more significant. The connectors indicate the parent - child relationship. The number between brackets represent the number of genes within that gene set (number of genes for which a gene statistic has been submitted as well as the total number of genes)

2.6 Visualize Individual Genes in a Gene Set

To know which genes contribute most to the significance of a gene set or to focus on a certain gene set of interest, you can plot the significance of each gene beloning to that gene set. This plot shows the significance (-log10(`geneStatistic`)) of the genes within the gene set of interest. The `plotGeneSetSignificance` function needs 4 parameters and there is also one optional parameter:

- `geneSet` = object of class `geneSetMLP` created by getGeneSets
- `geneSetIdentifier` = identifier of the gene set of interest
- `geneStatistic` = named numeric vector which should have a uniform distribution between 0 and 1. The names are the corresponding Entrez Gene identifiers.
- `annotationPackage` = string representing the annotation package used to retrieve gene symbols and gene descriptions
- `barColors` = optional color vector

```
> geneSetID <- rownames(mlpOut)[1]
> pdf("geneSignificance.pdf", width = 10, height = 10)
> op <- par(mar = c(25, 10, 6, 2))
> plotGeneSetSignificance(
+                 geneSet = geneSet,
+                 geneSetIdentifier = geneSetID,
+                 geneStatistic = pvalues,
+                 annotationPackage = annotation(expressionSetGcrma)
+ )
> par(op)
> tmp <- dev.off()
```

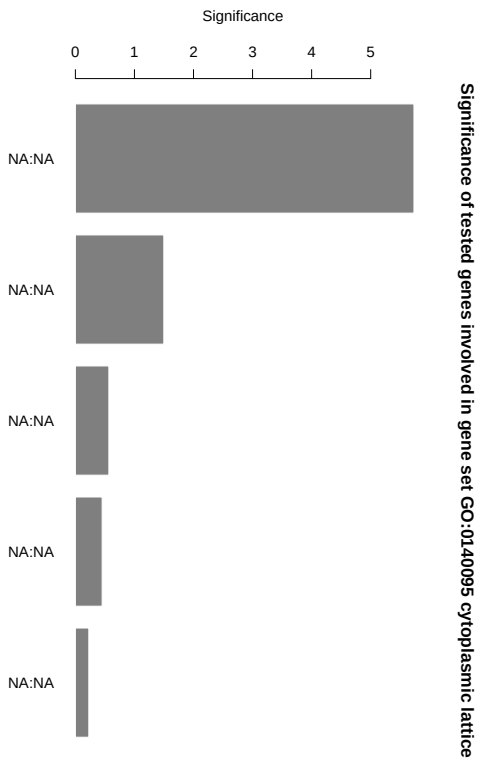

Figure 4: Example of a gene significance plot for a gene set of interest. The height of a bar represents the significance (-log10(`geneStatistic`)) of the gene indicated on the x-axis.

3 References

Raghavan N, De Bondt AM, Talloen W, Moechars D, Göhlmann HW, Amaratunga D. The high-level similarity of some disparate gene expression measures. *Bioinformatics.* 2007 Nov 15;23(22):3032-8. Epub 2007 Sep 24.PMID: 17893087

Raghavan N, Amaratunga D, Cabrera J, Nie A, Qin J, McMillian M. On methods for gene function scoring as a means of facilitating the interpretation of microarray results. *J Comput Biol.* 2006 Apr; 13(3):798-809.PMID: 16706726