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

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

November 9, 2022

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 [6, 5, 7]. 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 [2] 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 [4, 8].

The integration can be achieved through two distinct approaches:

1. **GSE + INTEGRATION**: Separate GSE analysis 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 constructed *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 40128 mapped keys (of 66102 keys)
org.Hs.egACCNUM2EG has 806658 mapped keys (of 806658 keys)
org.Hs.egALIAS2EG has 131663 mapped keys (of 131663 keys)
org.Hs.egCHR has 65948 mapped keys (of 66102 keys)
org.Hs.egCHRLENGTHS has 640 mapped keys (of 640 keys)
org.Hs.egCHRLOC has 28289 mapped keys (of 66102 keys)
org.Hs.egCHRLOCEND has 28289 mapped keys (of 66102 keys)
org.Hs.egENSEMBL has 35530 mapped keys (of 66102 keys)
org.Hs.egENSEMBL2EG has 38636 mapped keys (of 38636 keys)
org.Hs.egENSEMBLPROT has 6895 mapped keys (of 66102 keys)
org.Hs.egENSEMBLPROT2EG has 21427 mapped keys (of 21427 keys)
org.Hs.egENSEMBLTRANS has 13066 mapped keys (of 66102 keys)
org.Hs.egENSEMBLTRANS2EG has 38893 mapped keys (of 38893 keys)
org.Hs.egENZYME has 2229 mapped keys (of 66102 keys)
org.Hs.egENZYME2EG has 975 mapped keys (of 975 keys)
org.Hs.egGENENAME has 66102 mapped keys (of 66102 keys)
org.Hs.egGENETYPE has 66102 mapped keys (of 66102 keys)
org.Hs.egGO has 20709 mapped keys (of 66102 keys)
org.Hs.egGO2ALLEGS has 22834 mapped keys (of 22834 keys)
org.Hs.egGO2EG has 18644 mapped keys (of 18644 keys)
org.Hs.egMAP has 64078 mapped keys (of 66102 keys)
org.Hs.egMAP2EG has 2003 mapped keys (of 2003 keys)
org.Hs.egOMIM has 16572 mapped keys (of 66102 keys)
org.Hs.egOMIM2EG has 22739 mapped keys (of 22739 keys)
org.Hs.egPATH has 5868 mapped keys (of 66102 keys)
org.Hs.egPATH2EG has 229 mapped keys (of 229 keys)
org.Hs.egPMID has 43141 mapped keys (of 66102 keys)
org.Hs.egPMID2EG has 737868 mapped keys (of 737868 keys)
org.Hs.egREFSEQ has 38890 mapped keys (of 66102 keys)
org.Hs.egREFSEQ2EG has 285470 mapped keys (of 285470 keys)
org.Hs.egSYMBOL has 66102 mapped keys (of 66102 keys)
org.Hs.egSYMBOL2EG has 66091 mapped keys (of 66091 keys)
org.Hs.egUCSCKG has 31379 mapped keys (of 66102 keys)
org.Hs.egUNIPROT has 18991 mapped keys (of 66102 keys)

Additional Information about this package:

DB schema: HUMAN_DB
DB schema version: 2.1
Organism: Homo sapiens
Date for NCBI data: 2022-Mar17
Date for GO data: 2022-03-10

```
Date for KEGG data: 2011-Mar15
Date for Golden Path data: 2022-Nov23
Date for Ensembl data: 2021-Dec21
```

For instance the `org.Hs.egGO2ALLEGS` environment contains the mapping of all ENTREZ Gene identifiers to the **Gene Ontology Terms** [1], while `org.Hs.egPATH2EG` maps the identifiers to **KEGG** pathways [3]. 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] 22834

> 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 [1:44] "142" "291" "1763" "1890" ...
  ..- attr(*, "names")= chr [1:44] "IMP" "TAS" "IDA" "IMP" ...
 $ GO:0000003: Named chr [1:1913] "2" "18" "49" "49" ...
  ..- attr(*, "names")= chr [1:1913] "IEA" "IEA" "IBA" "IEA" ...
 $ GO:0000012: Named chr [1:17] "142" "1161" "2074" "3981" ...
  ..- attr(*, "names")= chr [1:17] "IGI" "IDA" "IDA" "IDA" ...
 $ GO:0000017: Named chr [1:4] "6523" "6523" "6523" "6524"
  ..- attr(*, "names")= chr [1:4] "IDA" "IMP" "ISS" "IDA"
 $ GO:0000018: Named chr [1:153] "60" "86" "142" "604" ...
  ..- attr(*, "names")= chr [1:153] "IDA" "IDA" "IDA" "IEA" ...

> names(go)[1:5]

[1] "GO:0000002" "GO:0000003" "GO:0000012"
[4] "GO:0000017" "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:

```
> someKeggID <- c("00450", "04971", "00970", "04260", "05320")
> kegg <- lapply(kegg[someKeggID],function(x) unique(unlist(mget(x,org.Hs.egSYMBOL))))
> go <- lapply(go[sample(1:length(go),5)],function(x) unique(unlist(mget(x,org.Hs.egSYMBOL))))
> str(kegg)

List of 5
 $ 00450: chr [1:17] "KYAT1" "CTH" "MARS1" "MTR" ...
 $ 04971: chr [1:74] "ACTB" "ADCY1" "ADCY2" "ADCY3" ...
 $ 00970: chr [1:63] "AARS1" "CARS1" "DARS1" "EPRS1" ...
 $ 04260: chr [1:77] "ACTC1" "ATP1A1" "ATP1A2" "ATP1A3" ...
 $ 05320: chr [1:52] "FAS" "FASLG" "CD28" "CD80" ...

> str(go)

List of 5
 $ GO:1904830: chr "MIR21"
 $ GO:0006526: chr [1:5] "ASL" "ASS1" "CLN3" "OTC" ...
 $ GO:1902963: chr [1:2] "SORL1" "PICALM"
 $ GO:1904027: chr [1:3] "EMILIN1" "CHADL" "MIR29B1"
 $ GO:0051321: chr [1:266] "ATM" "ATRX" "BRCA2" "BRDT" ...
```

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

```
> library(KEGGREST)
> names(kegg) <-  sapply(keggGet(paste0("hsa", someKeggID)), "[[", "NAME")
```

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 28336 mapped keys (of 28336 keys)
GOBPCHILDREN has 16312 mapped keys (of 28336 keys)
GOBPOFFSPRING has 16312 mapped keys (of 28336 keys)
GOBPPARENTS has 28336 mapped keys (of 28336 keys)
GOCCANCESTOR has 4183 mapped keys (of 4183 keys)
GOCCCHILDREN has 1384 mapped keys (of 4183 keys)
GOCCOFFSPRING has 1384 mapped keys (of 4183 keys)
GOCCPARENTS has 4183 mapped keys (of 4183 keys)
GOMFANCESTOR has 11185 mapped keys (of 11185 keys)
GOMFCHILDREN has 2023 mapped keys (of 11185 keys)
GOMFOFFSPRING has 2023 mapped keys (of 11185 keys)
GOMFPARENTS has 11185 mapped keys (of 11185 keys)
GOOBSOLETE has 3703 mapped keys (of 3703 keys)
GOTERM has 43705 mapped keys (of 43705 keys)
```

```
Additional Information about this package:

DB schema: GO_DB
DB schema version: 2.1
Date for GO data: 2022-03-10
```

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

```
[1] "GO:1904830.negative regulation of aortic smooth muscle cell differentiation"
[2] "GO:0006526.arginine biosynthetic process"
[3] "GO:1902963.negative regulation of metalloendopeptidase activity involved in amyloid precurs
[4] "GO:1904027.negative regulation of collagen fibril organization"
[5] "GO:0051321.meiotic cell cycle"
```

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)
> fgsList$go
```

```
$`GO:1904830.negative regulation of aortic smooth muscle cell differentiation`
[1] "MIR21"

$`GO:0006526.arginine biosynthetic process`
[1] "ASL"  "ASS1" "CLN3" "OTC"  "NAGS"

$`GO:1902963.negative regulation of metalloendopeptidase activity involved in amyloid precursor
[1] "SORL1"  "PICALM"

$`GO:1904027.negative regulation of collagen fibril organization`
[1] "EMILIN1" "CHADL"   "MIR29B1"

$`GO:0051321.meiotic cell cycle`
  [1] "ATM"      "ATRX"     "BRCA2"    "BRDT"
  [5] "BUB1"     "BUB1B"    "OSGIN2"   "CALR"
  [9] "CCNE1"    "CDC20"    "CDC25A"   "CDC25B"
 [13] "CDC25C"   "CDK2"     "CENPC"    "CKS2"
 [17] "CORT"     "DMRT1"    "DUSP1"    "EDN1"
 [21] "EDNRA"    "ERCC1"    "EREG"     "ERCC4"
 [25] "FANCA"    "FANCD2"   "GOLGA2"   "GPR3"
 [29] "MSH6"     "H2AX"     "HSPA2"    "HUS1"
 [33] "INCENP"   "ING2"     "INSR"     "LFNG"
 [37] "LIF"      "MLH1"     "MOS"      "MRE11"
 [41] "MSH4"     "MSH5"     "MSX1"     "MSX2"
 [45] "MYBL1"    "MYH9"     "NBN"      "NEK2"
 [49] "NPPC"     "NPR2"     "NUMA1"    "OVOL1"
 [53] "PDE3A"    "PLK1"     "PPP2CA"   "PPP2R1A"
```

```
 [57] "PSMD13"   "RAD1"     "RAD21"    "RAD51"
 [61] "RAD51C"   "RAD51B"   "RAD51D"   "RBBP8"
 [65] "RPA1"     "RPS6KA2"  "AURKA"    "AURKC"
 [69] "SYCP1"    "TERF1"    "TOP2A"    "TOP2B"
 [73] "TOP3A"    "TTK"      "TUBG1"    "UBB"
 [77] "UBE2B"    "WNT5A"    "XRCC2"    "ZSCAN21"
 [81] "BAG6"     "SMC1A"    "RAD54L"   "FKBP6"
 [85] "CCNA1"    "PKMYT1"   "SMC3"     "CCNB2"
 [89] "CCNE2"    "EXO1"     "ZW10"     "BUB3"
 [93] "PTTG1"    "PIWIL1"   "TRIP13"   "MARF1"
 [97] "ESPL1"    "UTP14C"   "WASHC5"   "NCAPD2"
[101] "REC8"     "BCL2L11"  "SMC4"     "ACTR3"
[105] "ACTR2"    "RAD50"    "RBM7"     "NPM2"
[109] "SYCP2"    "NDC80"    "TUBGCP3"  "SMC2"
[113] "P3H4"     "RAD51AP1" "STAG3"    "PTTG2"
[117] "TUBGCP2"  "EHMT2"    "SPIN1"    "TDRKH"
[121] "HSF2BP"   "PRDM7"    "DMC1"     "FOXJ3"
[125] "SIRT2"    "PLCB1"    "UBR2"     "NCAPD3"
[129] "SUN1"     "NCAPH"    "SPO11"    "ZNF318"
[133] "SUN2"     "RAD54B"   "CATSPERZ" "PTTG3P"
[137] "FBXO5"    "MLH3"     "SMC1B"    "TUBG2"
[141] "TUBGCP4"  "NCAPH2"   "PSMC3IP"  "DNMT3L"
[145] "SLC2A8"   "SYCP3"    "DUSP13"   "FZR1"
[149] "YTHDF2"   "SIRT7"    "WNT4"     "MOV10L1"
[153] "DDX4"     "NSUN2"    "ZCWPW1"   "PIWIL2"
[157] "MNS1"     "NDC1"     "CHFR"     "FOXJ2"
[161] "TEX15"    "TEX14"    "TEX12"    "TEX11"
[165] "TDRD1"    "METTL3"   "CYP26B1"  "FMN2"
[169] "SPIRE1"   "PRDM9"    "FANCM"    "CCNB1IP1"
[173] "DMRTC2"   "FIGNL1"   "YTHDC2"   "BOLL"
[177] "BRME1"    "C11orf80" "RMI1"     "MUS81"
[181] "SHCBP1L"  "KIF18A"   "SLC25A31" "NUF2"
[185] "BRIP1"    "MND1"     "HORMAD1"  "SLX4"
[189] "SPIRE2"   "SPATA22"  "MASTL"    "MAEL"
[193] "TUBGCP6"  "CCNB3"    "RSPH1"    "TDRD12"
[197] "SYCE1"    "TUBGCP5"  "KLHDC3"   "SLC26A8"
[201] "TDRD9"    "CNTD1"    "M1AP"     "H1-8"
[205] "ZFP42"    "HUS1B"    "ASZ1"     "TAF1L"
[209] "STK35"    "PSMA8"    "PIWIL4"   "TERB2"
[213] "EME1"     "KASH5"    "PDIK1L"   "HORMAD2"
[217] "MEI1"     "SGO2"     "SGO1"     "MCMDC2"
[221] "SHOC1"    "EXD1"     "HFM1"     "WBP2NL"
[225] "FAM9A"    "FAM9B"    "FAM9C"    "AGO4"
[229] "EME2"     "PLD6"     "CENPX"    "SYCP2L"
[233] "MAPK15"   "SPDYA"    "MEIOB"    "ANKRD31"
[237] "SYCE2"    "ASPM"     "MAJIN"    "REC114"
[241] "TERB1"    "MEIOC"    "RNF212"   "FBXO43"
[245] "C14orf39" "NANOS2"   "IHO1"     "STRA8"
```

```
[249] "TUBB8"    "TOPAZ1"   "USP17L2"  "CENPS"
[253] "MEIOSIN"  "C1orf146" "TRIM75"   "TEX19"
[257] "PIWIL3"   "OOEP"     "WEE2"     "RAD21L1"
[261] "BTBD18"   "SYCE3"    "MEIKIN"   "SYCE1L"
[265] "RNF212B"  "MEI4"
```

3 Data analysis with RTopper

To compute gene-to-phenotype association scores the first step required is the conversion 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, containing 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 [9]);
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, using 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 of 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 assessing 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] NA NA NA 0.743 0.947
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 0.615 NA 0.454 0.391
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
```

```
> gseABSsim.int
```

```
$integrated
$integrated$go
                                                GO:1904830.negative regulation of aortic smooth musc

                                                                                      GO:0006526.argini

GO:1902963.negative regulation of metalloendopeptidase activity involved in amyloid precursor pr

                                                                    GO:1904027.negative regulation of colla

                                                                                                  GO:005



$integrated$kegg
  Selenocompound metabolism - Homo sapiens (human)
                                                NA
     Gastric acid secretion - Homo sapiens (human)
                                           0.3606394
Aminoacyl-tRNA biosynthesis - Homo sapiens (human)
                                                NA
 Cardiac muscle contraction - Homo sapiens (human)
                                           0.6023976
 Autoimmune thyroid disease - Homo sapiens (human)
                                           0.5994006
```

3.4.4 Passing 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] NA NA NA 0.743 0.947
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 0.615 NA 0.454 0.391
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
```

```
> 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 deferentially 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-representation of the a specific gene set within the genes defined as deferentially 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] NA NA NA 1 1
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 1 NA 1 1
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
```

```
> cat("Fisher:")
```

```
Fisher:
```

```
> gseUP.int.3$integrated$kegg
```

```
  Selenocompound metabolism - Homo sapiens (human)
                                                NA
     Gastric acid secretion - Homo sapiens (human)
                                                 1
Aminoacyl-tRNA biosynthesis - Homo sapiens (human)
                                                NA
 Cardiac muscle contraction - Homo sapiens (human)
                                                 1
 Autoimmune thyroid disease - Homo sapiens (human)
                                                 1
```

```
> cat("\n Wilcoxon:")

 Wilcoxon:

> gseUP.int$integrated$kegg

  Selenocompound metabolism - Homo sapiens (human)
                                                NA
     Gastric acid secretion - Homo sapiens (human)
                                         0.6154686
Aminoacyl-tRNA biosynthesis - Homo sapiens (human)
                                                NA
 Cardiac muscle contraction - Homo sapiens (human)
                                         0.4541267
 Autoimmune thyroid disease - Homo sapiens (human)
                                         0.3908406
```

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 minimum 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] NA NA NA 0.514 0.874
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 0.844 NA 0.414 0.262
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
 $ dat.agilent   :List of 2
  ..$ go  : Named num [1:5] NA NA NA 0.781 0.47
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 0.432 NA 0.908 0.957
```

```
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
 $ dat.cnvHarvard:List of 2
  ..$ go  : Named num [1:5] NA NA NA 0.829 0.274
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 0.855 NA 0.622 0.357
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
 $ dat.cnvMskcc  :List of 2
  ..$ go  : Named num [1:5] NA NA NA 0.0682 0.1915
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 0.171 NA 0.0899 0.5715
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
```

> str(gseABS.geoMean.sep)

```
List of 1
 $ combinedScore:List of 2
  ..$ go  : Named num [1:5] NA NA NA 0.388 0.383
  .. ..- attr(*, "names")= chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cel
  ..$ kegg: Named num [1:5] NA 0.48 NA 0.381 0.475
  .. ..- attr(*, "names")= chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric
```

> gseABS.geoMean.sep

```
$combinedScore
$combinedScore$go
                                              GO:1904830.negative regulation of aortic smooth musc

                                                                                  GO:0006526.argini

GO:1902963.negative regulation of metalloendopeptidase activity involved in amyloid precursor pr

                                                                GO:1904027.negative regulation of colla

                                                                                              GO:005


$combinedScore$kegg
  Selenocompound metabolism - Homo sapiens (human)
                                                NA
     Gastric acid secretion - Homo sapiens (human)
                                         0.4804713
Aminoacyl-tRNA biosynthesis - Homo sapiens (human)
                                                NA
 Cardiac muscle contraction - Homo sapiens (human)
                                         0.3807105
 Autoimmune thyroid disease - Homo sapiens (human)
                                         0.4754729
```

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] NA NA NA 0.2597 0.0539 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cell differentia
  .. .. ..$ : chr [1:2] "rawp" "BH"
  ..$ kegg: num [1:5, 1:2] NA 0.389 NA 0.554 0.614 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric acid secreti
  .. .. ..$ : chr [1:2] "rawp" "BH"
```

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

```
List of 1
 $ integrated:List of 2
  ..$ go  : num [1:5, 1:2] NA NA NA 0.2597 0.0539 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "GO:1904830.negative regulation of aortic smooth muscle cell differentia
  .. .. ..$ : chr [1:2] "rawp" "Holm"
  ..$ kegg: num [1:5, 1:2] NA 0.389 NA 0.554 0.614 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:5] "Selenocompound metabolism - Homo sapiens (human)" "Gastric acid secreti
  .. .. ..$ : chr [1:2] "rawp" "Holm"
```

4 System Information

Session information:

```
> sessionInfo()
```

```
R version 4.2.1 (2022-06-23)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.0
```

```
Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib

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    stats     graphics  grDevices
[5] utils     datasets  methods   base

other attached packages:
 [1] limma_3.54.0         GO.db_3.15.0
 [3] KEGGREST_1.38.0      org.Hs.eg.db_3.15.0
 [5] AnnotationDbi_1.60.0 IRanges_2.32.0
 [7] S4Vectors_0.36.0     RTopper_1.44.1
 [9] Biobase_2.58.0       BiocGenerics_0.44.0

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.9              compiler_4.2.1
 [3] GenomeInfoDb_1.34.2     XVector_0.38.0
 [5] bitops_1.0-7            tools_4.2.1
 [7] zlibbioc_1.44.0         bit_4.0.4
 [9] RSQLite_2.2.14          memoise_2.0.1
[11] lattice_0.20-45         pkgconfig_2.0.3
[13] png_0.1-7               rlang_1.0.4
[15] Matrix_1.4-1            DBI_1.1.3
[17] cli_3.3.0               curl_4.3.2
[19] fastmap_1.1.0           GenomeInfoDbData_1.2.8
[21] httr_1.4.3              Biostrings_2.66.0
[23] vctrs_0.4.1             bit64_4.0.5
[25] multtest_2.54.0         grid_4.2.1
[27] R6_2.5.1                survival_3.3-1
[29] blob_1.2.3              MASS_7.3-58
[31] splines_4.2.1           RCurl_1.98-1.7
[33] cachem_1.0.6            crayon_1.5.1
```

5 References

References

[1] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–9, 2000. 1061-4036 (Print) Journal Article.

[2] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B*, 57:289–300, 1995.

[3] M. Kanehisa, S. Goto, S. Kawashima, Y. Okuno, and M. Hattori. The KEGG resource for deciphering the genome. *Nucleic Acids Res*, 32(Database issue):D277–80, 2004. 1362-4962 (Electronic) Journal Article.

[4] V. K. Mootha, C. M. Lindgren, K. F. Eriksson, A. Subramanian, S. Sihag, J. Lehar, P. Puigserver, E. Carlsson, M. Ridderstrale, E. Laurila, N. Houstis, M. J. Daly, N. Patterson, J. P. Mesirov, T. R. Golub, P. Tamayo, B. Spiegelman, E. S. Lander, J. N. Hirschhorn, D. Altshuler, and L. C. Groop. PGC-1alpha-responsive genes involved in oxidative phosphorylation are coordinately downregulated in human diabetes. *Nat Genet*, 34(3):267–273, 2003. 1061-4036 (Print) Journal Article.

[5] G. K. Smyth. Linear models and empirical Bayes methods for assessing differential expression in microarray experiments. *Statistical Applications in Genetics and Molecular Biology*, 3(Article 3), 2004.

[6] G. K. Smyth. Limma: linear models for microarray data. In R. Gentleman, R. V. Carey, S. Dudoit, R. Irizarry, and W. Huber, editors, *Bioinformatics and Computational Biology Solutions using R and Bioconductor*, pages 397–420. Springer, New York, 2005.

[7] G. K. Smyth, J. Michaud, and H. S. Scott. Use of within-array replicate spots for assessing differential expression in microarray experiments. *Bioinformatics*, 2005. 1367-4803 (Print) Evaluation Studies Journal Article Validation Studies.

[8] Aravind Subramanian, Pablo Tamayo, Vamsi K Mootha, Sayan Mukherjee, Benjamin L Ebert, Michael A Gillette, Amanda Paulovich, Scott L Pomeroy, Todd R Golub, Eric S Lander, and Jill P Mesirov. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci U S A*, 102(43):15545–15550, Oct 2005.

[9] Svitlana Tyekucheva, Luigi Marchionni, Rachel Karchin, and Giovanni Parmigiani. Integrating diverse genomic data using gene sets. *Genome Biology (in press)*, 2011.