The RTopper package: perform run Gene Set Enrichment across genomic platforms

Luigi Marchionni
Department of Oncology
Johns Hopkins University
email: marchion@jhu.edu

May 3, 2016

Contents

1 Overview

Gene Set Enrichment (GSE) analysis has been widely use to assist the interpretation of gene expression data. We propose here to apply GSE for the integration of genomic data obtained from distinct analytical platform.

In the present implementation of the `RTopper` GSE analysis is performed using the `geneSetTest` function from the `limma` package **???**. This function enables testing the hypothesis that a specific set of genes (a Functional Gene Set, FGS) is more highly ranked on a given statistics. In particular this functions computes a p-value for each FGS by one or two-sided Wilcoxon rank-sum test. Alternative user-defined functions can also be used.

Furthermore multiple hypothesis testing correction is achieved by applying the Benjamini and Hochberg method **?** as implemented in the `multtest` R/Bioconductor package. Overall, this approach is conceptually analogous to Gene Set Enrichment Analysis (GSEA), as proposed by Mootha and colleagues **??**.

The integration can be achieved through two distinct approaches:

1. **GSE + INTEGRATION**: Separate GSE analysisn on the individual genomic platforms followed by GSE results integration;
2. **INTEGRATION + GSE**: Integration of genomic data measurement using a logistic model followed by GSE analysis;

2 RTopper data structure

In this tutorial we demonstrate the functionality of `RTopper` package. To this end we will make use of simplified data generated within The Cancer Genome Atlas (TCGA) project, using Glioblastoma Multiforme (GBM) genomics data obtained from the same patients' cohort using distinct platforms,

including Differential Gene Expression (DGE), Copy Number Variation (CNV), and Differential Methylation (DM). This data is included with the `RTopper` package as the dataset `exampleData`, which consists of genomic measurements (the list `dat`) for 500 genes (in rows) and 95 patients (in columns) from 4 distinct platforms:

1. DGE obtained using Affymetrix;
2. DGE obtained using Agilent;
3. CNV data generated ad Harvard;
4. CNV data generated ad the MSKCC;

The phenotypic class for each patient is defined in the a data.frame `pheno` consisting of 95 rows (patients, *pheno$Sample*) and 2 columns, the first being patients identifiers, and the second variable giving the group indicator (*pheno$Class*).

To load the data set type `data(exampleData)`, and to view a description of this data type `?exampleData`. The structure of the data is shown below:

```
> library(RTopper)
> data(exampleData)
> ls()

[1] "dat"   "pheno"

> class(dat)

[1] "list"

> names(dat)

[1] "dat.affy"       "dat.agilent"
[3] "dat.cnvHarvard" "dat.cnvMskcc"

> sapply(dat,class)

      dat.affy    dat.agilent dat.cnvHarvard
  "data.frame"   "data.frame"   "data.frame"
  dat.cnvMskcc
  "data.frame"

> sapply(dat,dim)

     dat.affy dat.agilent dat.cnvHarvard
[1,]      500         500            500
[2,]       95          95             95
     dat.cnvMskcc
[1,]          500
[2,]           95

> dim(pheno)

[1] 95  2

> str(pheno)
```

```
'data.frame':        95 obs. of  2 variables:
 $ Sample: chr  "TCGA.02.0003" "TCGA.02.0007" "TCGA.02.0011" "TCGA.02.0021" ...
 $ Class : int  0 0 1 1 0 0 0 0 0 0 ...
```

In summary to perform the analysis with functions from `RTopper` the genomic data used as input must be in the following format:

1. **Genomic measurements**: a list of data.frames, in which each list item corresponds to a genomic platform, and comprises a data.frame with rows being genes and columns patients;
2. **Phenotype data**: a data.frame with 2 columns: patients and their phenotypes;
3. The number of columns of the *Genomic measurements* data.frames must match the number of rows of the *Phenotype data*;
4. The same set of genes must be measured in each platform and gene labels must be stored as rownames;

Below are shown the first 6 rows and 4 columns of each data.frame contained in `dat`, which share the same genes (shown for some of the possible combinations). Similarly column names in the `dat` data.frames correspond to rownames of `pheno`.

```
> ###data structure
> lapply(dat,function(x) head(x)[,1:3])

$dat.affy
      TCGA.02.0003 TCGA.02.0007 TCGA.02.0011
AACS      7.747995     7.685409     7.535661
AARS      9.381544     9.930156    10.197194
ABI1      8.173255     8.962803     9.895811
ACHE      5.127197     4.547297     5.146552
ACTC1     6.612645     5.825879     8.067945
ACTN2     6.257383     5.330557     5.842319

$dat.agilent
      TCGA.02.0003 TCGA.02.0007 TCGA.02.0011
AACS    -1.0070000   -1.1164000    -0.913000
AARS    -1.2665000   -0.8981250     0.263500
ABI1    -0.2765000    0.3356250     1.027250
ACHE     0.4403750   -0.0222500     0.115000
ACTC1    0.3641538    0.1234615     1.046692
ACTN2    4.3348000    2.2278000     3.330600

$dat.cnvHarvard
      TCGA.02.0003 TCGA.02.0007 TCGA.02.0011
AACS   -0.08273213  -0.08917331  -0.02075644
AARS   -0.10233281  -0.20620608  -0.05157664
ABI1   -0.86886659  -0.01214599   0.59307754
ACHE    0.31560002  -1.00166150  -0.14519639
ACTC1  -1.17495078  -0.26698279  -0.95662761
ACTN2  -0.11319016  -0.09657971   0.02582138
```

```
$dat.cnvMskcc
      TCGA.02.0003 TCGA.02.0007 TCGA.02.0011
AACS    -0.0383875  -0.09140000  0.008233333
AARS     0.0075600   0.02801667  0.104850000
ABI1    -0.7006900   0.21270000  0.499472727
ACHE     0.8676000  -0.23970000  0.075000000
ACTC1   -0.9779500  -0.11625000 -0.692950000
ACTN2   -0.1258571  -0.05394444  0.010200000
```

```
> sum(rownames(dat[[1]])%in%rownames(dat[[2]]))
```

```
[1] 500
```

```
> sum(rownames(dat[[2]])%in%rownames(dat[[3]]))
```

```
[1] 500
```

2.1 Creation of Functional Gene Sets

Functional Gene Sets (FGS) are list of genes that share a specific biological function. Examples of FGS are genes that operate in the same signaling pathway (*i.e.* Notch signaling genes), or that share the same biological function (*i.e.* Cell adhesion genes). FGS can be retrieved from various database, or can be construncted *ad hoc*. A convenient source of FGS are the R-Bioconductor metaData packages, and S4 classes and methods for handling FGS are provided by the `GSEABase` package. Below is shown a simple way to extract FGS from the human genome metaData package `org.Hs.eg.db`. As a general rule the name of the metaData package, without the `.db` extension, can be used a function to see the content of the package, as shown below:

```
> library(org.Hs.eg.db)
> org.Hs.eg()
```

```
Quality control information for org.Hs.eg:


This package has the following mappings:

org.Hs.egACCNUM has 42215 mapped keys (of 59887 keys)
org.Hs.egACCNUM2EG has 797608 mapped keys (of 797608 keys)
org.Hs.egALIAS2EG has 119263 mapped keys (of 119263 keys)
org.Hs.egCHR has 59389 mapped keys (of 59887 keys)
org.Hs.egCHRLENGTHS has 93 mapped keys (of 93 keys)
org.Hs.egCHRLOC has 26396 mapped keys (of 59887 keys)
org.Hs.egCHRLOCEND has 26396 mapped keys (of 59887 keys)
org.Hs.egENSEMBL has 25399 mapped keys (of 59887 keys)
org.Hs.egENSEMBL2EG has 27937 mapped keys (of 27937 keys)
org.Hs.egENSEMBLPROT has 7482 mapped keys (of 59887 keys)
org.Hs.egENSEMBLPROT2EG has 23556 mapped keys (of 23556 keys)
org.Hs.egENSEMBLTRANS has 7943 mapped keys (of 59887 keys)
org.Hs.egENSEMBLTRANS2EG has 36025 mapped keys (of 36025 keys)
org.Hs.egENZYME has 2230 mapped keys (of 59887 keys)
```

```
org.Hs.egENZYME2EG has 975 mapped keys (of 975 keys)
org.Hs.egGENENAME has 59887 mapped keys (of 59887 keys)
org.Hs.egGO has 18750 mapped keys (of 59887 keys)
org.Hs.egGO2ALLEGS has 20856 mapped keys (of 20856 keys)
org.Hs.egGO2EG has 16339 mapped keys (of 16339 keys)
org.Hs.egMAP has 39638 mapped keys (of 59887 keys)
org.Hs.egMAP2EG has 2300 mapped keys (of 2300 keys)
org.Hs.egOMIM has 15230 mapped keys (of 59887 keys)
org.Hs.egOMIM2EG has 19643 mapped keys (of 19643 keys)
org.Hs.egPATH has 5869 mapped keys (of 59887 keys)
org.Hs.egPATH2EG has 229 mapped keys (of 229 keys)
org.Hs.egPMID has 33984 mapped keys (of 59887 keys)
org.Hs.egPMID2EG has 492261 mapped keys (of 492261 keys)
org.Hs.egREFSEQ has 40787 mapped keys (of 59887 keys)
org.Hs.egREFSEQ2EG has 274649 mapped keys (of 274649 keys)
org.Hs.egSYMBOL has 59887 mapped keys (of 59887 keys)
org.Hs.egSYMBOL2EG has 59868 mapped keys (of 59868 keys)
org.Hs.egUCSCKG has 23393 mapped keys (of 59887 keys)
org.Hs.egUNIGENE has 26386 mapped keys (of 59887 keys)
org.Hs.egUNIGENE2EG has 28875 mapped keys (of 28875 keys)
org.Hs.egUNIPROT has 19258 mapped keys (of 59887 keys)


Additional Information about this package:

DB schema: HUMAN_DB
DB schema version: 2.1
Organism: Homo sapiens
Date for NCBI data: 2016-Mar14
Date for GO data: 20160305
Date for KEGG data: 2011-Mar15
Date for Golden Path data: 2010-Mar22
Date for Ensembl data: 2016-Mar9
```

For instance the `org.Hs.egGO2ALLEGS` environment contains the mapping of all ENTREZ Gene identifiers to the **Gene Ontology Terms ?**, while `org.Hs.egPATH2EG` maps the identifiers to **KEGG** pathways **?**. The corresponding lists of FGS can be retrieve from the corresponding environments using the the R command `as.list()`, as shown below for KEGG and GO:

```
> kegg <- as.list(org.Hs.egPATH2EG)
> go <- as.list(org.Hs.egGO2ALLEGS)
> length(kegg)

[1] 229

> length(go)

[1] 20856

> str(kegg[1:5])
```

```
List of 5
 $ 04610: chr [1:69] "2" "462" "623" "624" ...
 $ 00232: chr [1:7] "9" "10" "1544" "1548" ...
 $ 00983: chr [1:52] "9" "10" "978" "1066" ...
 $ 01100: chr [1:1130] "9" "10" "15" "18" ...
 $ 00380: chr [1:42] "15" "26" "38" "39" ...
```

```
> names(kegg)[1:5]
```

```
[1] "04610" "00232" "00983" "01100" "00380"
```

```
> str(go[1:5])
```

```
List of 5
 $ GO:0000002: Named chr [1:32] "142" "291" "1763" "1890" ...
  ..- attr(*, "names")= chr [1:32] "IMP" "TAS" "IDA" "TAS" ...
 $ GO:0000003: Named chr [1:1596] "18" "49" "49" "49" ...
  ..- attr(*, "names")= chr [1:1596] "IEA" "IEA" "IMP" "ISS" ...
 $ GO:0000011: Named chr "64145"
  ..- attr(*, "names")= chr "IBA"
 $ GO:0000012: Named chr [1:11] "142" "3981" "7014" "7141" ...
  ..- attr(*, "names")= chr [1:11] "IEA" "IDA" "NAS" "IDA" ...
 $ GO:0000018: Named chr [1:64] "604" "641" "940" "958" ...
  ..- attr(*, "names")= chr [1:64] "IEA" "IMP" "IEA" "IEA" ...
```

```
> names(go)[1:5]
```

```
[1] "GO:0000002" "GO:0000003" "GO:0000011"
[4] "GO:0000012" "GO:0000018"
```

In the `kegg` list genes are identified by their ENTREZ Gene identifiers, while in the `dat` genes are identified by their Gene Symbol. Below is an example of the code that can be used to perform the identifiers conversion, using only a subset of KEGG and GO FGS:

```
> kegg <- lapply(kegg[sample(1:length(kegg),5)],function(x) unique(unlist(mget(x,org.Hs.egSYMBO
> go <- lapply(go[sample(1:length(go),5)],function(x) unique(unlist(mget(x,org.Hs.egSYMBOL))))
> str(kegg)
```

```
List of 5
 $ 03050: chr [1:44] "IFNG" "PSMA1" "PSMA2" "PSMA3" ...
 $ 00565: chr [1:36] "PAFAH1B1" "PAFAH1B2" "PAFAH1B3" "PAFAH2" ...
 $ 04510: chr [1:200] "ACTB" "ACTG1" "ACTN4" "ACTN1" ...
 $ 00310: chr [1:44] "ACAT1" "ACAT2" "ALDH2" "ALDH1B1" ...
 $ 00524: chr [1:5] "GCK" "HK1" "HK2" "HK3" ...
```

```
> str(go)
```

```
List of 5
 $ GO:0030804: chr [1:93] "ABCA1" "ACR" "ADCY7" "ADCYAP1" ...
 $ GO:0048009: chr [1:37] "AKT1" "BMP2" "BMP5" "CDH3" ...
 $ GO:0043237: chr [1:7] "DAG1" "NID1" "SHH" "THBS4" ...
 $ GO:0033842: chr [1:2] "B4GALNT3" "B4GALNT4"
 $ GO:0046933: chr [1:11] "ATP5A1" "ATP5B" "ATP5C1" "ATP5D" ...
```

Finally, it is also possible to annotate FGS, mapping pathways identifiers to pathway names, as shown below for KEGG, using the KEGG.db.

```
> library(KEGG.db)
> KEGG()
```

```
Quality control information for KEGG:


This package has the following mappings:

KEGGENZYMEID2GO has 4178 mapped keys (of 4178 keys)
KEGGEXTID2PATHID has 75100 mapped keys (of 75100 keys)
KEGGGO2ENZYMEID has 5224 mapped keys (of 5224 keys)
KEGGPATHID2EXTID has 3152 mapped keys (of 3152 keys)
KEGGPATHID2NAME has 478 mapped keys (of 478 keys)
KEGGPATHNAME2ID has 478 mapped keys (of 478 keys)


Additional Information about this package:

DB schema: KEGG_DB
DB schema version: 2.1
Date for KEGG data: 2011-Mar15
```

```
> names(kegg) <- paste(names(kegg),unlist(mget(names(kegg),KEGGPATHID2NAME)),sep=".")
> names(kegg)
```

```
[1] "03050.Proteasome"
[2] "00565.Ether lipid metabolism"
[3] "04510.Focal adhesion"
[4] "00310.Lysine degradation"
[5] "00524.Butirosin and neomycin biosynthesis"
```

Similarly GO Terms can be retrieved from the GO.db (please refer to the vignettes of the corresponding packages for details).

```
> library(GO.db)
> GO()
```

```
Quality control information for GO:


This package has the following mappings:

GOBPANCESTOR has 28477 mapped keys (of 28477 keys)
GOBPCHILDREN has 16829 mapped keys (of 28477 keys)
GOBPOFFSPRING has 16829 mapped keys (of 28477 keys)
GOBPPARENTS has 28477 mapped keys (of 28477 keys)
GOCCANCESTOR has 3897 mapped keys (of 3897 keys)
GOCCCHILDREN has 1317 mapped keys (of 3897 keys)
```

```
GOCCOFFSPRING has 1317 mapped keys (of 3897 keys)
GOCCPARENTS has 3897 mapped keys (of 3897 keys)
GOMFANCESTOR has 10021 mapped keys (of 10021 keys)
GOMFCHILDREN has 2046 mapped keys (of 10021 keys)
GOMFOFFSPRING has 2046 mapped keys (of 10021 keys)
GOMFPARENTS has 10021 mapped keys (of 10021 keys)
GOOBSOLETE has 2012 mapped keys (of 2012 keys)
GOTERM has 42396 mapped keys (of 42396 keys)


Additional Information about this package:

DB schema: GO_DB
DB schema version: 2.1
Date for GO data: 20160305
```

```
> names(go) <- paste(names(go),Term(names(go)),sep=".")
> names(go)
```

```
[1] "GO:0030804.positive regulation of cyclic nucleotide biosynthetic process"
[2] "GO:0048009.insulin-like growth factor receptor signaling pathway"
[3] "GO:0043237.laminin-1 binding"
[4] "GO:0033842.N-acetyl-beta-glucosaminyl-glycoprotein 4-beta-N-acetylgalactosaminyltransferas
[5] "GO:0046933.proton-transporting ATP synthase activity, rotational mechanism"
```

Finally we can be combine the two FGS collections into a named list for further used in GSE analysis (see below).

```
> fgsList <- list(go=go,kegg=kegg)
```

3 Data analysis with RTopper

To compute gene-to-phenotype association scores the first step required is the convertion of the data into a list, where each list item corresponds to a gene, and comprises a data.frame with the rows being patients, and columns being measurements for each data type, along with the class phenotype (*the response*). Importantly each element of the list with the data should have the same genes and patients.

The `convertToDr` function is used to make such conversion. Below is a short description of the arguments to this function:

- `dataIntersection`: a list of data.frames containing the same set of patients(columns) and genes (rows)
- `response`: a data.frame indicating patients' phenotypic class;
- `nPlatforms`: the number of platforms;

This can be achieved as follows using our examples data:

```
> dataDr <- convertToDr(dat, pheno, 4)
> class(dataDr)
```

```
[1] "list"
```

```
> length(dataDr)
```

```
[1] 500
```

```
> names(dataDr)[1:5]
```

```
[1] "AACS"  "AARS"  "ABI1"  "ACHE"  "ACTC1"
```

```
> str(dataDr[1:2])
```

```
List of 2
 $ AACS:'data.frame':        95 obs. of  5 variables:
  ..$ dat.affy      : num [1:95] 7.75 7.69 7.54 7.3 7.01 ...
  ..$ dat.agilent   : num [1:95] -1.007 -1.116 -0.913 -1.061 -1.775 ...
  ..$ dat.cnvHarvard: num [1:95] -0.0827 -0.0892 -0.0208 -0.1811 -0.0625 ...
  ..$ dat.cnvMskcc  : num [1:95] -0.03839 -0.0914 0.00823 0.03456 0.0573 ...
  ..$ response      : int [1:95] 0 0 1 1 0 0 0 0 0 0 ...
 $ AARS:'data.frame':        95 obs. of  5 variables:
  ..$ dat.affy      : num [1:95] 9.38 9.93 10.2 9.54 9.37 ...
  ..$ dat.agilent   : num [1:95] -1.266 -0.898 0.264 -0.599 -1.437 ...
  ..$ dat.cnvHarvard: num [1:95] -0.1023 -0.2062 -0.0516 -0.0923 -0.1199 ...
  ..$ dat.cnvMskcc  : num [1:95] 0.00756 0.02802 0.10485 0.0841 0.12262 ...
  ..$ response      : int [1:95] 0 0 1 1 0 0 0 0 0 0 ...
```

It is now possible to compute gene-to-phenotype association scores, using as input the gene-centered list produced by `convertToDr`. Therefore the `computeDrStat` function assumes that each gene-centered data.frame contains a column (the last one) called `'response'`, as created by the `convertToDr`. Below is a short description of the arguments to this function:

- `data`: a list of data.frames, one for each gene analyzed, contining the the genomic measurements from all platforms (by column) for all the patients (by row), along with the phenotypic response;
- `columns`: a numeric vector indicating column indexes corresponding the genomic measurements to be used for computing the gene-to-phenotype association scores; the default is `columns = c(1:(ncol(data) - 1))`, assuming the phenotypic response to be the last column;
- `method`: the method used to compute the association score;
- `integrate`: logical, whether an integrated gene-to-phenotype score should be computed, or separate scores for each platform/data sets specified by `columns`;

In the current implementation of the `RTopper` there are three methods for computing gene-to-phenotype association scores:

1. `dev`: this approach computes the score as the difference of deviances (as described in Tyekucheva et al, manuscript under review **?**);
2. `aic`: this approach computes the score as the Akaike information criterion for model selection;
3. `bic`: this approach computes the score as the penalized likelihood ratio;

3.1 Integrated Gene-to-Phenotype score computation

This approach first integrates genomic data across platform, and subsequently perform GSE to identify the FGS most strongly associated with the integrated score. Below is an example of application to compute the gene-to-phenotype association scores for 4 data type simultaneously:

```
> bicStatInt <- computeDrStat(dataDr, columns = c(1:4), method="bic", integrate = TRUE)
> names(bicStatInt)

[1] "integrated"

> str(bicStatInt)

List of 1
 $ integrated: Named num [1:500] -11.43 -15.93 -8.85 -13.52 -7.26 ...
  ..- attr(*, "names")= chr [1:500] "AACS" "AARS" "ABI1" "ACHE" ...
```

3.2 Separate Gene-to-Phenotype score computation

This approach first computes computes gene-to-phenotype score separately for each platform, uses the scores to perform separate GSE analysis in each platform for identifying the FGS most strongly associated with the score, and finally integrates the results from GSE analysis, Below is an example of this approach:

```
> bicStatSep <- computeDrStat(dataDr, columns = c(1:4), method="bic", integrate = FALSE)
> names(bicStatSep)

[1] "dat.affy"       "dat.agilent"
[3] "dat.cnvHarvard" "dat.cnvMskcc"

> str(bicStatSep)

List of 4
 $ dat.affy      : Named num [1:500] 0.545 -4.269 -2.334 -4.471 -3.625 ...
  ..- attr(*, "names")= chr [1:500] "AACS" "AARS" "ABI1" "ACHE" ...
 $ dat.agilent   : Named num [1:500] -3.57 -4.5 -3.66 -4.52 -1.05 ...
  ..- attr(*, "names")= chr [1:500] "AACS" "AARS" "ABI1" "ACHE" ...
 $ dat.cnvHarvard: Named num [1:500] -4.49 -3.64 3.13 -3.26 -2.57 ...
  ..- attr(*, "names")= chr [1:500] "AACS" "AARS" "ABI1" "ACHE" ...
 $ dat.cnvMskcc  : Named num [1:500] -4.53 -4.48 2.1 -2.55 -4.25 ...
  ..- attr(*, "names")= chr [1:500] "AACS" "AARS" "ABI1" "ACHE" ...
```

3.3 Gene Set Enrichment using integrated and separate score

After the gene-to-phenotype scores have been obtained it is possible to perform a GSE analysis. To this end we will use the `runBatchGSE` function, as shown below. This function enables to perform GSE analysis over multiple collections of FGS, and over multiple ranking statistics. In the current implementation of the `runBatchGSE` the default is performing the enrichment analysis using the `geneSetTest` function from the `limma` package, and most of the arguments passed to `runBatchGSE` are indeed passed to `geneSetTest` (see the relative help for the details).

As an alternative the user can also define his own function to test for FGS enrichment, passing the selection of genes within the FGS and the ranking ranking statistics in the same way as done for `geneSetTest`. In this tutorial we apply `geneSetTest` in order to perform a Wilcoxon rank-sum test, using the absolute value of the gene-to-phenotype scores as the ranking statistics.

```
> args(runBatchGSE)
```

```
function (dataList, fgsList, ...)
NULL
```

Below a short description of the arguments that can be passed to this function:

- `dataList`: a list containing gene-to-phenotype scores to be used as ranking statistics in the GSE analysis;
- `fgsList`: a list of FGS collection, in which each element is a list of character vectors, one for each gene set;
- `...`: any other argument to be passed to lower level functions, including the lower level enrichment function to be used (like the `geneSetTest` function from the `limma` package, which is used as the default);
- `absolute`: logical specifying whether the absolute values of the ranking statistics should be used in the test (the default being TRUE);
- `gseFunc`: a function to perform GSE analysis, when not specified (the default) the `geneSetTest` from the `limma` package is used. When a function is specified, the membership of the analyzed genes to a FGS, and the ranking statistics must be defined in the same way this is done for `geneSetTest`, and the new function must return an integer (usually a p-value) (see the help for `geneSetTest` in the `limma` package)

Below are few examples to perform Wilcoxon rank-sum test over multiple FGS collections, and over multiple ranking statistics, usin the `runBatchGSE`. To this end we will use the **KEGG** and **GO** collections created above, and the separate and integrated gene-to-phenotype scores computed using the `computeDrStat`. The output of this function is a named list of lists, containing an element for each ranking statistics considered in the input. Each one of these elements, in turn, is another list, containing the GSE results for each collection sets. In the examples below we will therefore obtain a list of length one in the case ot the integrated gene-to-phenotype score, and a list of length four (on element for each genomic platform) in the case of the separate scores. For all the rankings we will obtain GSE result for both the collections of FGS.

3.4 INTEGRATION + GSE

The integrated gene-to-phenotype scores we have computed can be used to perform a GSE analysis. Below are reported few examples, using the default options, as well as passing several specific arguments to `geneSetTest` (see the relative help for details).

3.4.1 One-sided Wilcoxon rank-sum test using absolute ranking statistics

This can be accomplished by calling the `runBatchGSE` with default values, or by specifying each argument, as shown below:

```
> gseABS.int <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList)
> gseABS.int <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                  absolute=TRUE, type="f", alternative="mixed")
```

3.4.2 One-sided Wilcoxon rank-sum test using signed ranking statistics

When the signed ranking statistics has a sign, it is possible to perform a one-sided test assensing both tails separately, as well as a two-sided test. This can be accomplished by passing the corresponding arguments to `runBatchGSE`, as shown below:

```
> gseUP.int <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                  absolute=FALSE, type="t", alternative="up")
> gseDW.int <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                  absolute=FALSE, type="t", alternative="down")
> gseBOTH.int <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                  absolute=FALSE, type="t", alternative="either")
```

3.4.3 Performing a simulation-based GSE test

It is also possible to perform an enrichment analysis comparing each FGS to randomly selected gene lists of the same size of the FGS. In this case the p-value is computed by simulation as the proportion of times the mean of the statistics in the FGS is smaller (or larger) than in the `nsim` random simulated sets of genes.

```
> gseABSsim.int <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                     absolute=TRUE, type="f", alternative="mixed",
+                                     ranks.only=FALSE, nsim=1000)
> gseUPsim.int <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                     absolute=FALSE, type="t", alternative="up",
+                                     ranks.only=FALSE, nsim=1000)
```

Results from this analysis are named lists of lists, as shown below:

```
> str(gseUP.int)
```

```
List of 1
 $ integrated:List of 2
  ..$ go  : Named num [1:5] 0.871 0.918 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 0.295 0.433 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F
```

```
> gseABSsim.int
```

```
$integrated
$integrated$go
                              GO:0030804.positive regulation of cyclic nucleotide biosynthetic pr
                                                                                            0.181
                                     GO:0048009.insulin-like growth factor receptor signaling pa
                                                                                            0.088
```

```
                                                                    GO:0043237.laminin-1 bi

GO:0033842.N-acetyl-beta-glucosaminyl-glycoprotein 4-beta-N-acetylgalactosaminyltransferase act

                          GO:0046933.proton-transporting ATP synthase activity, rotational mech


$integrated$kegg
                          03050.Proteasome
                                        NA
             00565.Ether lipid metabolism
                                        NA
                    04510.Focal adhesion
                                 0.5944056
                00310.Lysine degradation
                                 0.5764236
00524.Butirosin and neomycin biosynthesis
                                        NA
```

3.4.4 Passsing alternative enrichment functions to `runBatchGSE`

Below is show how to define and pass alternative enrichment functions to `runBatchGSE`. We will first show how to use the `limma` `wilcoxGST` function, which is a synonym for `geneSetTest` using `ranks.only=TRUE` and `type="t"`.

```
> library(limma)
> gseUP.int.2 <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                  absolute=FALSE, gseFunc=wilcoxGST, alternative="up")
```

As shown below this approach will return the same results obtained with `geneSetTest` passing appropriate arguments.

```
> str(gseUP.int.2)
```

```
List of 1
 $ integrated:List of 2
  ..$ go  : Named num [1:5] 0.871 0.918 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 0.295 0.433 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F
```

```
> all(gseUP.int.2$go==gseUP.int$go)
```

```
[1] TRUE
```

We can finally also pass any new user-defined enrichment function, provided that the arguments are passed in the same way as with `geneSetTest`, as shown below using the Fisher's exact test, and a threshold for defining the list of differentially expressed genes.

```
> gseFunc <- function (selected, statistics, threshold) {
+         diffExpGenes <- statistics > threshold
```

```
+         tab <- table(diffExpGenes, selected)
+         pVal <- fisher.test(tab)[["p.value"]]
+         }
> gseUP.int.3 <- runBatchGSE(dataList=bicStatInt, fgsList=fgsList,
+                                  absolute=FALSE, gseFunc=gseFunc, threshold=7.5)
```

As shown below this approach will test for over-represtation of the a specific gene set within the genes defined as differentially expressed (in our example the genes showing an integrated association score larger than 7.5). Results are somewhat comparable to what obtained using the Wilcoxon rank-sum test.

```
> str(gseUP.int.3)
```

```
List of 1
 $ integrated:List of 2
  ..$ go  : Named num [1:5] 1 1 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 1 1 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F
```

```
> data.frame(fisher=gseUP.int.3$integrated$kegg,wilcoxon=gseUP.int$integrated$kegg)
```

```
                                       fisher
03050.Proteasome                           NA
00565.Ether lipid metabolism               NA
04510.Focal adhesion                        1
00310.Lysine degradation                    1
00524.Butirosin and neomycin biosynthesis  NA
                                      wilcoxon
03050.Proteasome                            NA
00565.Ether lipid metabolism                NA
04510.Focal adhesion                 0.2949405
00310.Lysine degradation             0.4328257
00524.Butirosin and neomycin biosynthesis   NA
```

3.5 GSE + INTEGRATION

The individual gene-to-phenotype scores computed for each platform can be similarly used to perform separate GSE analyses for each considered genomic platform, applying the same code and functions used to perform GSE analysis in the **INTEGRATION + GSE** approach above.

```
> gseABS.sep <- runBatchGSE(dataList=bicStatSep, fgsList=fgsList)
```

This step of GSE analysis on separate platform is then followed by GSE results integration, which is achieved using the `combineGSE` function, which summarizes the individual p-values from the tests. To this end different methods are available, including the computation of the geometric or arithmetic means, the use of the median, the selection of the minimun or the maximum p-value, and the random selection (respectively `geometricMean`, `mean`, `median`, `min`, `max`, and `random`). Few examples are shown below:

```
> gseABS.geoMean.sep <- combineGSE(gseABS.sep, method="geometricMean")
> gseABS.max.sep <- combineGSE(gseABS.sep, method="max")
```

Also in this case the results from the combination are named lists of lists, as shown below:

```
> names(gseABS.sep)

[1] "dat.affy"       "dat.agilent"
[3] "dat.cnvHarvard" "dat.cnvMskcc"

> str(gseABS.sep)

List of 4
 $ dat.affy      :List of 2
  ..$ go  : Named num [1:5] 0.2853 0.0819 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 0.542 0.67 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F
 $ dat.agilent   :List of 2
  ..$ go  : Named num [1:5] 0.133 0.394 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 0.7722 0.0569 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F
 $ dat.cnvHarvard:List of 2
  ..$ go  : Named num [1:5] 0.398 0.431 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 0.246 0.826 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F
 $ dat.cnvMskcc  :List of 2
  ..$ go  : Named num [1:5] 0.458 0.47 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 0.015 0.483 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F

> str(gseABS.geoMean.sep)

List of 1
 $ combinedScore:List of 2
  ..$ go  : Named num [1:5] 0.288 0.284 NA NA NA
  .. ..- attr(*, "names")= chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosy
  ..$ kegg: Named num [1:5] NA NA 0.198 0.351 NA
  .. ..- attr(*, "names")= chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.F

> gseABS.geoMean.sep

$combinedScore
$combinedScore$go
                          GO:0030804.positive regulation of cyclic nucleotide biosynthetic pr
                                                                                        0.28
                                GO:0048009.insulin-like growth factor receptor signaling pa
                                                                                        0.28
                                                                      GO:0043237.laminin-1 bi
```

```
GO:0033842.N-acetyl-beta-glucosaminyl-glycoprotein 4-beta-N-acetylgalactosaminyltransferase act

                          GO:0046933.proton-transporting ATP synthase activity, rotational mech


$combinedScore$kegg
                          03050.Proteasome
                                        NA
              00565.Ether lipid metabolism
                                        NA
                     04510.Focal adhesion
                                 0.1982863
                  00310.Lysine degradation
                                 0.3511865
00524.Butirosin and neomycin biosynthesis
                                        NA
```

3.6 Multiple testing correction

Finally the `adjustPvalGSE` enables to adjust the p-values computed by the `runBatchGSE`. This functions is an interface to the `mt.rawp2adjp` function from the `multtest` package.

```
> gseABS.int.BH <- adjustPvalGSE(gseABS.int)
> gseABS.int.holm <- adjustPvalGSE(gseABS.int, proc = "Holm")
```

Also in this case the results after the adjustment are named lists of lists, as shown below:

```
> names(gseABS.int.BH)

[1] "integrated"

> names(gseABS.int.holm)

[1] "integrated"

> str(gseABS.int.BH)

List of 1
 $ integrated:List of 2
  ..$ go  : num [1:5, 1:2] 0.1317 0.0829 NA NA NA ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosynthetic proce
  .. .. ..$ : chr [1:2] "rawp" "BH"
  ..$ kegg: num [1:5, 1:2] NA NA 0.713 0.573 NA ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.Focal adhesion
  .. .. ..$ : chr [1:2] "rawp" "BH"

> str(gseABS.int.holm)
```

```
List of 1
 $ integrated:List of 2
  ..$ go  : num [1:5, 1:2] 0.1317 0.0829 NA NA NA ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "GO:0030804.positive regulation of cyclic nucleotide biosynthetic proce
  .. .. ..$ : chr [1:2] "rawp" "Holm"
  ..$ kegg: num [1:5, 1:2] NA NA 0.713 0.573 NA ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "03050.Proteasome" "00565.Ether lipid metabolism" "04510.Focal adhesion
  .. .. ..$ : chr [1:2] "rawp" "Holm"
```

4 System Information

Session information:

```
> sessionInfo()

R version 3.3.0 RC (2016-04-26 r70550)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

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

attached base packages:
[1] stats4    parallel  stats     graphics
[5] grDevices utils     datasets  methods
[9] base

other attached packages:
 [1] limma_3.28.0         GO.db_3.3.0
 [3] KEGG.db_3.2.2        org.Hs.eg.db_3.3.0
 [5] AnnotationDbi_1.34.0 IRanges_2.6.0
 [7] S4Vectors_0.10.0     RTopper_1.18.0
 [9] Biobase_2.32.0       BiocGenerics_0.18.0

loaded via a namespace (and not attached):
 [1] lattice_0.20-33 MASS_7.3-45
 [3] grid_3.3.0      DBI_0.4
 [5] RSQLite_1.0.0   Matrix_1.2-6
 [7] splines_3.3.0   tools_3.3.0
 [9] survival_2.39-2 multtest_2.28.0
```

5 References