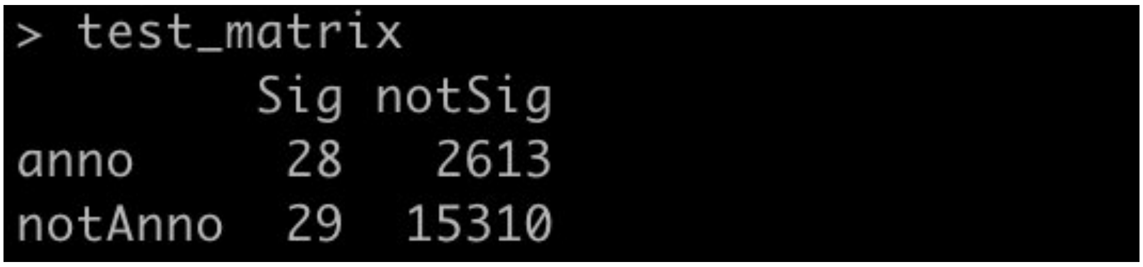
Gene Ontology可分为分子功能(Molecular Function), 生物过程(Biological Process)和细胞组成(Cellular Component)三部分。蛋白质或基因可以通过ID对应或者序列注释的方法找到与之对应的GO号，而GO号可对于到Term，即功能类别或着细胞定位。

GO的一个主要用途就是针对一组gene进行富集分析。给定的一组在特定条件下高表达的gene，使用gene的注释信息，富集分析可以发现哪个GO terms为高表达的。

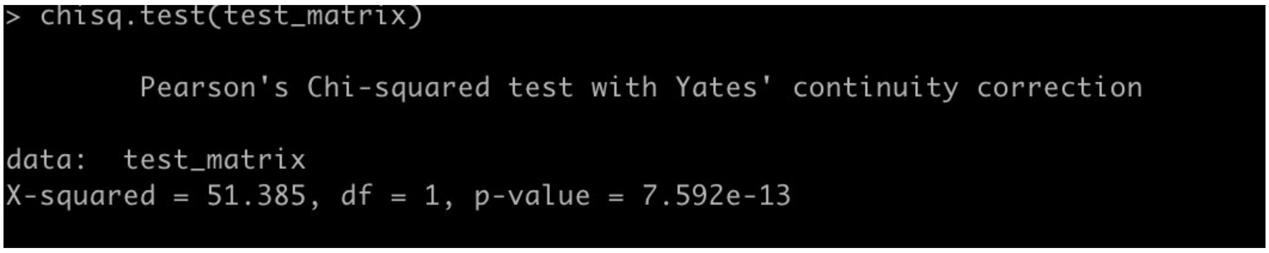
给定注释信息，我们可以将gene分为属于此类和非此类两组，就可以得到一个2x2的列联表做独立性分析，针对列联表就可以采用卡方检测和fisher's exact test，卡方检验只是近似估计值，特别是当sample size活expected all count比较小时，计算不够准确。fisher's exact test，使用超几何分布计算p值，比较准确。

例如共N个gene，其中M个属于该分类，那么抽取n个gene(来自同一个样本挑选出来用于富集分析的gene)，求其中有k个gene属于该分类的p值。



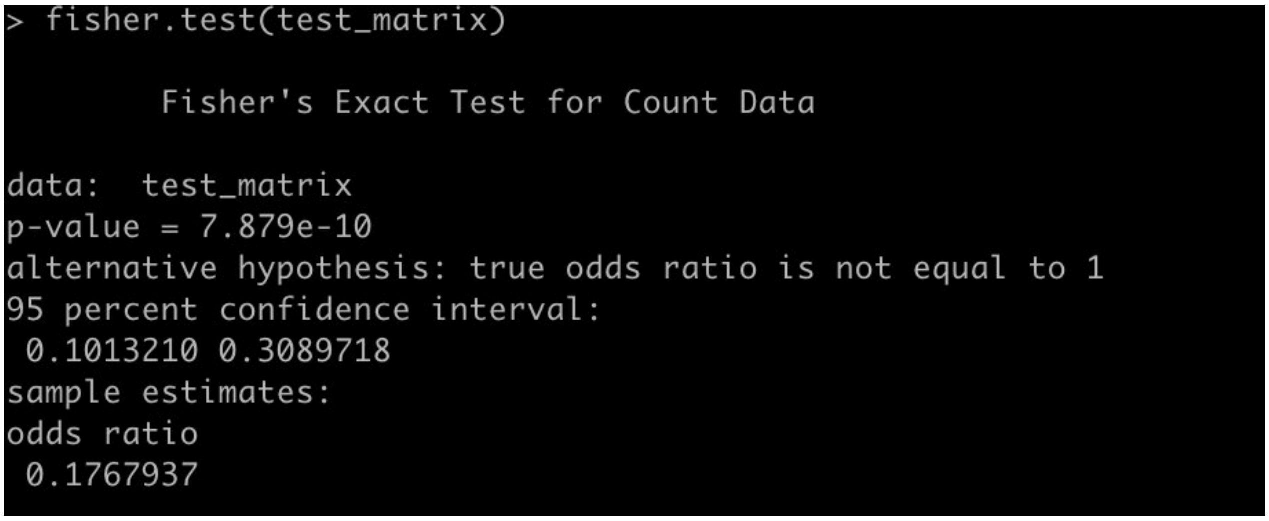
```
> test_matrix
         Sig notSig
anno      28   2613
notAnno   29  15310
```

针对该2x2表做独立性分析，卡方检验：



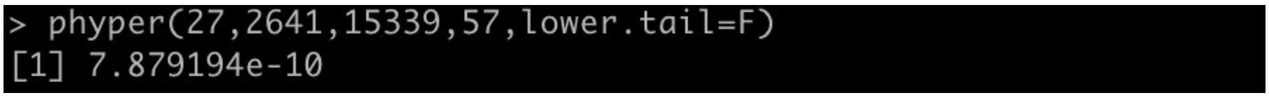
```
> chisq.test(test_matrix)

	Pearson's Chi-squared test with Yates' continuity correction

data:  test_matrix
X-squared = 51.385, df = 1, p-value = 7.592e-13
```

无放回抽样检测符合超几何分布，fisher's exact test就是使用超几何分布计算p值:



```
> fisher.test(test_matrix)

	Fisher's Exact Test for Count Data

data:  test_matrix
p-value = 7.879e-10
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.1013210 0.3089718
sample estimates:
odds ratio
 0.1767937
```

超几何检测，phyper(k-1,M,N-M,n,lower.tail=F)，p值：



```
> phyper(27,2641,15339,57,lower.tail=F)
[1] 7.879194e-10
```

摘抄：https://guangchuangyu.github.io/cn/2012/04/enrichment-analysis/

Loading genes and annotations data

`library(topGO)`

`library(ALL)`

`data(ALL)`

topGO package同时创建三个环境：GOBPTerm, GOMFTerm和GOCCTerm，对应为BP/MT/CC的所有GO terms。

`BPterms <- ls(GOBPTerm)`

一般而言，需要过滤掉低表达或者样本间变异很小的探针信息，采用genefilter package，genefilter函数处理。

`library(genefilter)`

`selProbes <- genefilter(ALL, filterfun(pOverA(0.1,log2(100)), function(x)(IQR(x)>0.25)))`

A filter function to filter according to the proportion of elements larger than A:pOverA(p=0.05, A=100, na.rm=TRUE)

`eset <- ALL[selProbes, ]`

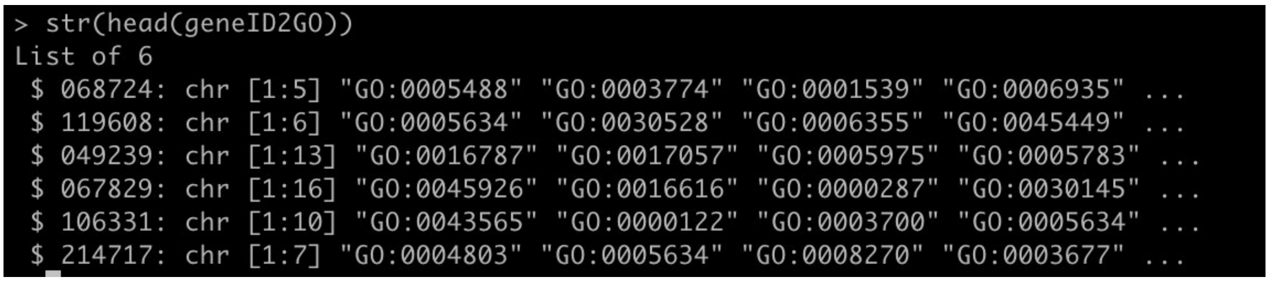
来自array或研究的所有gene为gene universe，当可得到gene-wise scores时，interesting genes为具有显著score的一组gene；或者直接定义一组gene为intereseting gene。

topGO进行富集分析的核心步骤就是创建topGOdata对象，该对象包含了用于GO分析的所有信息：gene universe, interesting gene, gene score(if available), GO ontology(GO graph)。

- 一组gene id及可选的gene-wise score。score可用于差异表达的t-test检验，表型的相关性...
- gene id和GO term之间的对应关系，一般可直接从bioconductor或microarray获得
- GO的分级结构，来自GO.db package

定制化注释信息也可用于构建topGOdata对象，注释信息需要以gene-to-GO或GO-to-gene的形式提供。

`geneID2GO <- readMapings(file=system.file("examples/geneid2go.map", package="topGO"))`



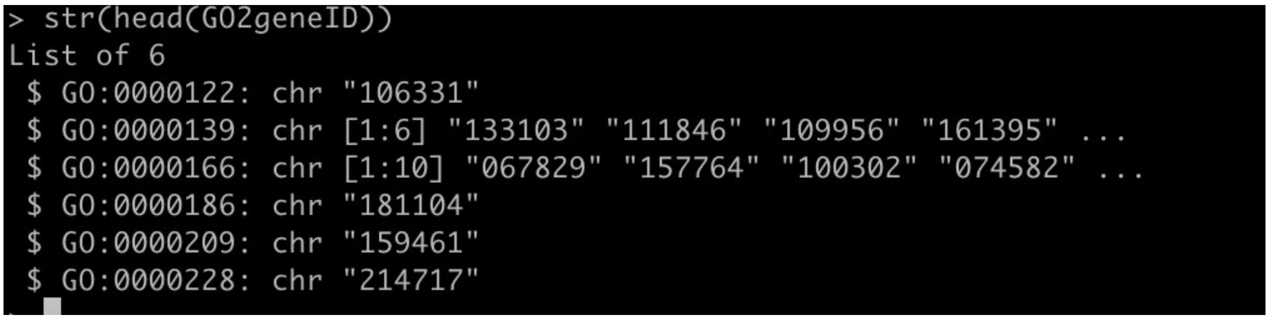
```
> str(head(geneID2GO))
List of 6
 $ 068724: chr [1:5] "GO:0005488" "GO:0003774" "GO:0001539" "GO:0006935" ...
 $ 119608: chr [1:6] "GO:0005634" "GO:0030528" "GO:0006355" "GO:0045449" ...
 $ 049239: chr [1:13] "GO:0016787" "GO:0017057" "GO:0005975" "GO:0005783" ...
 $ 067829: chr [1:16] "GO:0045926" "GO:0016616" "GO:0000287" "GO:0030145" ...
 $ 106331: chr [1:10] "GO:0043565" "GO:0000122" "GO:0003700" "GO:0005634" ...
 $ 214717: chr [1:7] "GO:0004803" "GO:0005634" "GO:0008270" "GO:0003677" ...
```

同时可以自定义文本来构建geneID2GO文件或GO2geneID文件供readMappings函数读取

`gene_ID<TAB>GO_ID1, GO_ID2, GO_ID3, ….`

gene-to-GO和GO-to-gene之间关系互转

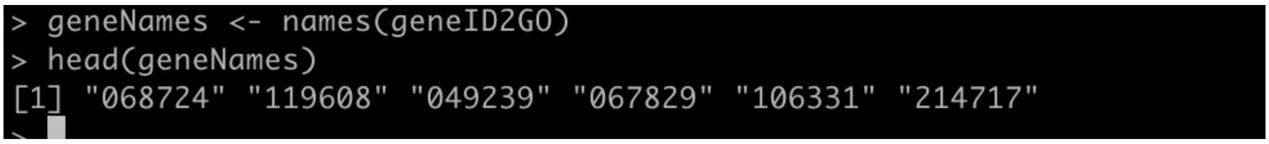
`GO2geneID <- inverseList(geneID2GO)`



```
> str(head(GO2geneID))
List of 6
 $ GO:0000122: chr "106331"
 $ GO:0000139: chr [1:6] "133103" "111846" "109956" "161395" ...
 $ GO:0000166: chr [1:10] "067829" "157764" "100302" "074582" ...
 $ GO:0000186: chr "181104"
 $ GO:0000209: chr "159461"
 $ GO:0000228: chr "214717"
```

- **定义一套interesting genes用于GO terms富集分析，仅需输入一组感兴趣的gene即可，这时就可以根据gene count计算统计显著性，例如fisher‘s exact test，Z scores， RNA-seq首选**

`geneNames <- names(geneID2GO)`



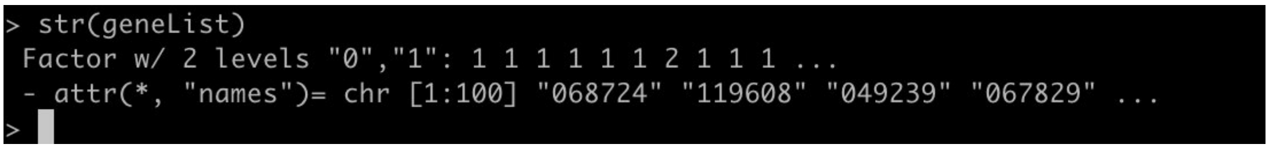
```
> geneNames <- names(geneID2GO)
> head(geneNames)
[1] "068724" "119608" "049239" "067829" "106331" "214717"
```

随机挑选10%的gene universe作为interesting genes

`myInterestingGenes <- sample(geneNames, length(geneNames)/10)`

`geneList <- factor(as.integer(geneNames %in% myInterestingGenes))`

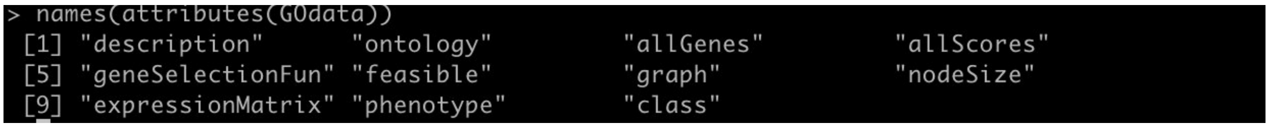
`names(geneList) <- geneNames`



```
> str(geneList)
 Factor w/ 2 levels "0","1": 1 1 1 1 1 1 2 1 1 1 ...
 - attr(*, "names")= chr [1:100] "068724" "119608" "049239" "067829" ...
>
```

geneList对象用于命名因子指定哪些gene是interesting，哪些gene不是

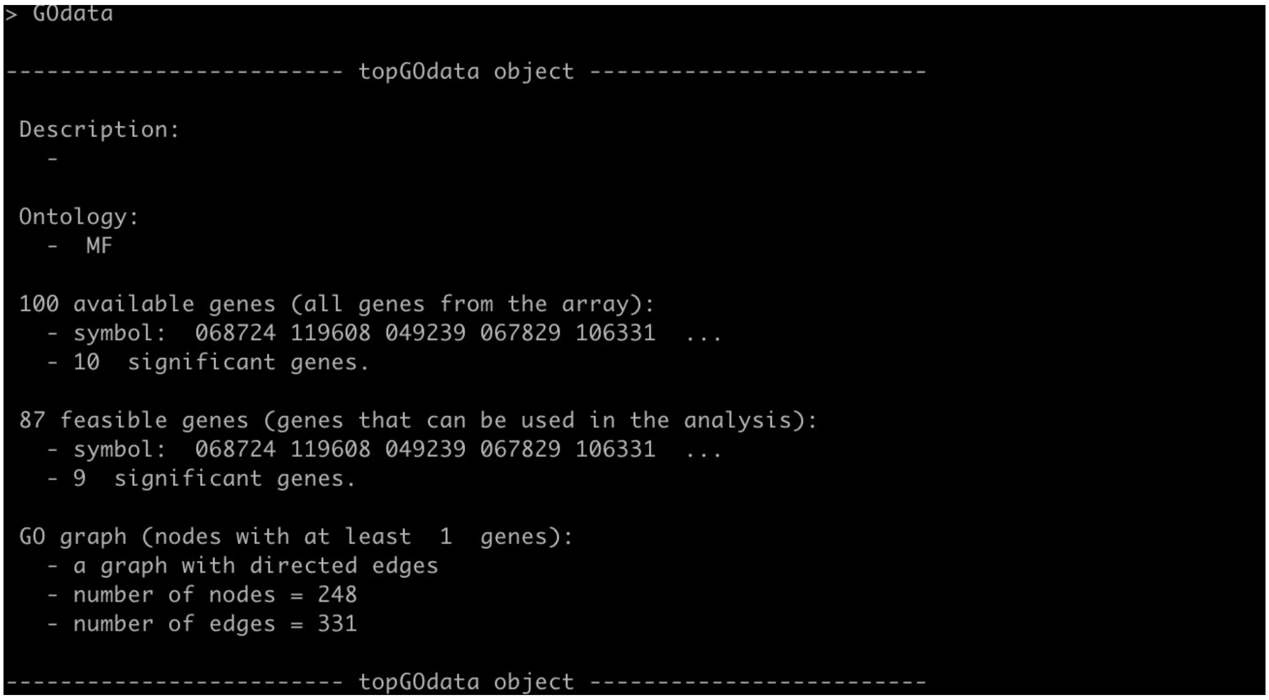
```
GOdata <- new("topGOdata", ontology="MF", allGenes=geneList,
annot=annFUN.gene2GO, gene2GO=geneID2GO, nodeSize=5)
```



```
> names(attributes(GOdata))
 [1] "description"       "ontology"          "allGenes"          "allScores"
 [5] "geneSelectionFun"  "feasible"          "graph"             "nodeSize"
 [9] "expressionMatrix"  "phenotype"         "class"
```

可见new "topGodata"拥有这些slots可以存储信息

feasible genes是由microarray来定义的

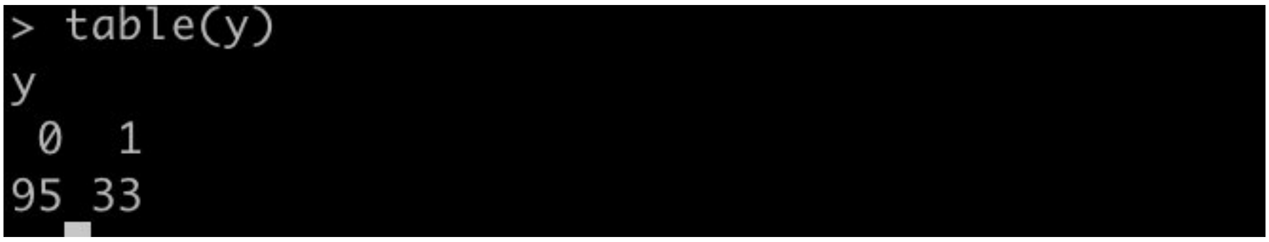


```
> GOdata

------------------------ topGOdata object ------------------------

 Description:
   -

 Ontology:
   -  MF

 100 available genes (all genes from the array):
   - symbol:  068724 119608 049239 067829 106331  ...
   - 10  significant genes.

 87 feasible genes (genes that can be used in the analysis):
   - symbol:  068724 119608 049239 067829 106331  ...
   - 9  significant genes.

 GO graph (nodes with at least  1  genes):
   - a graph with directed edges
   - number of nodes = 248
   - number of edges = 331

------------------------ topGOdata object ------------------------
```

- **大多数情况下，可以根据gene的score来计算interesting gene，例如根据差异表达基因的p-value。此时，topGOdata对象能够存储gene score，并根据规则来指定interesting gene。这样，就可以在不修改输入数据的情况下选择不同的统计方法进行GO分析。**

例如，discriminate between ALL cells delivered from either B-cell or T-cell precursors.

`y <- as.integer(sapply(eset$BT, function(x)return(substr(x,1,1) == "T")))`



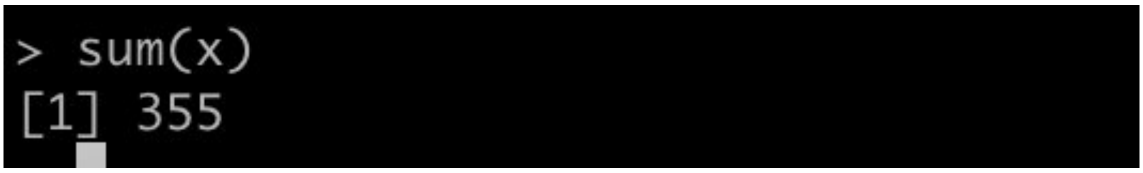
```
> table(y)
y
 0  1
95 33
```

`geneList <- getPvalues(exprs(eset), classlabel=y, alternative="greater")`

getPvalues,针对gene表达矩阵计算gene的adjusted-p值 `getPvalues(edata, classlabel, test = "t", alternative = c("greater", "two.sided", "less”)[1],genesID = NULL, correction = c("none", "Bonferroni", "Holm", "Hochberg", "SidakSS", "SidakSD” ,”BH", "BY")[8])`

`topDiffGenes <- function(allScores){return(allScore < 0.01)}`

`x <- topDiffGenes(geneList)`



```
> sum(x)
[1] 355
```

选择p-adjusted<0.01的gene

根据以上信息创建topGOdata

`GOdata <- new("topGOdata", description="GO analysis of ALL data: B-cell vs T-cell", ontology="BP", allGenes=geneList, geneSelectionFun= topDiffGenes, annot=annFUN.db, nodeSize=5, affyLib=affyLib)`

过滤掉少于5个annotated gene的GO term，一般设置5-10个会带来较稳定的结果，默认为1，意味着不删除任何GO term。

理想状态下，只有噪音探针，低表达探针和样本间小变异探针才会从分析中过滤出去。探针的数目对多重检验adjustment of p-values具有直接影响，太多的探针将会导致过于保守的adjsuted-p values，会导致Fisher's exact test结果偏差。

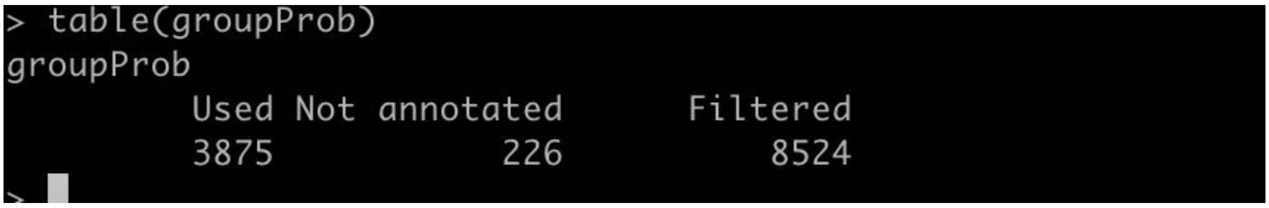
`allProb <- featureNames(ALL)`

`groupProb <- integer(length(allProb))+1`

`groupProb[allProb %in% genes(GOdata)] <- 0`，去除GOdata中没有的genes，probes

`groupProb[!selProbes] <- 2`

`groupProb <- factor(groupProb, labels=c("Used", "Not annotated", "Filtered"))`



```
> table(groupProb)
groupProb
         Used Not annotated      Filtered
         3875           226          8524
> 
```

在当前可行的probes执行差异表达分析，检测差异表达的genes是被富集分析排除在外

`pValue <- getPvalues(exprs(ALL), classlabel=y,alternative="greater")`

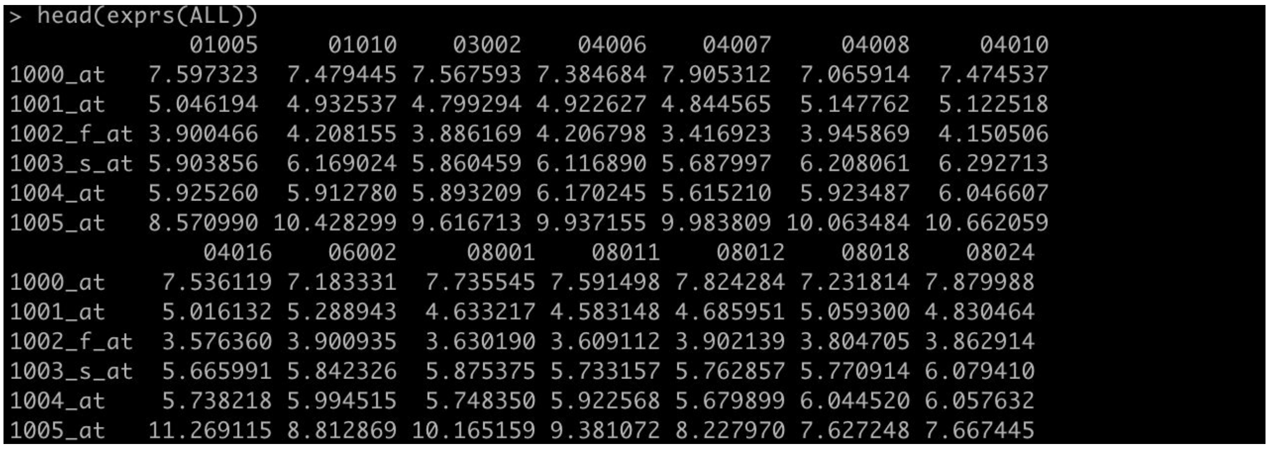
`geneVar <- apply(expres(ALL), 1, var)`

`dd <- data.frame(x=geneVar[allProb], y= log10(pValue[allProb]), groups=groupProb)`

`library(lattice)`

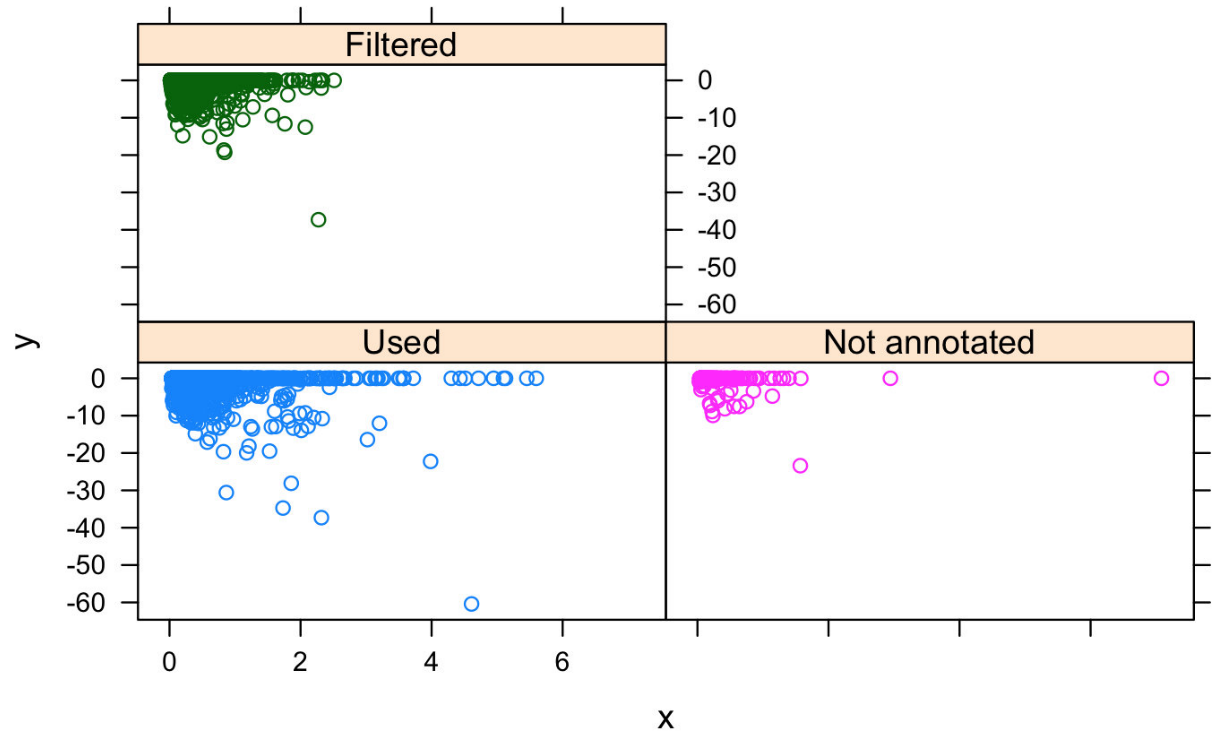
`xyplot(y~x|groups, data=dd, groups=groups)`

exprs获得ALL的表达矩阵信息(12625行，128列)，getPvalues函数根据表达矩阵，计算p-values，classlabel指定数据的表型，默认test为t检验，correction默认为'BH'



```
> head(exprs(ALL))
               01005     01010    03002    04006    04007     04008     04010
1000_at   7.597323  7.479445 7.567593 7.384684 7.905312  7.065914  7.474537
1001_at   5.046194  4.932537 4.799294 4.922627 4.844565  5.147762  5.122518
1002_f_at 3.900466  4.208155 3.886169 4.206798 3.416923  3.945869  4.150506
1003_s_at 5.903856  6.169024 5.860459 6.116890 5.687997  6.208061  6.292713
1004_at   5.925260  5.912780 5.893209 6.170245 5.615210  5.923487  6.046607
1005_at   8.570990 10.428299 9.616713 9.937155 9.983809 10.063484 10.662059
              04016     06002     08001    08011    08012    08018    08024
1000_at    7.536119 7.183331  7.735545 7.591498 7.824284 7.231814 7.879988
1001_at    5.016132 5.288943  4.633217 4.583148 4.685951 5.059300 4.830464
1002_f_at  3.576360 3.900935  3.630190 3.609112 3.902139 3.804705 3.862914
1003_s_at  5.665991 5.842326  5.875375 5.733157 5.762857 5.770914 6.079410
1004_at    5.738218 5.994515  5.748350 5.922568 5.679899 6.044520 6.057632
1005_at   11.269115 8.812869 10.165159 9.381072 8.227970 7.627248 7.667445
```

没做过microarrary分析，具体不太清楚



针对topGOdata对象，获得相关信息

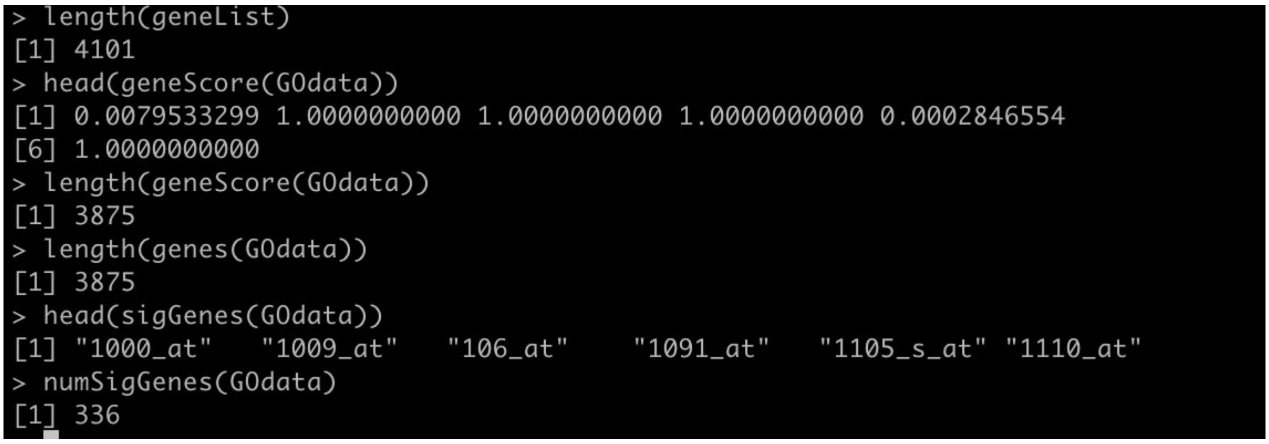
description函数获得GOdata的描述信息

`genes(GOdata)`获得feasible gene list

查询一组gene的score

`selGenes <- sample(genes(GOdata), 10)`

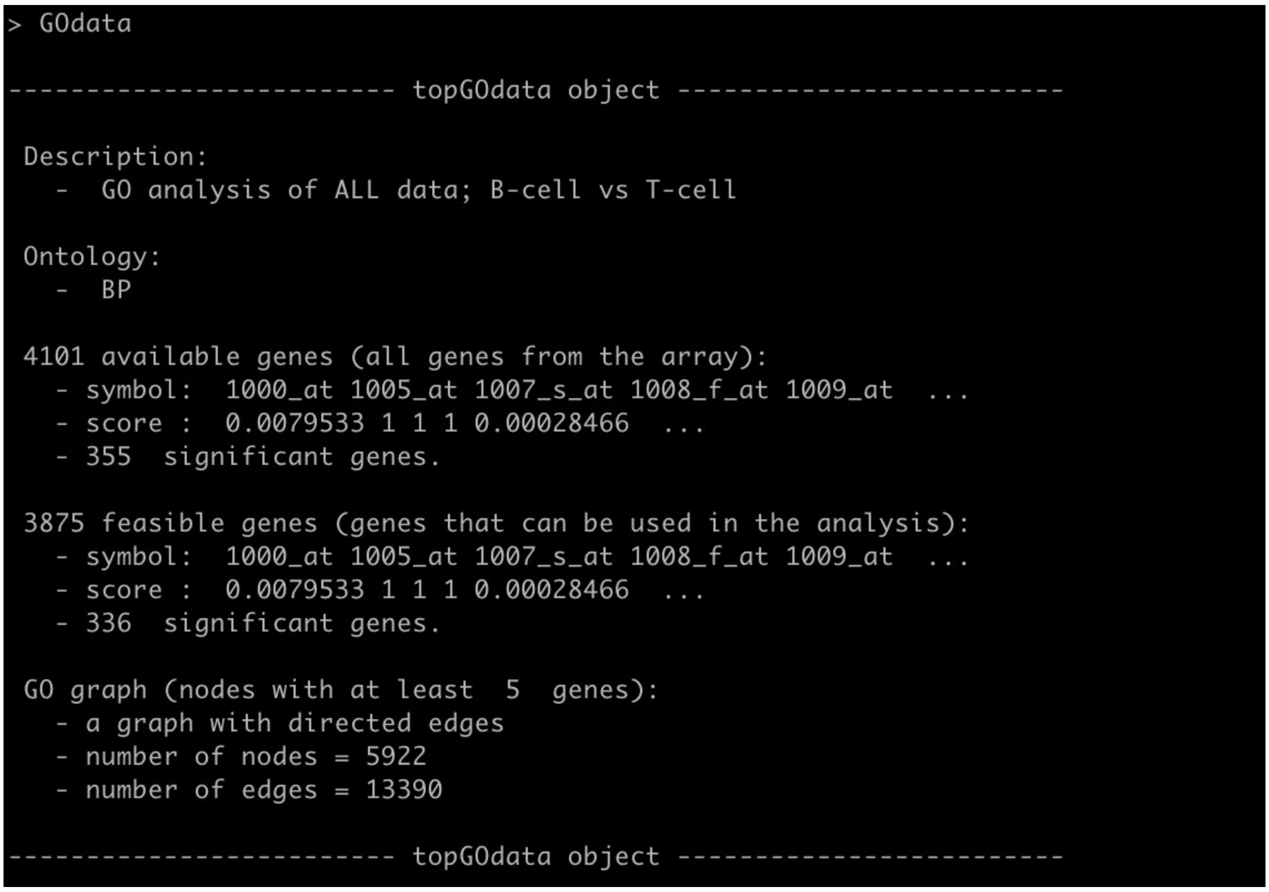
`gs <- geneScore(GOdata, whichGenes=selGenes)`



```
> length(geneList)
[1] 4101
> head(geneScore(GOdata))
[1] 0.0079533299 1.0000000000 1.0000000000 1.0000000000 0.0002846554
[6] 1.0000000000
> length(geneScore(GOdata))
[1] 3875
> length(genes(GOdata))
[1] 3875
> head(sigGenes(GOdata))
[1] "1000_at"   "1009_at"   "106_at"    "1091_at"   "1105_s_at" "1110_at"
> numSigGenes(GOdata)
[1] 336
```

若想更新topGOdata对象，仅包含feasible ones

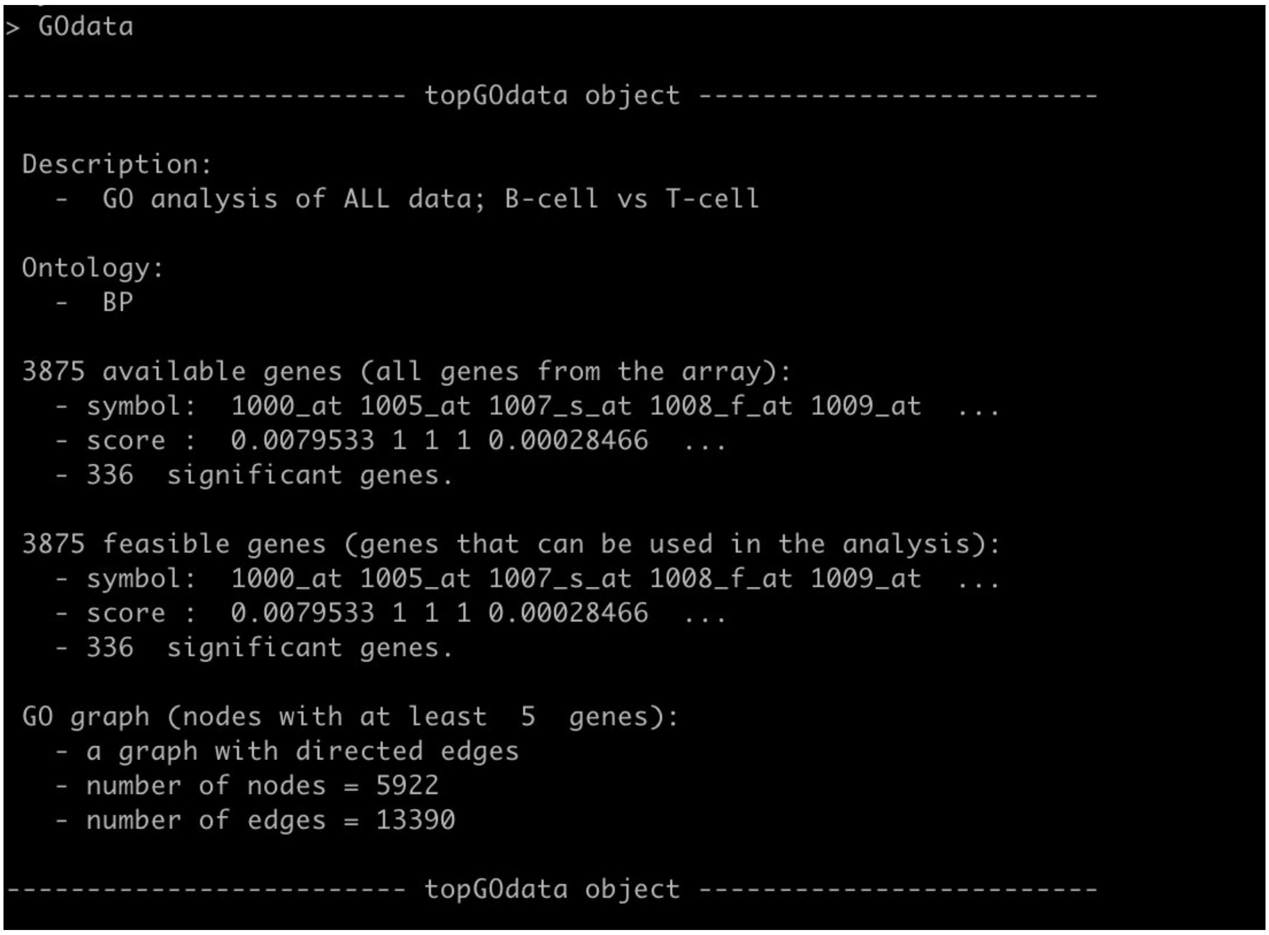
`.geneList <- geneScore(GOdata, use.names=T)`



```
> GOdata

------------------------- topGOdata object -------------------------

 Description:
   -  GO analysis of ALL data; B-cell vs T-cell

 Ontology:
   -  BP

 4101 available genes (all genes from the array):
   - symbol:  1000_at 1005_at 1007_s_at 1008_f_at 1009_at  ...
   - score :  0.0079533 1 1 1 0.00028466  ...
   - 355  significant genes.

 3875 feasible genes (genes that can be used in the analysis):
   - symbol:  1000_at 1005_at 1007_s_at 1008_f_at 1009_at  ...
   - score :  0.0079533 1 1 1 0.00028466  ...
   - 336  significant genes.

 GO graph (nodes with at least  5  genes):
   - a graph with directed edges
   - number of nodes = 5922
   - number of edges = 13390

------------------------- topGOdata object -------------------------
```

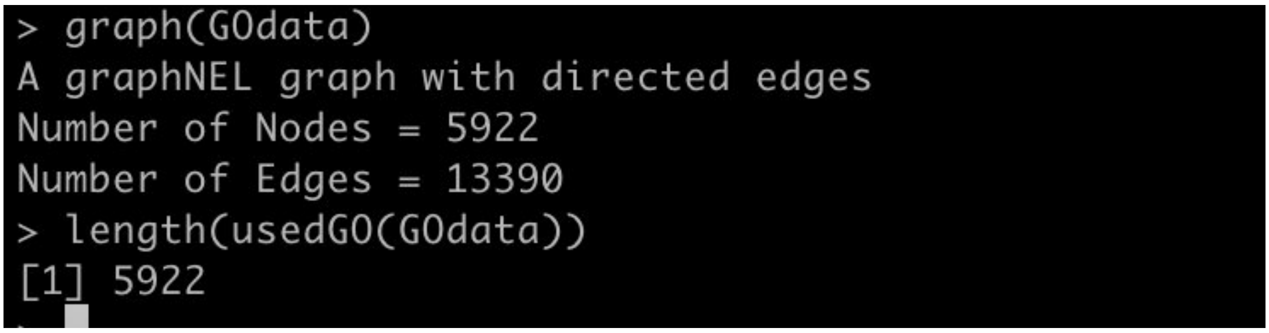
`GOdata <- updateGenes(GOdata, .geneList, topDiffGenes)`

topDiffGenes，指定p值小于0.01的gene；其中226个genes是不含注释信息的(共4101)



```
> GOdata

------------------------- topGOdata object -------------------------

 Description:
   -  GO analysis of ALL data; B-cell vs T-cell

 Ontology:
   -  BP

 3875 available genes (all genes from the array):
   - symbol:  1000_at 1005_at 1007_s_at 1008_f_at 1009_at  ...
   - score :  0.0079533 1 1 1 0.00028466  ...
   - 336  significant genes.

 3875 feasible genes (genes that can be used in the analysis):
   - symbol:  1000_at 1005_at 1007_s_at 1008_f_at 1009_at  ...
   - score :  0.0079533 1 1 1 0.00028466  ...
   - 336  significant genes.

 GO graph (nodes with at least  5  genes):
   - a graph with directed edges
   - number of nodes = 5922
   - number of edges = 13390

------------------------- topGOdata object -------------------------
```

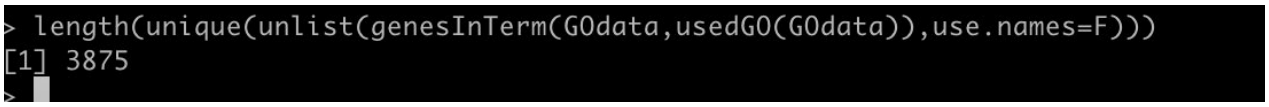
这样就排除了无注释信息的226个gene，就是GOdata中不包含的genes/probes



```
> graph(GOdata)
A graphNEL graph with directed edges
Number of Nodes = 5922
Number of Edges = 13390
> length(usedGO(GOdata))
[1] 5922
```

graph可返回GOdata中的GO term(node)数据，已经这些GO term所包含的edge数目(GO term之间的连线)

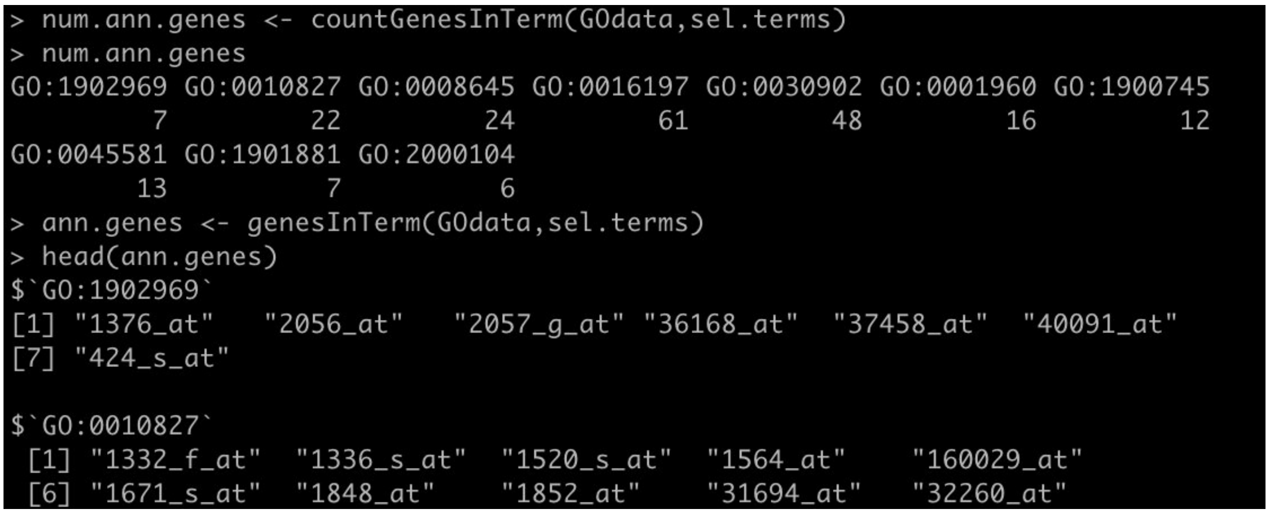
获得参与GOdata graph的所有gene数目



```
> length(unique(unlist(genesInTerm(GOdata,usedGO(GOdata)),use.names=F)))
[1] 3875
>
```

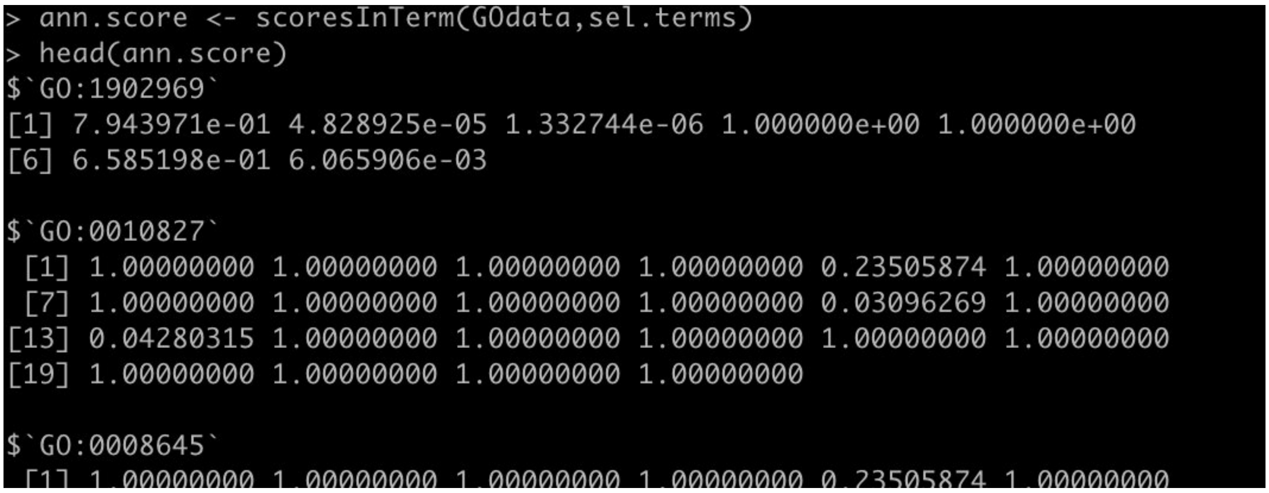
usedGO(GOdata)，对应返回GO term信息，genesInTerm(GOdata,sel.terms)，根据gel.terms返回对应的genes信息

`sel.terms <- sample(usedGO(GOdata),10)`



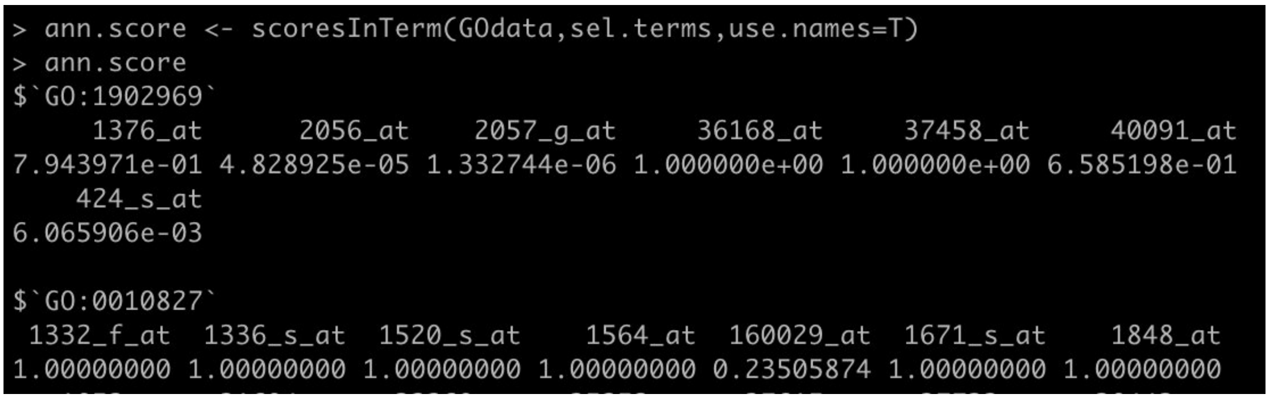
```
> num.ann.genes <- countGenesInTerm(GOdata,sel.terms)
> num.ann.genes
GO:1902969 GO:0010827 GO:0008645 GO:0016197 GO:0030902 GO:0001960 GO:1900745
         7         22         24         61         48         16         12
GO:0045581 GO:1901881 GO:2000104
        13          7          6
> ann.genes <- genesInTerm(GOdata,sel.terms)
> head(ann.genes)
$`GO:1902969`
[1] "1376_at"   "2056_at"   "2057_g_at" "36168_at"  "37458_at"  "40091_at"
[7] "424_s_at"

$`GO:0010827`
 [1] "1332_f_at"  "1336_s_at"  "1520_s_at"  "1564_at"    "160029_at"
 [6] "1671_s_at"  "1848_at"    "1852_at"    "31694_at"   "32260_at"
```

scoresInTerm函数获得对应score



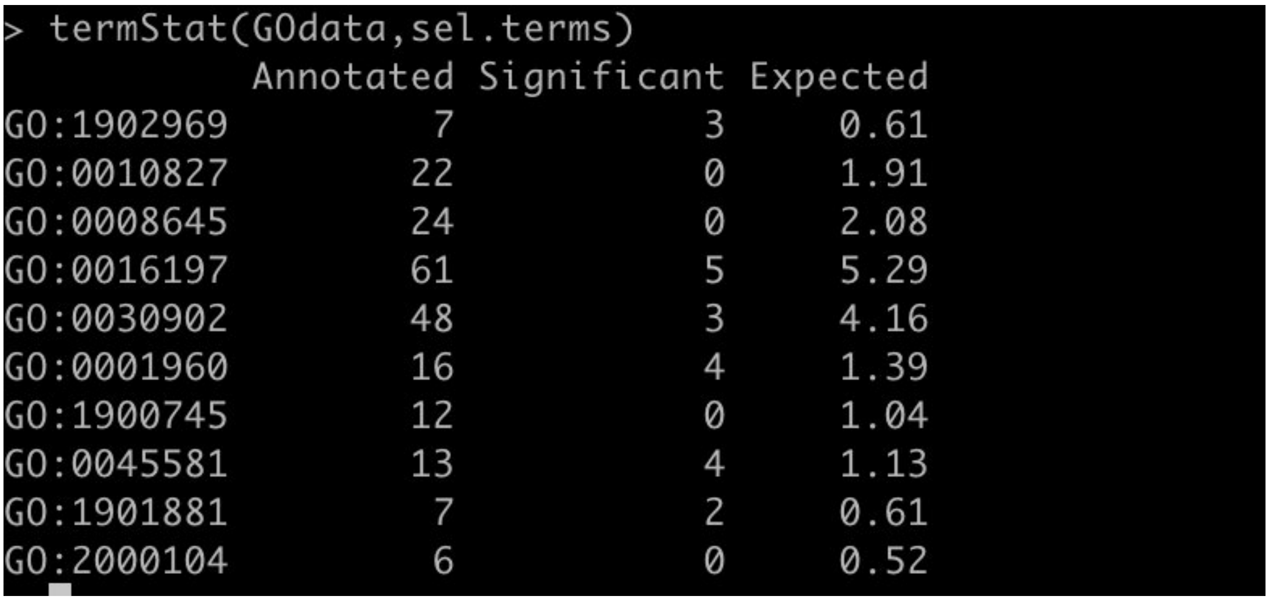
```
> ann.score <- scoresInTerm(GOdata,sel.terms)
> head(ann.score)
$`GO:1902969`
[1] 7.943971e-01 4.828925e-05 1.332744e-06 1.000000e+00 1.000000e+00
[6] 6.585198e-01 6.065906e-03

$`GO:0010827`
 [1] 1.00000000 1.00000000 1.00000000 1.00000000 0.23505874 1.00000000
 [7] 1.00000000 1.00000000 1.00000000 1.00000000 0.03096269 1.00000000
[13] 0.04280315 1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
[19] 1.00000000 1.00000000 1.00000000 1.00000000

$`GO:0008645`
 [1] 1.00000000 1.00000000 1.00000000 1.00000000 0.23505874 1.00000000
```

使用参数use.names获得gene名称



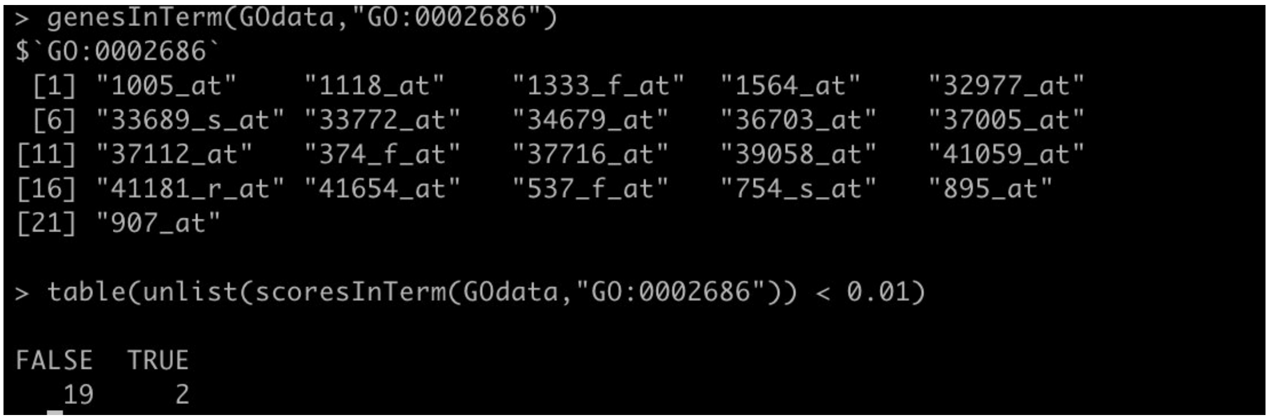
```
> ann.score <- scoresInTerm(GOdata,sel.terms,use.names=T)
> ann.score
$`GO:1902969`
     1376_at      2056_at    2057_g_at     36168_at     37458_at     40091_at
7.943971e-01 4.828925e-05 1.332744e-06 1.000000e+00 1.000000e+00 6.585198e-01
    424_s_at
6.065906e-03

$`GO:0010827`
 1332_f_at  1336_s_at  1520_s_at    1564_at  160029_at  1671_s_at    1848_at
1.00000000 1.00000000 1.00000000 1.00000000 0.23505874 1.00000000 1.00000000
```

termStat函数返回GO term的统计信息



```
> termStat(GOdata,sel.terms)
           Annotated Significant Expected
GO:1902969         7           3     0.61
GO:0010827        22           0     1.91
GO:0008645        24           0     2.08
GO:0016197        61           5     5.29
GO:0030902        48           3     4.16
GO:0001960        16           4     1.39
GO:1900745        12           0     1.04
GO:0045581        13           4     1.13
GO:1901881         7           2     0.61
GO:2000104         6           0     0.52
```

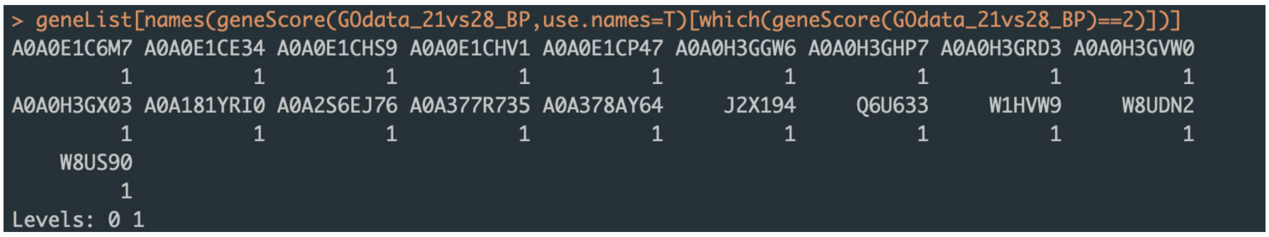
可通过其他函数——获得对应信息



```
> genesInTerm(GOdata,"GO:0002686")
$`GO:0002686`
 [1] "1005_at"    "1118_at"    "1333_f_at"  "1564_at"    "32977_at"  
 [6] "33689_s_at" "33772_at"   "34679_at"   "36703_at"   "37005_at"  
[11] "37112_at"   "374_f_at"   "37716_at"   "39058_at"   "41059_at"  
[16] "41181_r_at" "41654_at"   "537_f_at"   "754_s_at"   "895_at"    
[21] "907_at"    

> table(unlist(scoresInTerm(GOdata,"GO:0002686")) < 0.01)

FALSE  TRUE 
   19     2 
```

Excepted是理论情况下，存在n个显著基因。

若直接定义一组基因为interesting gene时，对应的geneScore(GOdata)为2，未选中基因为1(`geneList <- factor(as.integer(geneNames %in% myInterestingGenes))`)



```
> geneList[names(geneScore(GOdata_21vs28_BP,use.names=T)[which(geneScore(GOdata_21vs28_BP)==2)])]
A0A0E1C6M7 A0A0E1CE34 A0A0E1CHS9 A0A0E1CHV1 A0A0E1CP47 A0A0H3GGW6 A0A0H3GHP7 A0A0H3GRD3 A0A0H3GVW0 
         1          1          1          1          1          1          1          1          1 
A0A0H3GX03 A0A181YRI0 A0A2S6EJ76 A0A377R735 A0A378AY64     J2X194     Q6U633     W1HVW9     W8UDN2 
         1          1          1          1          1          1          1          1          1 
    W8US90 
         1 
Levels: 0 1
```

Running the enrichment tests

topGO package提供了多种统计检验和多种统计算法用于富集分析

- 基于gene counts，为最流行的统计家族，此时近需要输入一组感兴趣个体，无需其他信息，可采用Fisher's exact test，hypegeometric test和binomial test
- 基于gene scores或gene ranks，包含Kolmogorov-Smirnov like tests(又称为GSEA)，Gentleman's Category, t-test等
- 基于gene的expression，例如Goeman's global test或GlobalAncova, 直接作用于expression matrix

统计检验分类结构

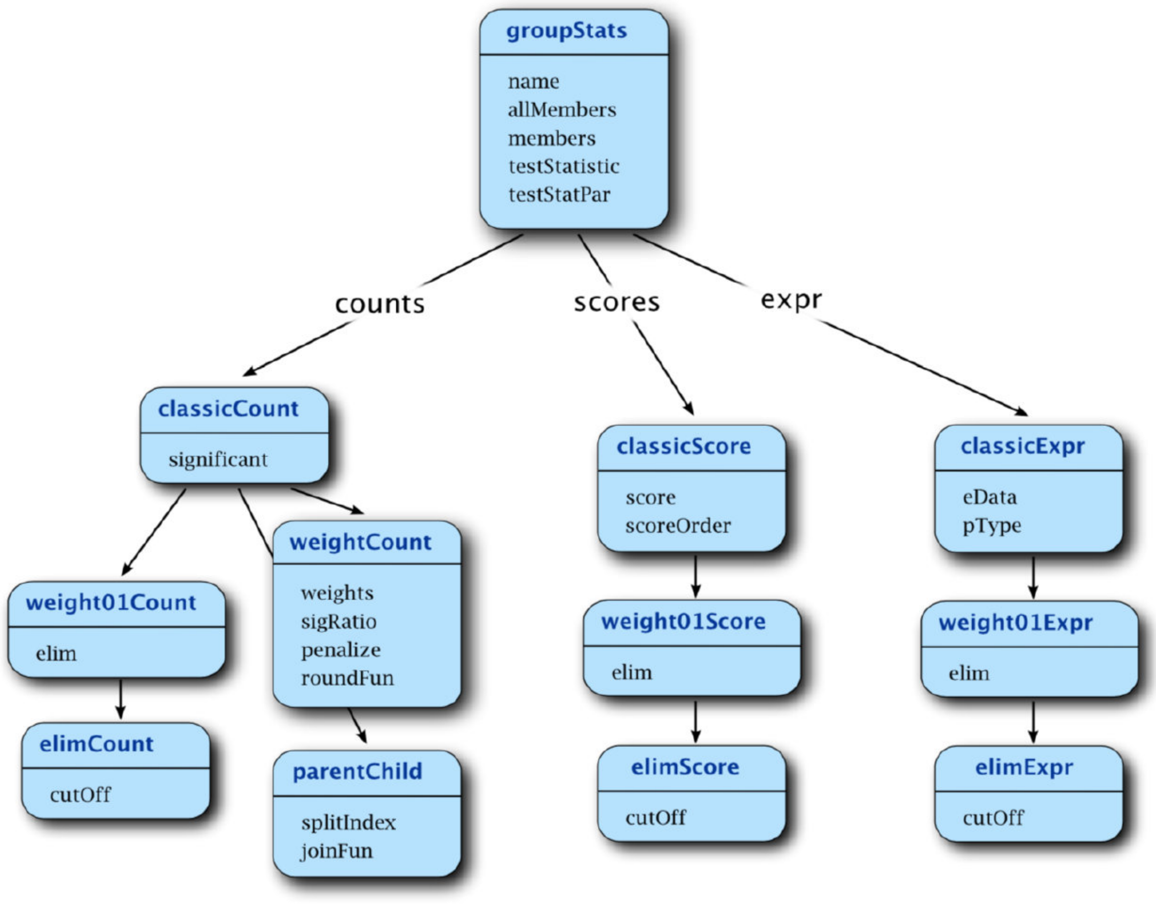


Figure 4: *The test statistics class structure.*

主要用于运行GO富集分析的函数为getSigGroups，该函数需要两个参数，一个为topGOdata对象，一个为groupStats class

- The groupStats classes，包含a gene set，指明如何进行统计检验

例如使用Fisher's exact test计算GO:0044255的富集过程，首先定义gene universe，同时获得GO:0044255所包含的genes，定义一组significant genes

```
goID <- "GO:0044255"
```

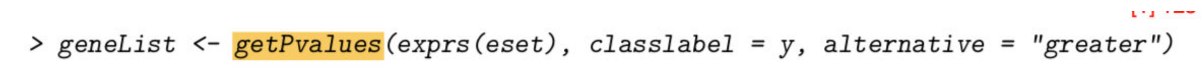
```
gene.universe <- genes(GOdata)
```

```
go.genes <- genesInTerm(GOdata, goID)[[1]]
```

`sig.genes <- sigGenes(GOdata)`

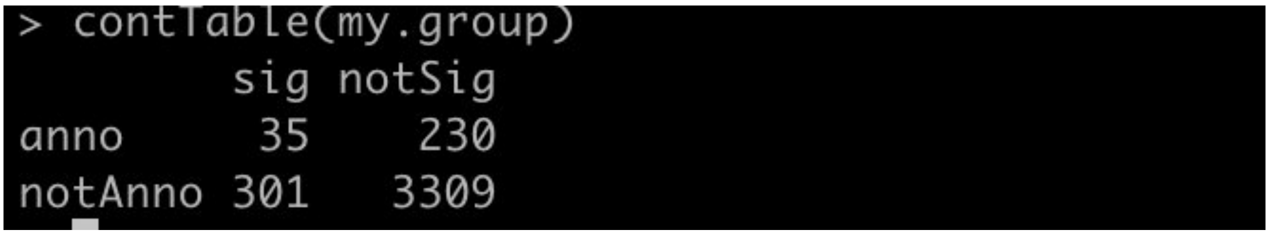
然后创建classicCount，就是一个2x2的列联表

```
my.group <- new("classicCout", testStatistic=GOFisherTest,
name="fisher",allMembers=gene.universe,
groupMemebers=go.genes,sigMembers=sig.genes)
```



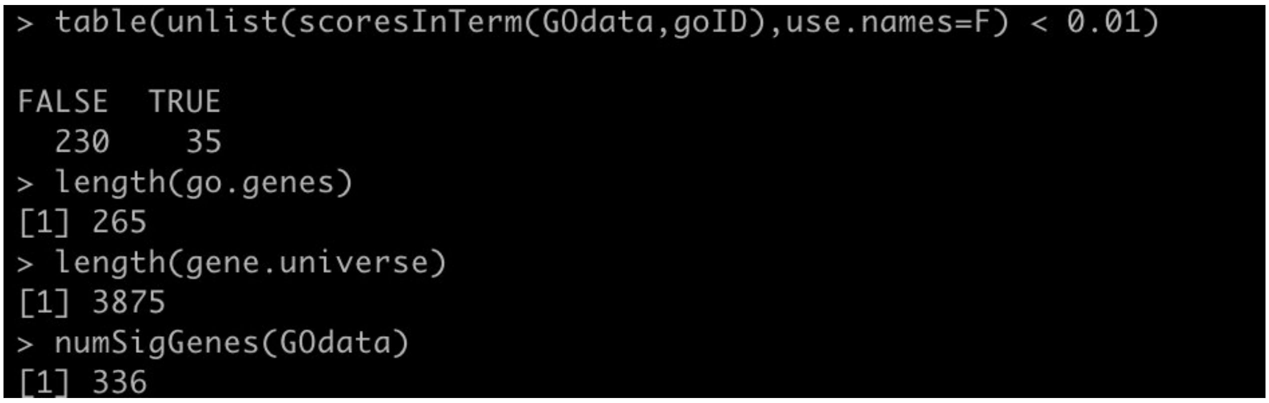
```
> geneList <- getPvalues(exprs(eset), classlabel = y, alternative = "greater")
```

contTable仅定义了根据gene count的分类，并用于根据对象构建的二维列联表



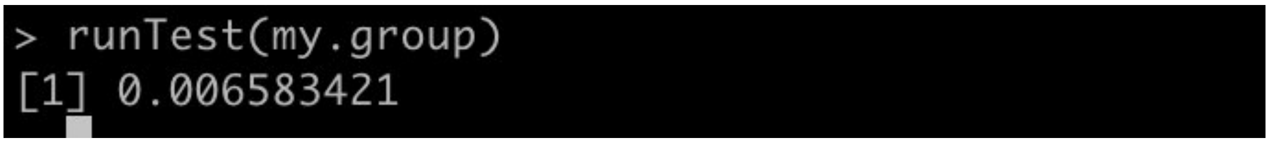
```
> contTable(my.group)
         sig notSig
anno      35    230
notAnno  301   3309
```

理解为抽样336个，其中有35个为anno



```
> table(unlist(scoresInTerm(GOdata,goID),use.names=F) < 0.01)

FALSE  TRUE
  230    35
> length(go.genes)
[1] 265
> length(gene.universe)
[1] 3875
> numSigGenes(GOdata)
[1] 336
```

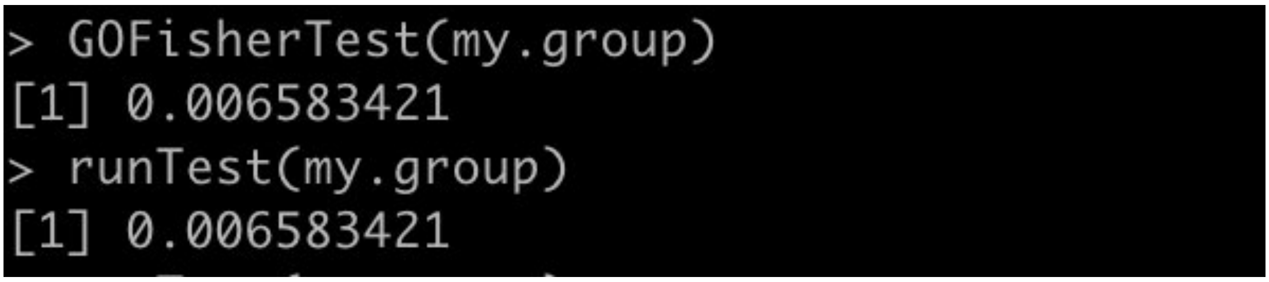
runTest根据groupStats已经定义好了统计检验方式，进行统计检验，返回值为GOFisherTest方式检出的Fisher's exact test p-value



```
> runTest(my.group)
[1] 0.006583421
```

testStatistic定义了test statistic function,包含：

GOFisherTest(object)，针对groupStats对象处理counts，基于列联表，运行Fisher's exact test，返回该检测p-value



```
> GOFisherTest(my.group)
[1] 0.006583421
> runTest(my.group)
[1] 0.006583421
```

~~**GOKSTest(object)，针对groupStats对象处理scores，运行Kolmogorov-Smirnov test，返回该检验p-value**~~

GOtTest(object)，针对groupStats对象处理Socres，运行t-test，当gene scores为t-statistics或服从正态分布，返回该检验p-value

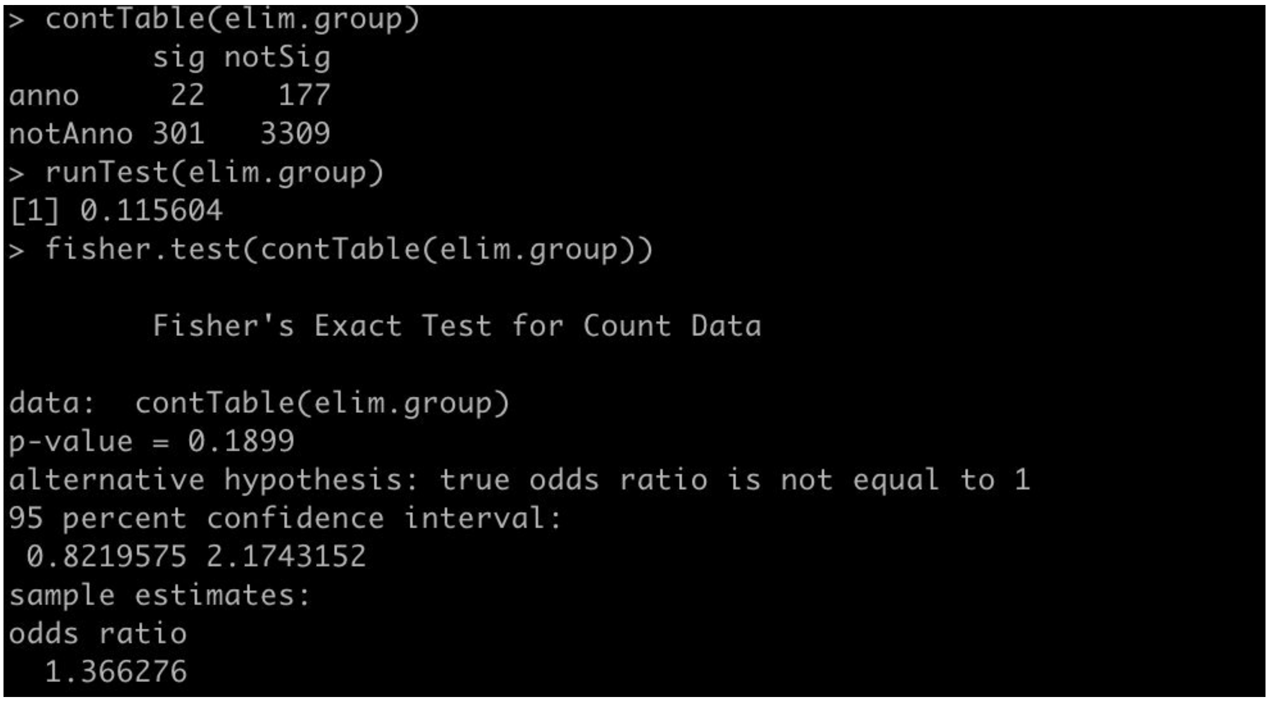
GOglobalTest, 采用Goeman's globaltest，返回该检验p-value

同样基于gene count，示例构建elimCount calss

`set.seed(123)`

`elim.genes <- sample(go.genes, length(go.genes)/4)`

`elim.group <- new("elimCount", testStatistic=GOFisherTest, name="fisher", allMembers=gene.universe, groupMembers=go.genes, sigMemebers=sig.genes, elim=elim.genes)`



```
> contTable(elim.group)
        sig notSig
anno     22    177
notAnno 301   3309
> runTest(elim.group)
[1] 0.115604
> fisher.test(contTable(elim.group))

        Fisher's Exact Test for Count Data

data:  contTable(elim.group)
p-value = 0.1899
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.8219575 2.1743152
sample estimates:
odds ratio
  1.366276
```

以上为两个groupStats class示例(my.group, elim.group)，它代表了一个gene set以及如何执行统计检验的信息！！！

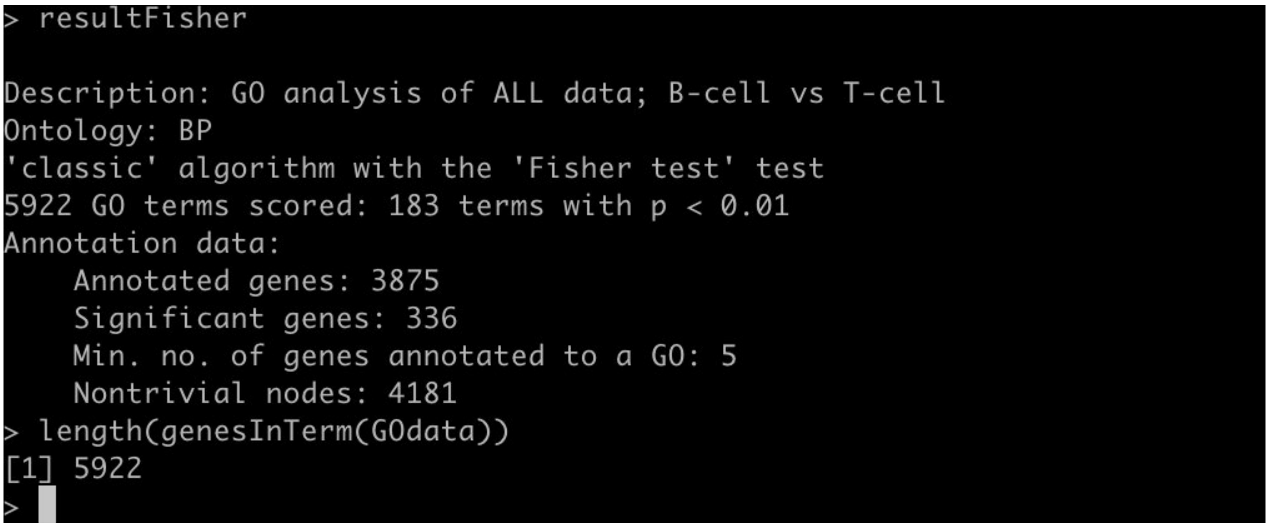
- Performing the test

参数testStatistic包含了统计函数，上面例子中的GOFisherTest就采用的是Fisher's exact test。用户可以定义自己的统计函数然后应用于classic算法中，例如计算Z scores。

首先定义统计方法；然后运行统计检验，**getSigGroups，针对一个topGOdata(包含所有用于检验的数据), 以及test.stat(定义了统计检验方法)，运行统计分析**

`test.stat <- new("classicCount", testStatistic=GOFisherTest, name="Fisher test")`

`resultFisher <- getSigGroups(GOdata, test.stat)`

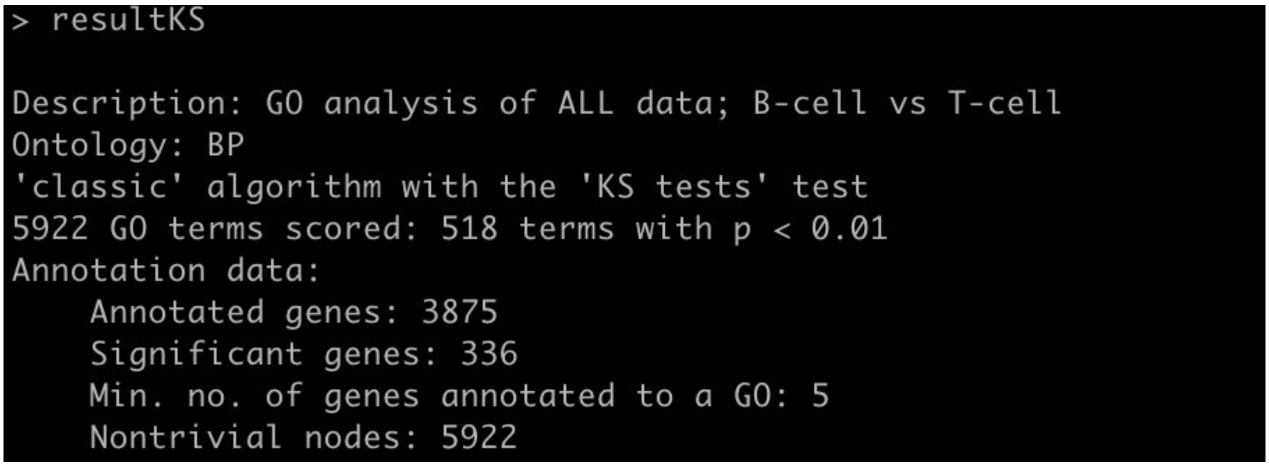


```
> resultFisher

Description: GO analysis of ALL data; B-cell vs T-cell
Ontology: BP
'classic' algorithm with the 'Fisher test' test
5922 GO terms scored: 183 terms with p < 0.01
Annotation data:
    Annotated genes: 3875
    Significant genes: 336
    Min. no. of genes annotated to a GO: 5
    Nontrivial nodes: 4181
> length(genesInTerm(GOdata))
[1] 5922
>
```

~~使用Kolmogorov-Smirnov test，需要提供gene-wise scoes~~

`test.stat <- new("classicScore", testStatistic=GOKSTest, name="KS tests")`

`resultKS <- getSigGroups(GOdata, test.stat)`



```
> resultKS

Description: GO analysis of ALL data; B-cell vs T-cell
Ontology: BP
'classic' algorithm with the 'KS tests' test
5922 GO terms scored: 518 terms with p < 0.01
Annotation data:
    Annotated genes: 3875
    Significant genes: 336
    Min. no. of genes annotated to a GO: 5
    Nontrivial nodes: 5922
```

~~同样KS检验运行elim算法~~

`test.stat <- new("elimScore", testStatistic=GOKSTest, name="Fisher test", cutOff=0.01)`

`resultElim <- getSigGroups(GOdata, test.stat)`

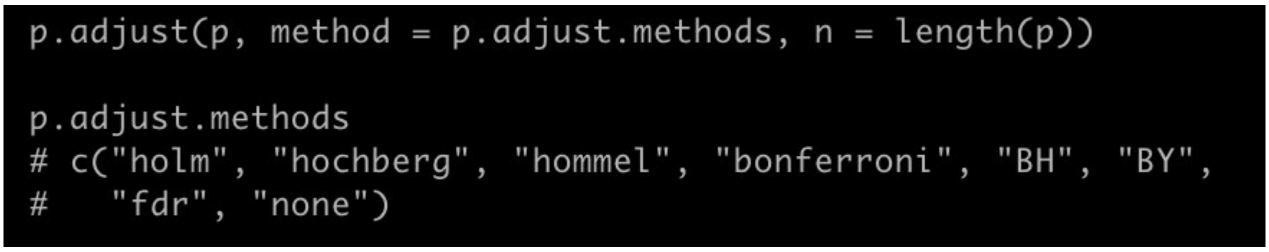
针对Fisher's exact test运行weight算法

`test.stat <- new("weightCount", testStatistic=GOFisherTest, name="Fisher test", sigRatio="ratio")`

`resultWeight <- getSigGroups(GOdata, test.stat)`

- The adjustment of p-values

getSigGroups函数返回的p-values为row p-values，这里没有多重检测矫正该值。可以自己做p-values矫正



```
p.adjust(p, method = p.adjust.methods, n = length(p))

p.adjust.methods
# c("holm", "hochberg", "hommel", "bonferroni", "BH", "BY",
#   "fdr", "none")
```

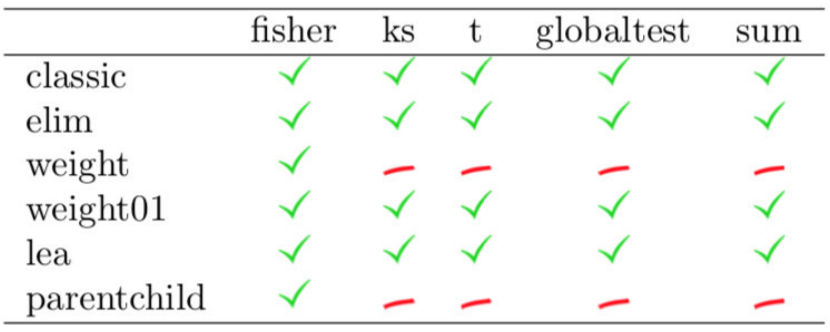
- runTest: a high-level interface of testing 推荐!!!

runTest函数仅能用于提前定义好的检验方法和算法(with a predefined set of test statistics and algorithm)，该函数有三个主要参数，topGOdata对象，algorithm，指定处理GO graph 结构的算法方式，statistic，指定统计算法

使用classic算法计算Fisher's exact test

`resultFis <- runTest(GOdata, algorithm="classic", statistic="fisher")`

多种算法可结合统计方式



	fisher	ks	t	globaltest	sum
classic	✓	✓	✓	✓	✓
elim	✓	✓	✓	✓	✓
weight	✓	–	–	–	–
weight01	✓	✓	✓	✓	✓
lea	✓	✓	✓	✓	✓
parentchild	✓	–	–	–	–

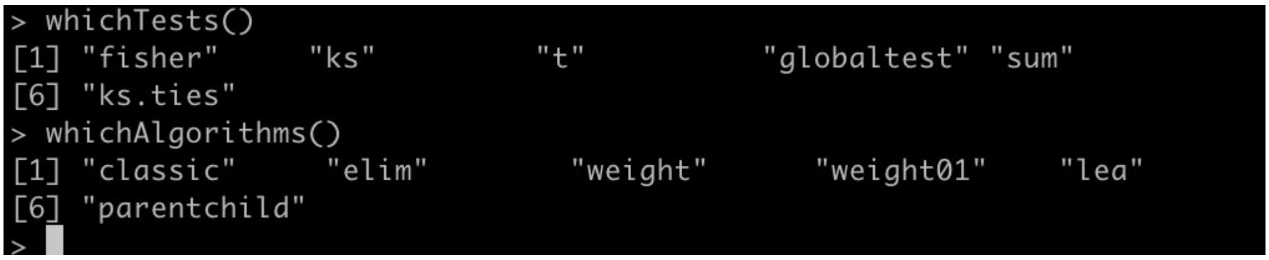
Table 1: *Algorithms currently supported by* topGO.

`weight01.fisher <- runTest(GOdata, algorithm="weight01", statistic="fisher")`

`weight01.t <- runTest(GOdata, algorithem="weight01", statistic="t")`

~~`elim.ks <- runTest(GOdata, algorithm="elim", statistic="ks")`~~

展示对应的算法和检验



```
> whichTests()
[1] "fisher"     "ks"          "t"           "globaltest" "sum"
[6] "ks.ties"
> whichAlgorithms()
[1] "classic"     "elim"         "weight"       "weight01"     "lea"
[6] "parentchild"
> 
```

runTest相比于getSigGroups函数，只是更友好，使用更清晰

结果解释及可视化

- The topGOresult object

getSigGroups和runTest均返回topGOresult对象

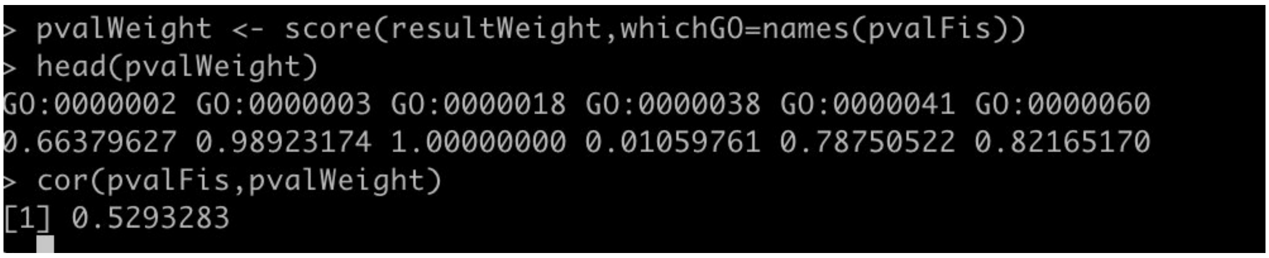
topGOresult对象结构简单，包含检验返回的p值或统计值(score)，以及test statistic和algorithm的基本信息。

score函数返回GO term的p-value，可以指定GO id，返回对应的p-values

`pvalFis <- score(resultFis)`

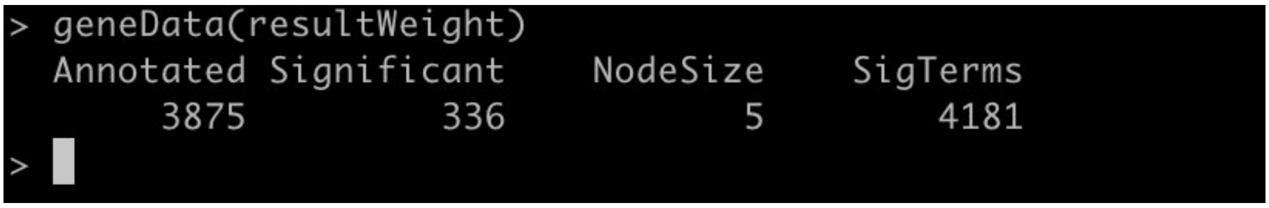
`hist(pvalFis, 50, xlab="p-values")`

`pvalWeight <- score(resultWeight, whichGO=names(pvalFis))`



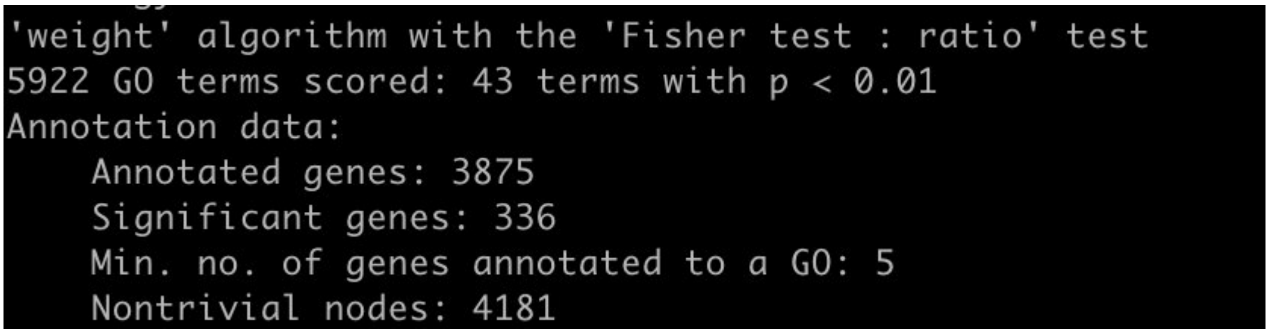
```
> pvalWeight <- score(resultWeight,whichGO=names(pvalFis))
> head(pvalWeight)
GO:0000002 GO:0000003 GO:0000018 GO:0000038 GO:0000041 GO:0000060
0.66379627 0.98923174 1.00000000 0.01059761 0.78750522 0.82165170
> cor(pvalFis,pvalWeight)
[1] 0.5293283
```

geneData函数返回topGOresult对象输入信息



```
> geneData(resultWeight)
  Annotated Significant    NodeSize    SigTerms
       3875         336           5        4181
> 
```

对应resultWeight信息



```
'weight' algorithm with the 'Fisher test : ratio' test
5922 GO terms scored: 43 terms with p < 0.01
Annotation data:
    Annotated genes: 3875
    Significant genes: 336
    Min. no. of genes annotated to a GO: 5
    Nontrivial nodes: 4181
```

自绘条形图，参考DESeq2，略

`colori <- c("resultFis"="khaki", "resultWeight"="powderblue")`

`h_Fis <- hist(pvalFis,plot=F)`

`h_Weight <- hist(pvalWeight, plot=F)`

`barplot(height=rbind(h_Fis$counts, h_Weight$counts), col=colori, space=0, ylab="p_value")`

`text(x=c(0, length(h_Fis$counts)), y =0, label=paste(c(0,1), adj=c(0.5,1.7), xpad=NA)`

`legend("topright", fill=rev(colori), legend=rev(names(colori)))`

- Summarising the results

GenTable函数返回topGOdata对象的表格信息

```
allRes <- GenTable(GOdata, classic=resultFis,KS=resultKS,weight=resultWeight,
orderBy="weight",ranksOf="classic",topNodes=20)
```

orderBy表示根据weight内的p-value顺序排序，rank显示在calssic中的顺序，topNodes表示显示的GO terms数目

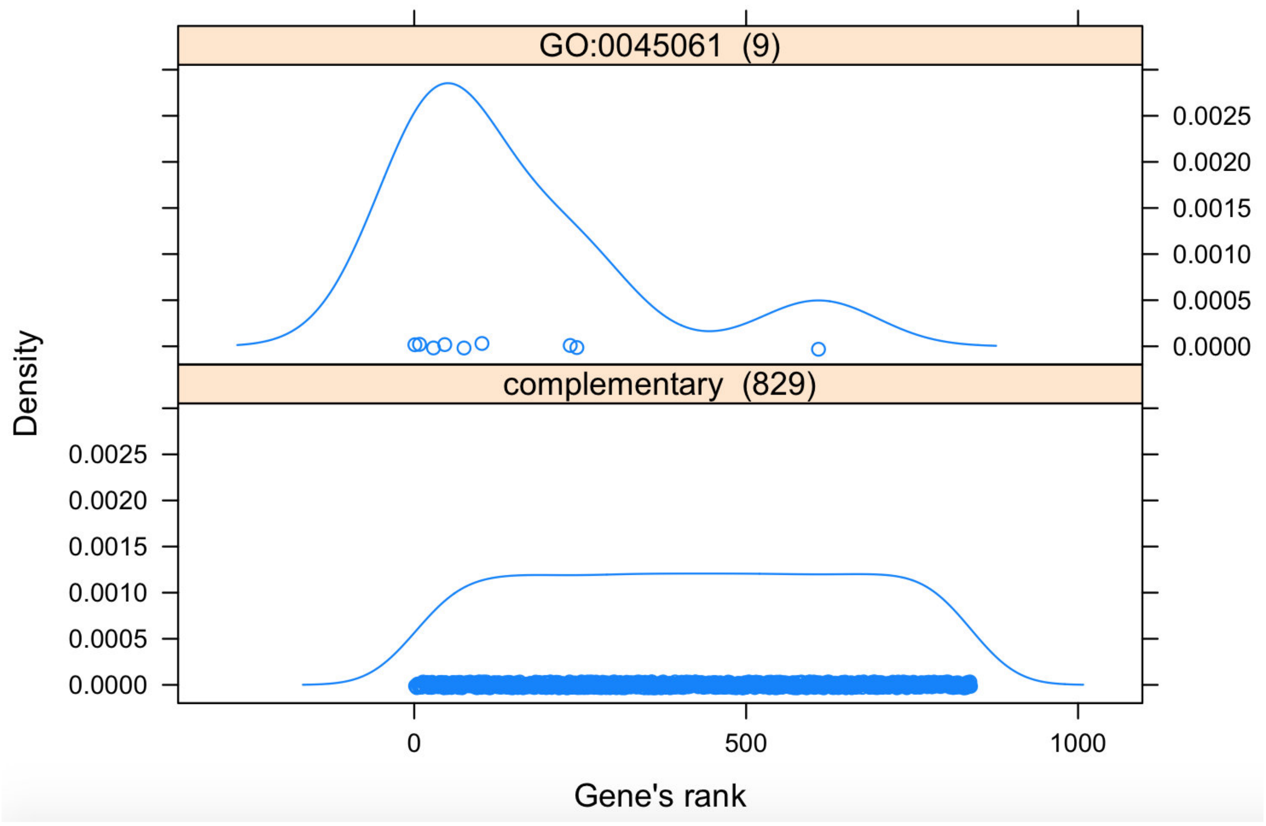
- Analysising individual GOs

查询感兴趣GOterm注释的gene分析，期待显著性富集GO term的注释gene具有更高的gene score相对于gene universe的平均gene score

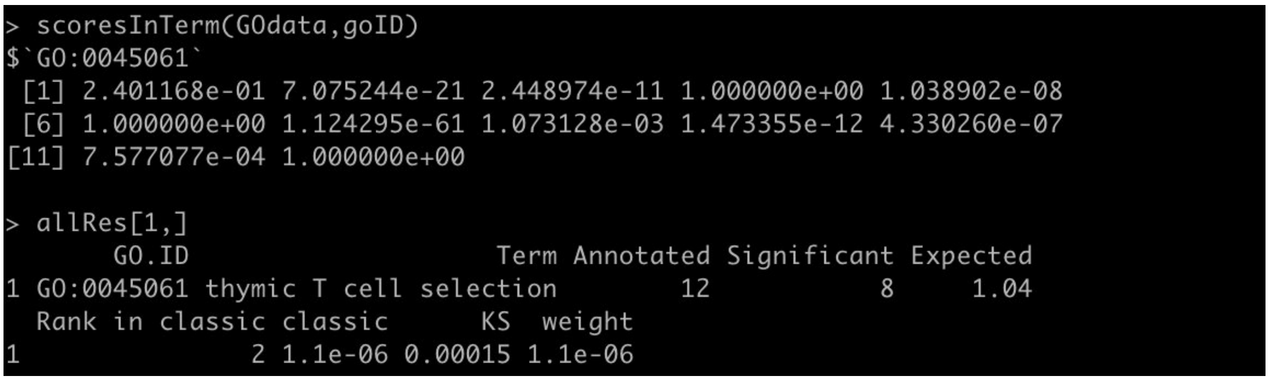
showGroupDensity函数绘制GO term内的gene score/rank的分布，使用ranks将会取代scores，rm.one=T，移除p-value为1的gene

```
goID <- allRes[1,"GO.ID"]
```

```
print(showGroupDensity(GOdata, goID, ranks=T))
```

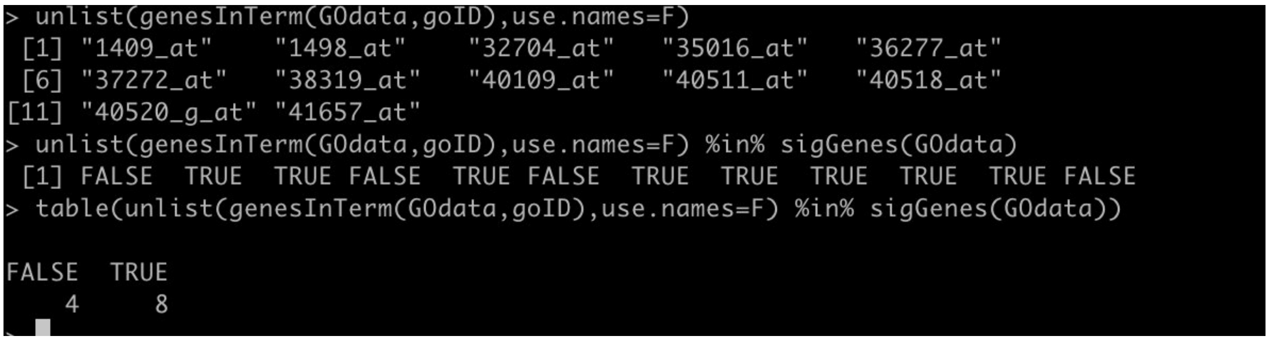


对应获得该GO term的score信息，rm.one默认为T，因此去掉3个为1的gene，剩余9个点；complementary点对应为其他未注释到该GO term上的genes的p值分布



```
> scoresInTerm(GOdata,goID)
$`GO:0045061`
 [1] 2.401168e-01 7.075244e-21 2.448974e-11 1.000000e+00 1.038902e-08
 [6] 1.000000e+00 1.124295e-61 1.073128e-03 1.473355e-12 4.330260e-07
[11] 7.577077e-04 1.000000e+00

> allRes[1,]
       GO.ID                   Term Annotated Significant Expected
1 GO:0045061 thymic T cell selection        12           8     1.04
  Rank in classic classic      KS  weight
1               2 1.1e-06 0.00015 1.1e-06
```

对应significatn genes为8，满足p-value <0.01

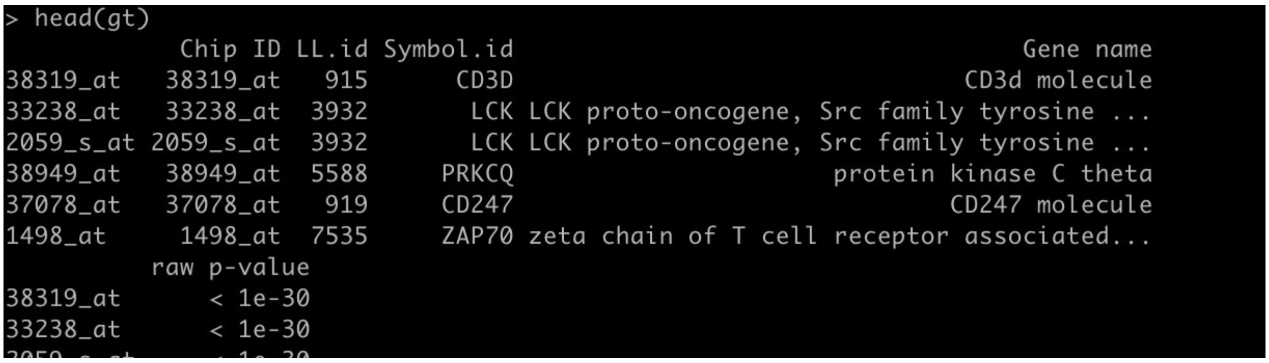


```
> unlist(genesInTerm(GOdata,goID),use.names=F)
 [1] "1409_at"    "1498_at"    "32704_at"   "35016_at"   "36277_at"
 [6] "37272_at"   "38319_at"   "40109_at"   "40511_at"   "40518_at"
[11] "40520_g_at" "41657_at"
> unlist(genesInTerm(GOdata,goID),use.names=F) %in% sigGenes(GOdata)
 [1] FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
> table(unlist(genesInTerm(GOdata,goID),use.names=F) %in% sigGenes(GOdata))

FALSE  TRUE
    4     8
```

printGenes函数对应打印映射到指定GO term的gene/probe信息

`goID <- allRes[10, "GO.ID"]`

`gt <- printGenes(GOdata, whichTerm=goID, chip=affyLib, numChar=40)`



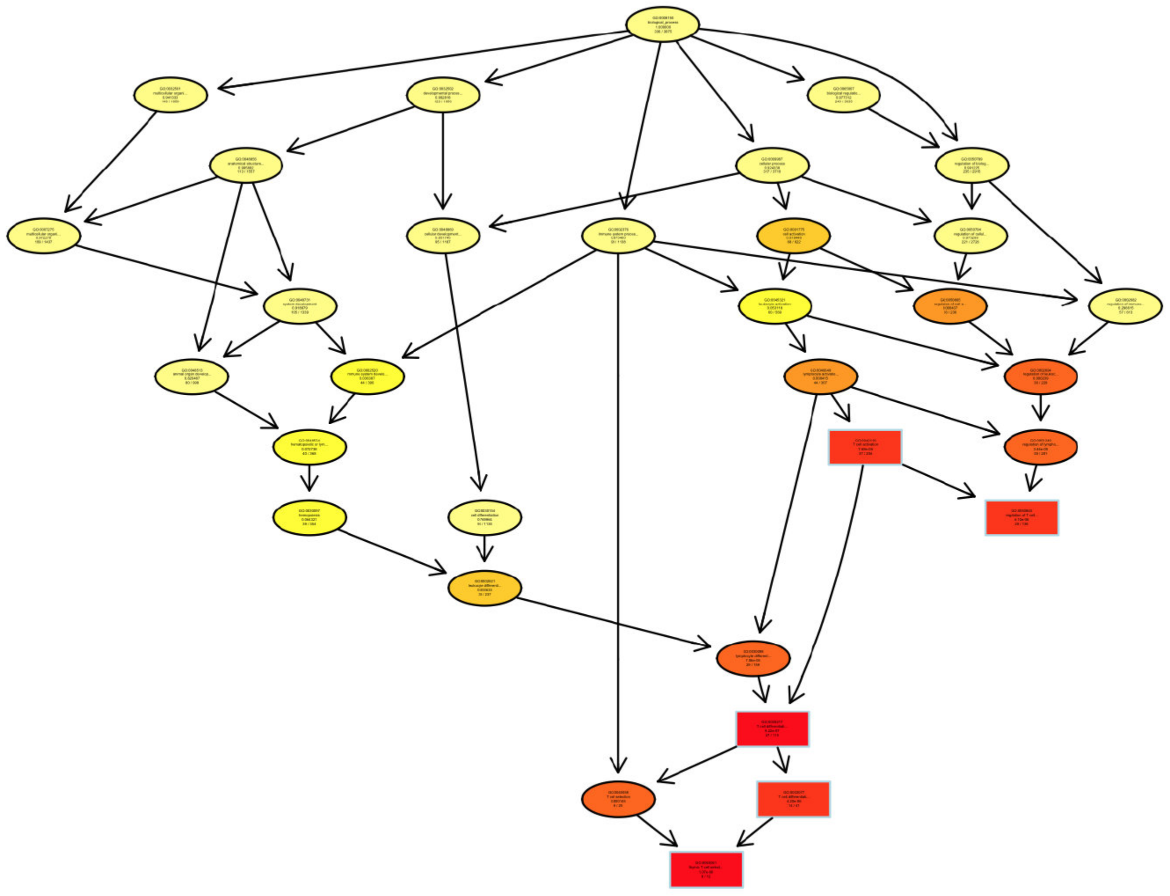
```
> head(gt)
            Chip ID LL.id Symbol.id                                    Gene name
38319_at   38319_at   915      CD3D                                 CD3d molecule
33238_at   33238_at  3932       LCK LCK proto-oncogene, Src family tyrosine ...
2059_s_at 2059_s_at  3932       LCK LCK proto-oncogene, Src family tyrosine ...
38949_at   38949_at  5588     PRKCQ                        protein kinase C theta
37078_at   37078_at   919     CD247                                CD247 molecule
1498_at     1498_at  7535     ZAP70 zeta chain of T cell receptor associated...
          raw p-value
38319_at      < 1e-30
33238_at      < 1e-30
```

- Visualising the GO structure

展示GO graph显著性GO term，showSigOfNodes展示subgraph，printGraph将showSigOfNodes保存本地

`showSigOfNodes(GOdata, score(resultFis), firstSigNodes=5, useInfo="all")`

`showSigOfNodes(GOdata, score(resultWeight), firstSigNodes=5, useInfo="all")`



```
printGraph(GOdata, resultFis, firstSigNodes = 5, fn.prefix = "tGO", useInfo = "all", pdfSW = TRUE)
```

```
printGraph(GOdata, resultWeight, firstSigNodes=5, fn.prefix="tGO", useInfo="all", pdfSW=T)
```

图中significant nodes为方形，颜色表示相对显著性，由深到浅显著性下降，黑色箭头表示is-a-relationships，红色箭头表示part-of relationships

了解不同的富集方式以及理解哪些显著性GO term是感兴趣的非常重要

使用printGraph函数强调两种方式差异

```
printGraph(GOdata, resultWeight, firstSigNodes=10, resultFis, fn.prefix="tGO", useInfo="def")
```

```
printGraph(GOdata, resultElim, firstSigNodes=15, resultFis, fn.prefix="tGO",useInfo="all")
```

示列

- Quick start guide

1. Data preparation: gene id, gene scores, differentially expressed genes, selected genes based on their scores, gene-to-GO annotations
2. Running the enrichment tests: statistic method and algorithm

3. Analysis of the results: summarize and visualize the results

- Data preparation

`library(toGO)`

`library(ALL)`

`data(ALL)`

`data(geneList)`

`affyLib <- paste(annotation(ALL), "db", sep=".")`

`library(package=affyLib, character.only=TRUE)`

`sum(topDiffGenes(geneList))`

topDiffGenes，选出显著性小于0.01的gene

创建topGOdata

`sampleGOdata <- new("topGOdata", description="Simple session", ontology="BP", allGenes=geneList, genesel=topDiffGenes, nodeSize=10, annot=annFUN.db, affyLib=affyLib)`

- Performing the enrichment tests

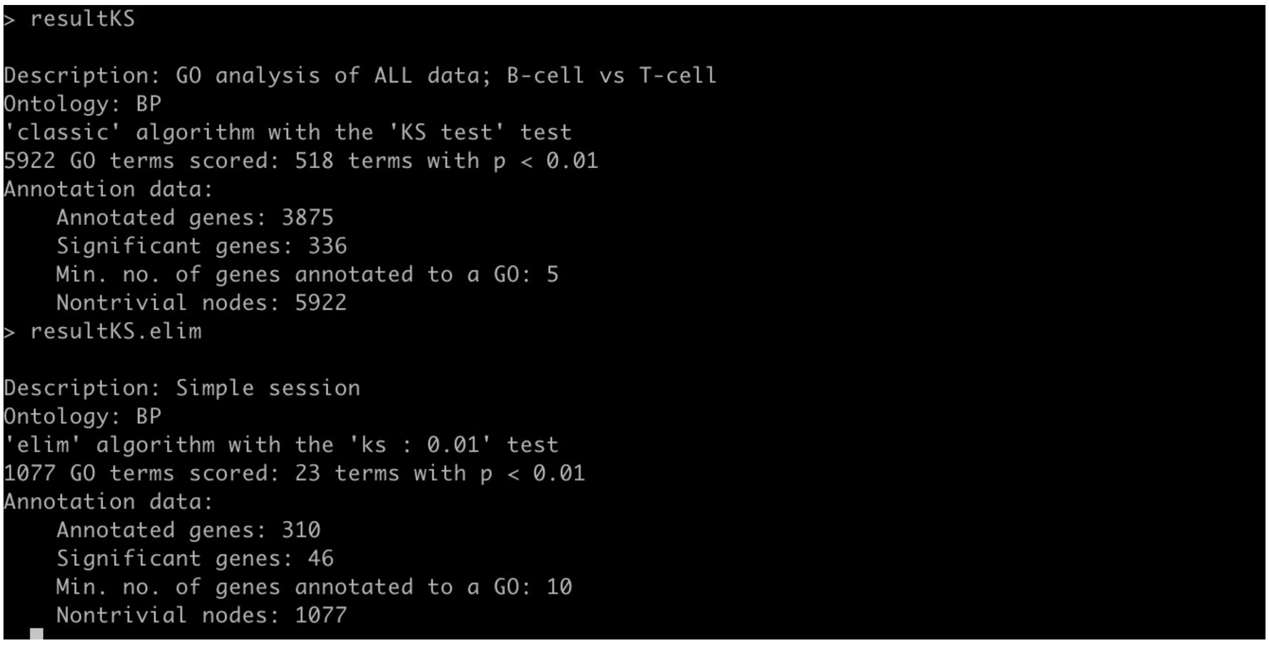
Fisher's exact test是基于gene counts，genes分类为差异表达和非差异表达； Kolmogorov-Smirnov like test是基于gene scores(GSEA)，gene scores代表表达gene的差异程度。runTest函数指定特殊的统计检验类型用于数据。

`resultFisher <- runTest(sampleGOdata, algorithm="classic", statistic="fisher")`

runTest返回对象topGOresult，同时使用Kolmogorov-Smirnov test检验富集

`resultKS <- runTest(sampleGOdata, algorithm="classic",statistic="ks")`

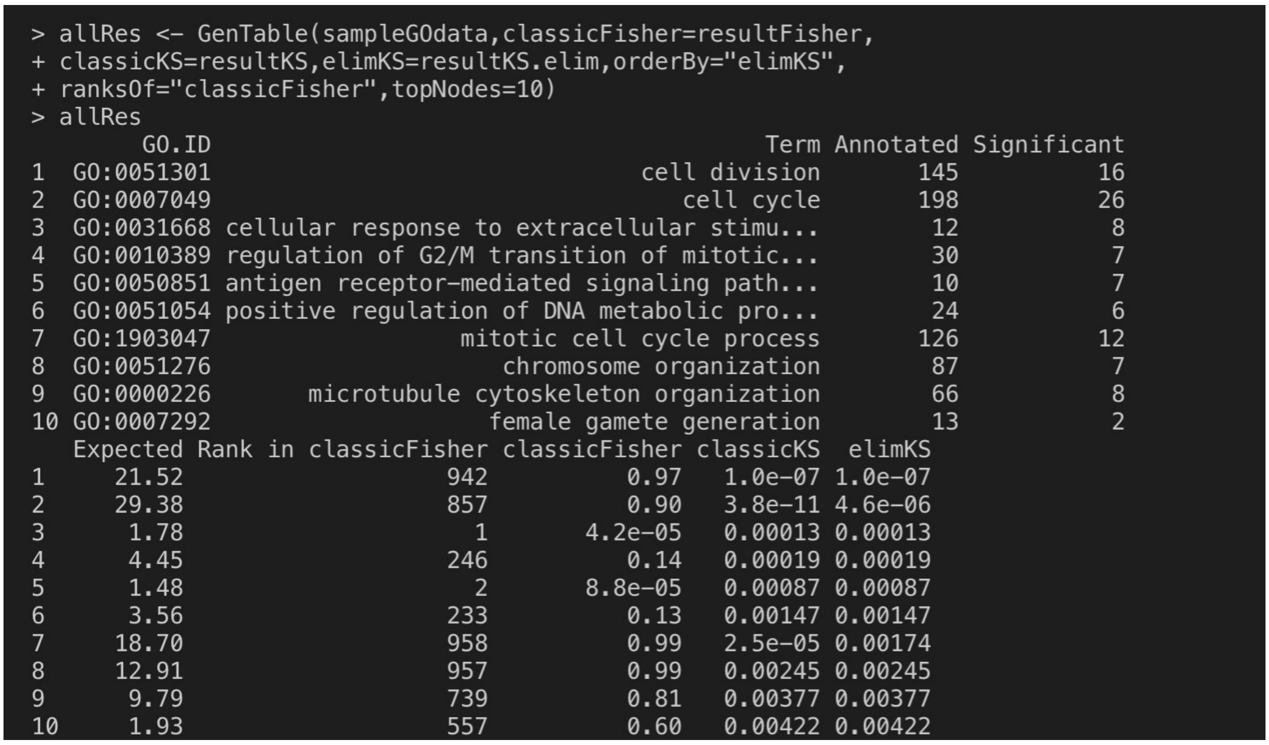
`resultKS.elim <- runTest(sampleGOdata, algorithm="elim", statistic="ks")`


```
> resultKS

Description: GO analysis of ALL data; B-cell vs T-cell
Ontology: BP
'classic' algorithm with the 'KS test' test
5922 GO terms scored: 518 terms with p < 0.01
Annotation data:
    Annotated genes: 3875
    Significant genes: 336
    Min. no. of genes annotated to a GO: 5
    Nontrivial nodes: 5922
> resultKS.elim

Description: Simple session
Ontology: BP
'elim' algorithm with the 'ks : 0.01' test
1077 GO terms scored: 23 terms with p < 0.01
Annotation data:
    Annotated genes: 310
    Significant genes: 46
    Min. no. of genes annotated to a GO: 10
    Nontrivial nodes: 1077
```

这里返回的p-value示未经过矫正过的多重检验值

- Analysis of results

`allRes <- GenTable(sampleGOdata, calssicFisher=resultFisher, classicKS=resultKS, elimKS=resultKS.elim, orderBy="elimKS", ranksOf="classicFisher", topNodes=10)`



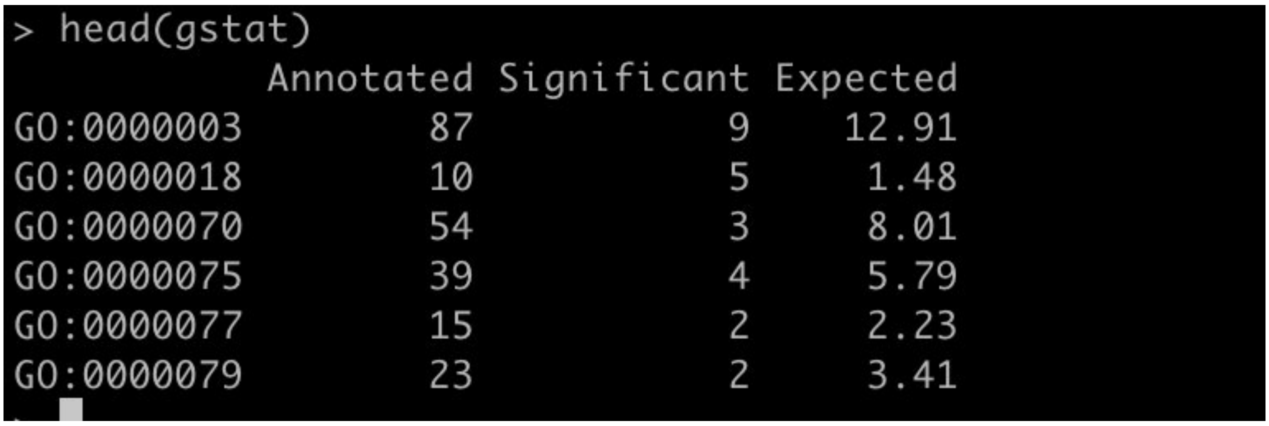
```
> allRes <- GenTable(sampleGOdata,classicFisher=resultFisher,
+ classicKS=resultKS,elimKS=resultKS.elim,orderBy="elimKS",
+ ranksOf="classicFisher",topNodes=10)
> allRes
        GO.ID                                        Term Annotated Significant
1  GO:0051301                               cell division       145          16
2  GO:0007049                                  cell cycle       198          26
3  GO:0031668 cellular response to extracellular stimu...        12           8
4  GO:0010389 regulation of G2/M transition of mitotic...        30           7
5  GO:0050851 antigen receptor-mediated signaling path...        10           7
6  GO:0051054 positive regulation of DNA metabolic pro...        24           6
7  GO:1903047                  mitotic cell cycle process       126          12
8  GO:0051276                     chromosome organization        87           7
9  GO:0000226       microtubule cytoskeleton organization        66           8
10 GO:0007292                    female gamete generation        13           2
   Expected Rank in classicFisher classicFisher classicKS  elimKS
1     21.52                   942          0.97   1.0e-07 1.0e-07
2     29.38                   857          0.90   3.8e-11 4.6e-06
3      1.78                     1       4.2e-05   0.00013 0.00013
4      4.45                   246          0.14   0.00019 0.00019
5      1.48                     2       8.8e-05   0.00087 0.00087
6      3.56                   233          0.13   0.00147 0.00147
7     18.70                   958          0.99   2.5e-05 0.00174
8     12.91                   957          0.99   0.00245 0.00245
9      9.79                   739          0.81   0.00377 0.00377
10     1.93                   557          0.60   0.00422 0.00422
```

score函数返回topGOresult对象的p-values，查看classic和elim方式返回值的差异，elim方式相对于classic方式会更保守，~~GO term根据"elimKS"返回p值排序，秩序值是该GOterm在classicFisher中排序~~

`pValue.classic <- score(resultKS)`

`pValue.elim <- score(resultKS.elim)[names(pValue.classic)]`

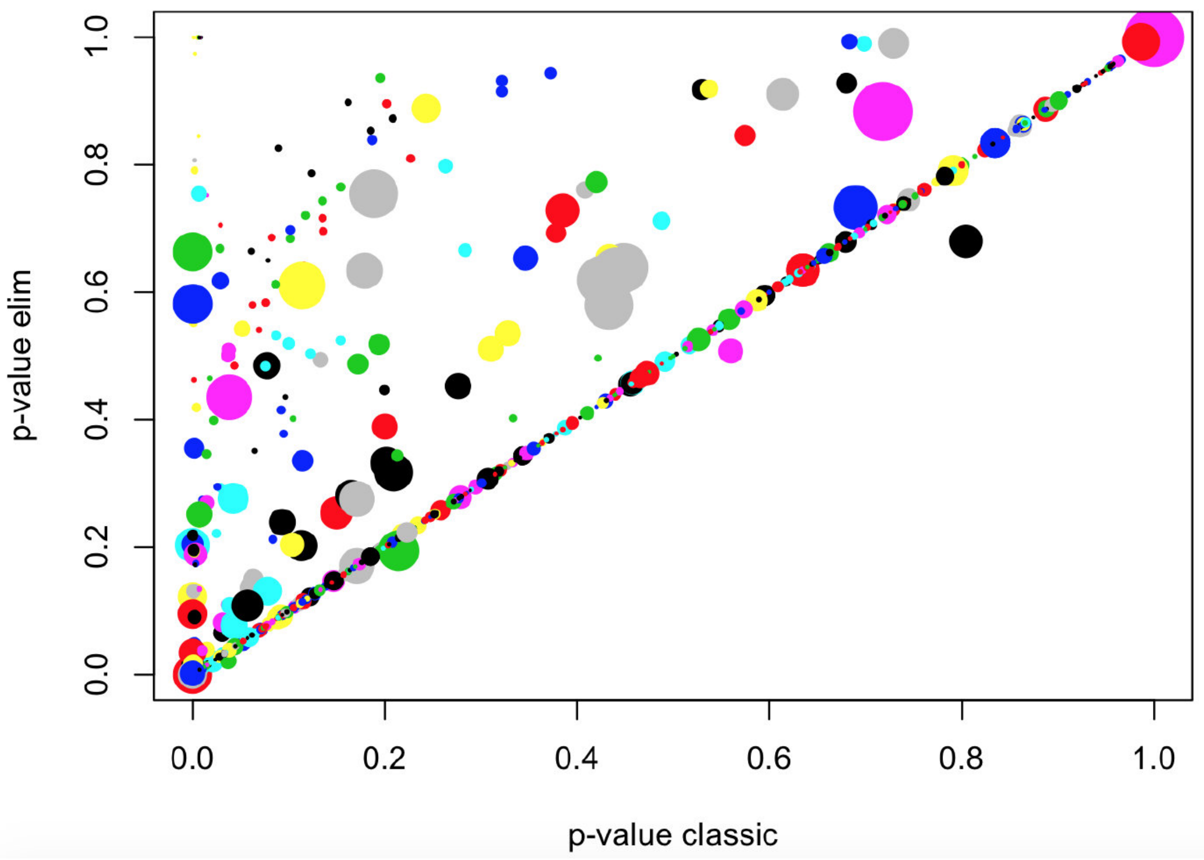
`gstat <- termStat(sampleGOdata, names(pValue.classic))`



```
> head(gstat)
           Annotated Significant Expected
GO:0000003        87           9    12.91
GO:0000018        10           5     1.48
GO:0000070        54           3     8.01
GO:0000075        39           4     5.79
GO:0000077        15           2     2.23
GO:0000079        23           2     3.41
```

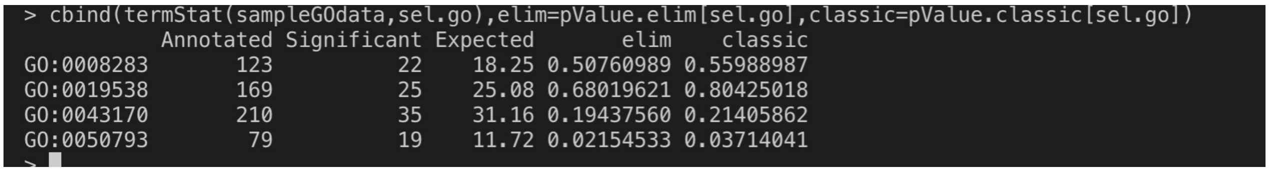
`gSize <- gstat$Annotated/max(gstat$Annotated)*4`

`plot(pValue.classic , pValue.elim, xlab="p-value classic", ylab="p-value elim", pch=19, cex=gSize,)`



可见elim返回的p值比classic更保守，同时也有一些GO term由classic返回相比elim保守，查看这些信息

```
sel.go <-names(pValue.classic)[pValue.elim < pValue.classic]
```

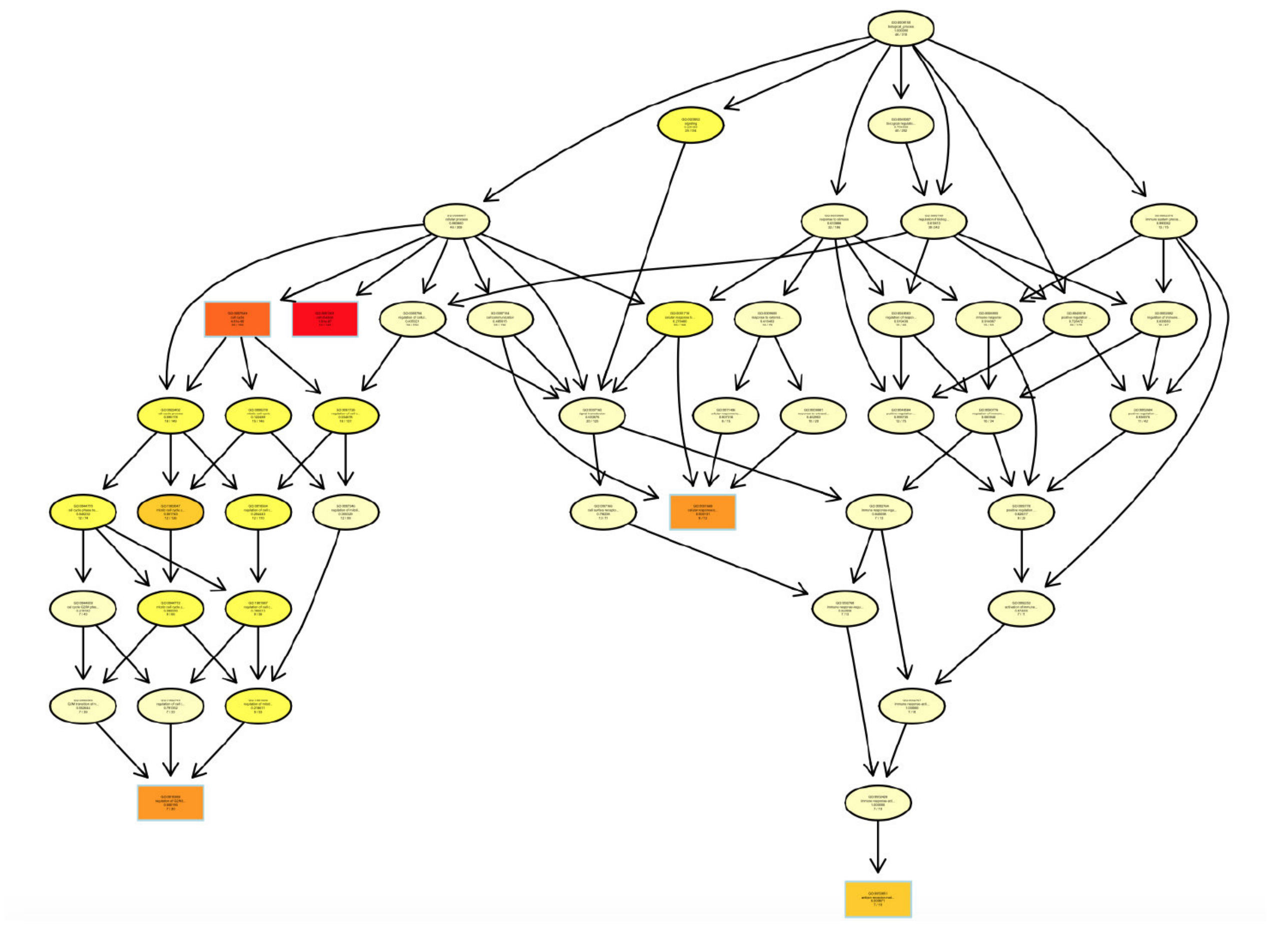


```
> cbind(termStat(sampleGOdata,sel.go),elim=pValue.elim[sel.go],classic=pValue.classic[sel.go])
           Annotated Significant Expected       elim    classic
GO:0008283       123          22    18.25 0.50760989 0.55988987
GO:0019538       169          25    25.08 0.68019621 0.80425018
GO:0043170       210          35    31.16 0.19437560 0.21405862
GO:0050793        79          19    11.72 0.02154533 0.03714041
>
```

可见这个4个GO terms的p-value不够显著，同时elim和classic相差无几

展示显著性nodes图

```
showSigOfNodes(sampleGOdata, score(resultKS.elim), firstSigNodes=5,
useInfo="all")
```



椭圆或者方框内信息为：第一行，GO ID；第二行，GO名称，第三行，初始p-value；第四行，该GO term显著性/总genes。颜色从深到亮黄，依次表示显著性降低。

---

emapper 结果GO分析（错误,目前无法找到predicted genes 对应的 EntrezID信息，同时需加载AnnotationHub对应sqlite文件注释信息）

使用org.EcK12.eg.db包注释

1. 从emapper的38588_prodigal_emapper_output.emapper.annotations中挑选具有GO terms的行，同时去除重复的行，获得predicted gene name 对应GO terms信息
2. R读取该信息，加载topGO和模式生物包org.EcK12.eg.db

`library(topGO)`

`library(org.Eck12.eg.db)` ##使用该org进行测试分析

`ecoli_db <- org.Eck12.eg.db`

`ecoli_allgens <- keys(ecoli_db)`

`genes_entrezid < select(ecoli_db, keys=gos, keytype="GO", columns=c("ENTREZID"))` ##gos为emapper获得的所有GO terms

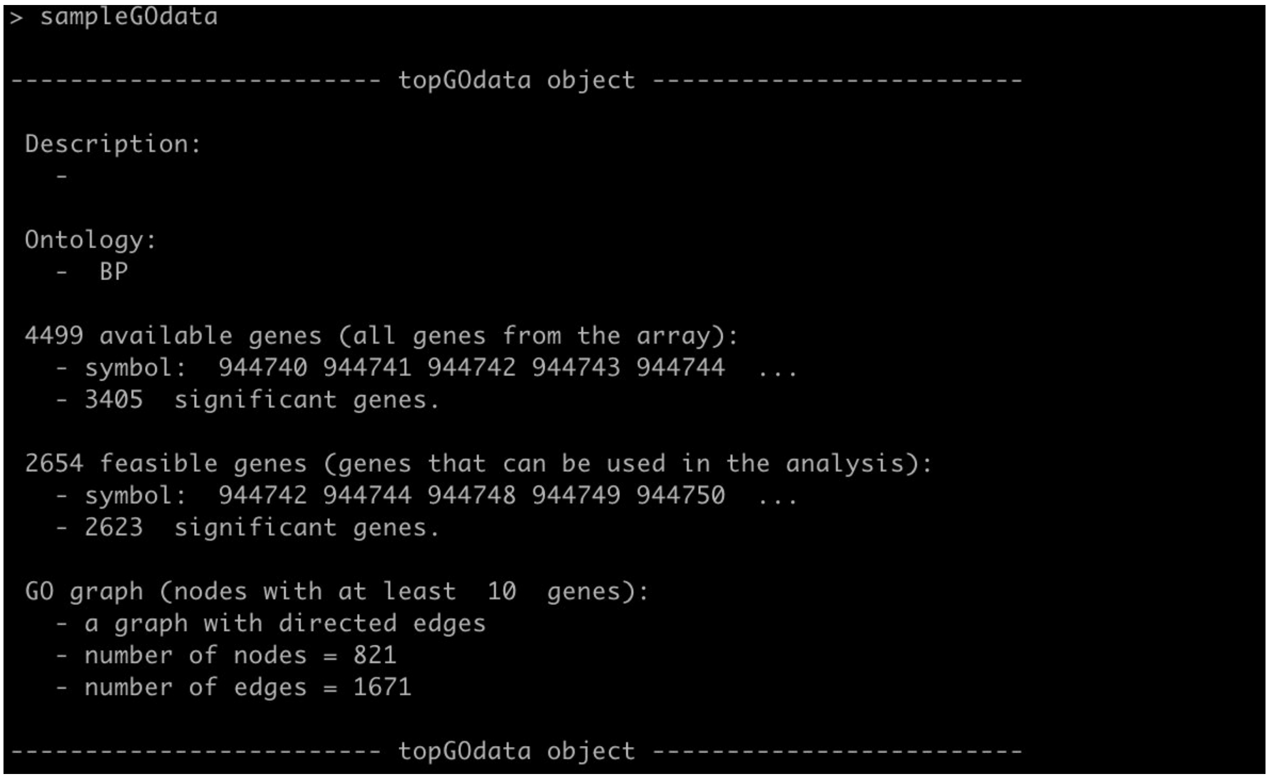
构建geneList向量

`geneList <- factor(as.integer(ecoli_allgenes %in% genes_entrezid))`

`names(geneList) <- ecoli_allgenes`

构建GOdata

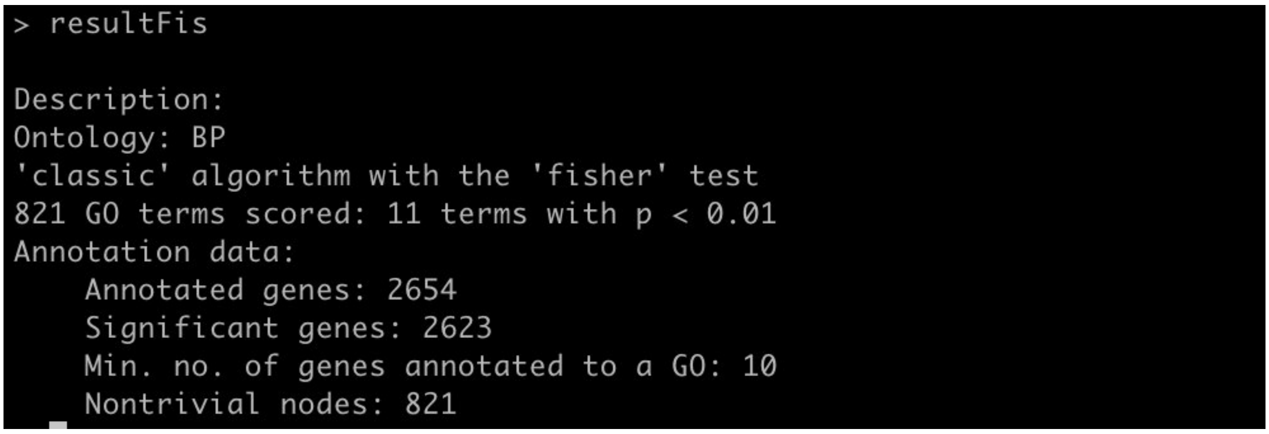
```
sampleGOdata <-
new("topGOdata",ontology="BP",allGenes=geneList,nodeSize=10,annot=annFUN.org,
mapping="org.EcK12.eg.db",ID="entrez")
```



```
> sampleGOdata

------------------------- topGOdata object -------------------------

 Description:
   -

 Ontology:
   -  BP

 4499 available genes (all genes from the array):
   - symbol:  944740 944741 944742 944743 944744  ...
   - 3405  significant genes.

 2654 feasible genes (genes that can be used in the analysis):
   - symbol:  944742 944744 944748 944749 944750  ...
   - 2623  significant genes.

 GO graph (nodes with at least  10  genes):
   - a graph with directed edges
   - number of nodes = 821
   - number of edges = 1671

------------------------- topGOdata object -------------------------
```

使用classic算法计算Fisher's exact test

```
resultFis <- runTest(sampleGOdata, algorithm="classic",statistic="fisher")
```

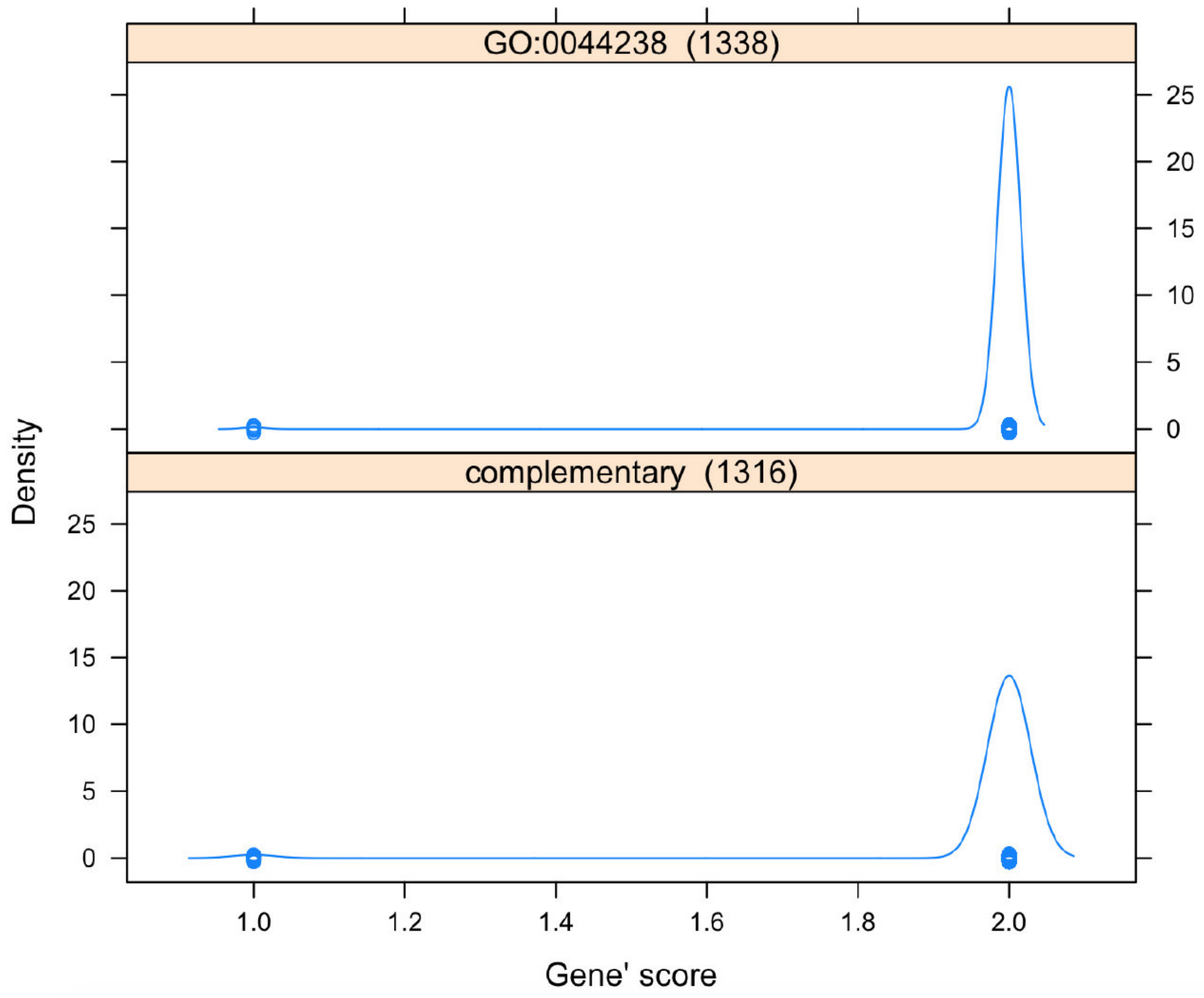


```
> resultFis

Description:
Ontology: BP
'classic' algorithm with the 'fisher' test
821 GO terms scored: 11 terms with p < 0.01
Annotation data:
    Annotated genes: 2654
    Significant genes: 2623
    Min. no. of genes annotated to a GO: 10
    Nontrivial nodes: 821
```

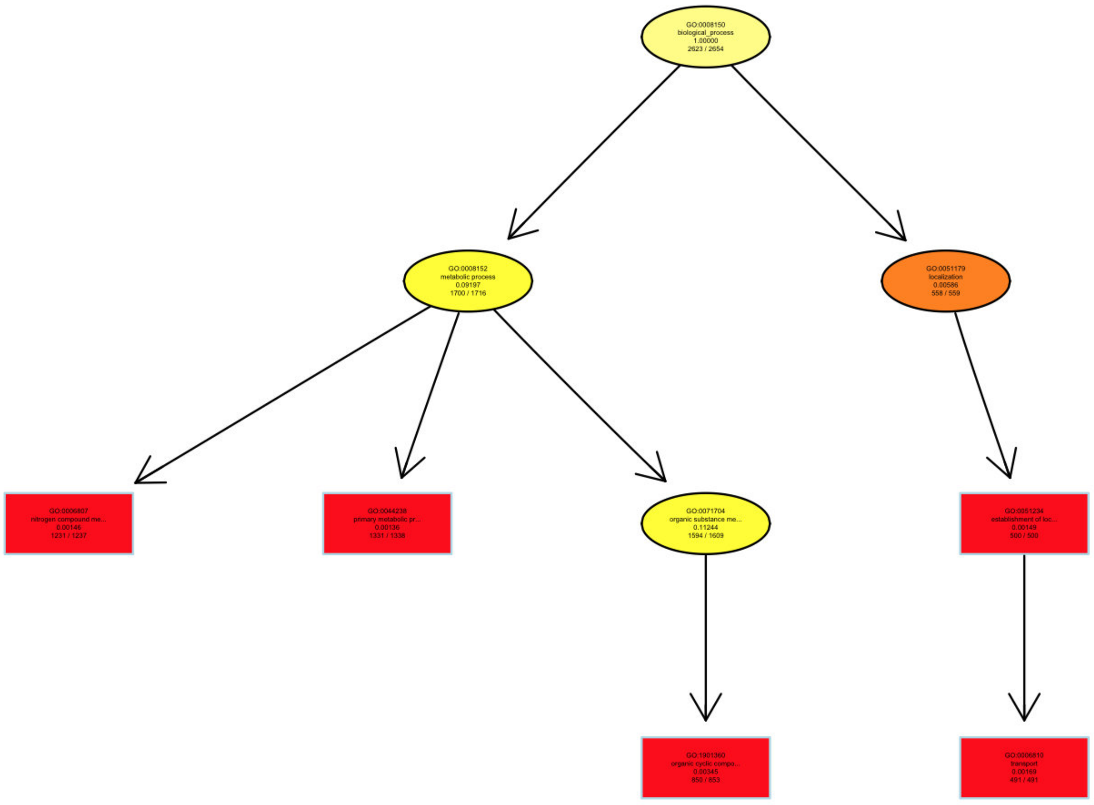
图示

由于数据虚假

```
showGroupDensity(sampleGOdata,whichGO=goID,rm.one=F)
```



```
showSigOfNodes(sampleGOdata,score(resultFis),firstSigNodes=5,useInfo="all")
```


GO:0008150
biological_process
1.00000
2623 / 2654
GO:0008152
metabolic process
0.09197
1700 / 1716
GO:0051179
localization
0.00586
558 / 559
GO:0006807
nitrogen compound me...
0.00146
1231 / 1237
GO:0044238
primary metabolic pr...
0.00136
1331 / 1338
GO:0071704
organic substance me...
0.11244
1594 / 1609
GO:0051234
establishment of loc...
0.00149
500 / 500
GO:1901360
organic cyclic compo...
0.00345
850 / 853
GO:0006810
transport
0.00169
491 / 491