How To Use GOstats
Testing Gene Lists for GO Term Association

S. Falcon and R. Gentleman

October 30, 2017

1 Introduction

The GOstats package has extensive facilities for testing the association of Gene Ontology (GO) The Gene Ontology Consortium (2000) terms to genes in a gene list. You can test for both over and under representation of GO terms using either the standard Hypergeometric test or a conditional Hypergeometric test that uses the relationships among the GO terms for conditioning (similar to that presented in Alexa et al. (2006)).

In this vignette we describe the preprocessing required to construct inputs for the main testing function, `hyperGTest`, the algorithms used, and the structure of the return value. We use a microarray data set (Chiaretti et al., 2004) from a clinical trial in acute lymphoblastic leukemia (ALL) to work an example analysis. In the ALL data, we focus on the patients with B-cell derived ALL, and in particular on comparing the group with ALL1/AF4 to those with no observed cytogenetic abnormalities.

To get started, load the packages needed for this analysis:

```
> library("ALL")
> library("hgu95av2.db")
> library("GO.db")
> library("annotate")
> library("genefilter")
> library("GOstats")
> library("RColorBrewer")
> library("xtable")
> library("Rgraphviz")
```

2 Preprocessing and Inputs

To perform an analysis using the Hypergeometric-based tests, one needs to define a *gene universe* (usually conceptualized as the number of balls in an urn) and a list of selected genes from the universe. While it is clear that the selected gene list determines to a large degree the results of the analysis, the fact that the universe has a large effect on the conclusions is, perhaps, less obvious.

For microarray data, one can use the unique gene identifiers assayed in the experiment as the gene universe. However, the presence of a gene on the array does not necessarily mean much. Some arrays, such as those from Affymetrix, attempt to include probes for as much of the genome as possible. Since not all genes will be expressed under all conditions (a widely held belief is that about 40% of the genome is expressed in any tissue), it may be sensible to reduce the universe to those that are expressed.

To identify the set of expressed genes from a microarray experiment, we propose that a non-specific filter be applied and that the genes that pass the filter be used to form the universe for any subsequent functional analyses. Below, we outline the non-specific filtering procedure used for the example analysis.

Once a gene universe has been established, one can apply any number of methods to select genes. For the example analysis we use a simple t-test to identify differentially expressed genes among the two subgroups in the sample population.

It is worth noting that the effect of increasing the universe size with genes that are irrelevant to the questions at hand, in general, has the effect of making the resultant p-values look more significant. For example, in a universe of 1000 genes where 400 have been selected, suppose that a GO term has 40 gene annotations from the universe of 1000. If 10 of the genes in the selected gene list are among the 40 genes annotated at this category, then the Hypergeometric p-value is 0.99. However, if the gene universe contained 5000 genes, the p-value would drop to 0.001.

2.1   Non-specific filtering

First we load the `ALL` data object and extract the subset of the data we wish to analyze: subjects with either no cytogenetic abnormality ("NEG") or those with "ALL1/AF4".

```
> data(ALL, package="ALL")
> ## For this data we can have ALL1/AF4 or BCR/ABL
> subsetType <- "ALL1/AF4"
> ## Subset of interest: 37BRC/ABL + 42NEG = 79 samples
> Bcell <- grep("^B", as.character(ALL$BT))
> bcrAblOrNegIdx <- which(as.character(ALL$mol) %in% c("NEG", subsetType))
> bcrAblOrNeg <- ALL[, intersect(Bcell, bcrAblOrNegIdx)]
> bcrAblOrNeg$mol.biol = factor(bcrAblOrNeg$mol.biol)
```

We begin our non-specific filtering by removing probe sets that have no Entrez Gene identifier in our annotation data.

```
> ## Remove genes that have no entrezGene id
> entrezIds <- mget(featureNames(bcrAblOrNeg), envir=hgu95av2ENTREZID)
> haveEntrezId <- names(entrezIds)[sapply(entrezIds, function(x) !is.na(x))]
> numNoEntrezId <- length(featureNames(bcrAblOrNeg)) - length(haveEntrezId)
> bcrAblOrNeg <- bcrAblOrNeg[haveEntrezId, ]
```

Similarly, we remove probe sets for which we have no GO annotation.

```
> ## Remove genes with no GO mapping
> haveGo <- sapply(mget(featureNames(bcrAblOrNeg), hgu95av2GO),
+                  function(x) {
+                      if (length(x) == 1 && is.na(x))
+                        FALSE
+                      else TRUE
+                  })
> numNoGO <- sum(!haveGo)
> bcrAblOrNeg <- bcrAblOrNeg[haveGo, ]
```

Now use the IQR of each probe set across samples to remove those probe sets that have little variation across samples. Also, since there is an imbalance of men and women by group, we remove probe sets that measure genes on the Y chromosome.

```
> ## Non-specific filtering based on IQR
> iqrCutoff <- 0.5
> bcrAblOrNegIqr <- apply(exprs(bcrAblOrNeg), 1, IQR)
> selected <- bcrAblOrNegIqr > iqrCutoff
> ## Drop those that are on the Y chromosome
> ## because there is an imbalance of men and women by group
> chrN <- mget(featureNames(bcrAblOrNeg), envir=hgu95av2CHR)
> onY <- sapply(chrN, function(x) any(x=="Y"))
> onY[is.na(onY)] <- FALSE
> selected <- selected & !onY
> nsFiltered <- bcrAblOrNeg[selected, ]
```

Here we ensure that each probe set maps to exactly one Entrez Gene ID. If multiple probes are found to map to the same Entrez Gene ID, we select the probe with largest IQR (from the computation above).

```
> numNsWithDups <- length(featureNames(nsFiltered))
> nsFilteredIqr <- bcrAblOrNegIqr[selected]
> uniqGenes <- findLargest(featureNames(nsFiltered), nsFilteredIqr,
+                          "hgu95av2")
> nsFiltered <- nsFiltered[uniqGenes, ]
> numSelected <- length(featureNames(nsFiltered))
> ##set up some colors
> BCRcols = ifelse(nsFiltered$mol == subsetType, "goldenrod", "skyblue")
> cols = brewer.pal(10, "RdBu")
```

Finally, we can define the gene universe we will use for the Hypergeometric tests.

```
> ## Define gene universe based on results of non-specific filtering
> affyUniverse <- featureNames(nsFiltered)
> entrezUniverse <- unlist(mget(affyUniverse, hgu95av2ENTREZID))
> if (any(duplicated(entrezUniverse)))
```

```
+     stop("error in gene universe: can't have duplicate Entrez Gene Ids")
> ## Also define an alternate universe based on the entire chip
> chipAffyUniverse <- featureNames(bcrAblOrNeg)
> chipEntrezUniverse <- mget(chipAffyUniverse, hgu95av2ENTREZID)
> chipEntrezUniverse <- unique(unlist(chipEntrezUniverse))
```

Summary of non-specific filtering: Our non-specific filtering procedure removed probes missing either Entrez Gene identifies or mappings to GO terms. Because of an imbalance of men and women by group, probes measuring genes on the Y chromosome were dropped. The inter-quartile range was used with a cutoff of 0.5 to select probes with sufficient variability across samples to be informative; probes with little variability across all samples are inherently uninteresting. Finally, the set of remaining probes was refined by ensuring that each probe maps to exactly one Entrez Gene identifier. For those probes mapping to the same Entrez Gene ID, the probe with largest IQR was selected.

Producing a set of Entrez Gene identifiers that map to a unique set of probes at the non-specific filtering stage is important because genes are mapped to GO categories using Entrez Gene IDs and we want to avoid double counting any GO categories. In all, the filtering left 3318 genes.

2.2 Gene selection via t-test

We apply a standard t-test to identify a set of genes with differential expression between the ALL1/AF4 and NEG groups.

```
> ttestCutoff <- 0.05
> ttests = rowttests(nsFiltered, "mol.biol")
> smPV = ttests$p.value < ttestCutoff
> pvalFiltered <- nsFiltered[smPV, ]
> selectedEntrezIds <- unlist(mget(featureNames(pvalFiltered),
+                                  hgu95av2ENTREZID))
```

There are 662 genes with p-values less than 0.05. We do not make use of any p-value correction methods since we are interested in a relatively long gene list.

A detail often omitted from GO association analyses is the fact that the t-test, and most similar statistics, are directional. For a given gene, average expression might be higher in the ALL1/AF4 group than in the NEG group, whereas for a different gene it might be the NEG group that shows the increased expression. By only looking at the p-values for the test statistics, the directionality is lost. The danger is that an association with a GO category may be found where the genes are not differentially expressed in the same direction. One way to tackle this problem is by separating the selected gene list into two lists according to direction and running two analyses. A more elegant approach is the subject of further research.

2.3 Inputs

Often one wishes to perform many similar analyses using slightly different sets of parameters and to facilitate this pattern of usage the main interface to the Hypergeometric tests, hyper-GTest, takes a single parameter object as its argument. This argument is an instance of class

GOHyperGParams. There are also parameter classes *KEGGHyperGParams* and *PFAMHyper-GParams* defined in the `Category` package that allow for testing for association with KEGG pathways and PFAM protein domains, respectively.

Using a parameter class instead of individual arguments makes it easier to organize and execute a series of related analyses. For example, one can create a list of *GOHyperGParams* instances and perform the Hypergeometric test on each using R's `lapply` function:

```
resultList <- lapply(lisOfParamObjs, hyperGTest)
```

In the absence of a parameter class, this could be achieved using `mapply`, but the result would be less readable. Because parameter objects can be copied and modified, they tend to reduce duplication of code. We'll demonstrate this in the following example.

Below, we create a parameter instance by specifying the gene list, the universe, the name of the annotation data package, and the GO ontology we wish to interrogate. For the example analysis, we have stored the vector of Entrez Gene identifiers making up the gene universe in `entrezUniverse`. The selected genes are stored in `selectedEntrezIds`. If you are following along with your own data and have an *ExpressionSet* instance resulting from a non-specific filtering procedure, you can create the `entrezUniverse` and `selectedEntrezIds` vectors using code similar to that shown here:

```
> entrezUniverse <- unlist(mget(featureNames(yourData),
+                               hgu95av2ENTREZID))
> if (any(duplicated(entrezUniverse)))
+   stop("error in gene universe: can't have duplicate Entrez Gene Ids")
> pvalFiltered <- yourData[hasSmallPvalue, ]
> selectedEntrezIds <- unlist(mget(featureNames(pvalFiltered),
+                                   hgu95av2ENTREZID))
```

Here is a description of all the arguments needed to construct a *GOHyperGParams* instance.

geneIds A vector of gene identifiers that defines the selected list of genes. This is often the output of a test for differential expression among two sample groups. For experiments using Affymetrix expression arrays, this should be a vector of Entrez Gene IDs. If you are using the YEAST annotation package, the vector will consist of yeast systematic names.

universeGeneIds A vector of gene identifiers that defines the universe of possible genes. We recommend using the set of gene IDs that result from non-specific filtering. The identifiers should be of the same type as the geneIds; for Affymetrix arrays, these will be Entrez Gene IDs.

annotation A string giving the name of the annotation data package that corresponds to the chip used in the experiment.

ontology A two-letter string specifying one of the three GO ontologies: BP, CC, or MF. The hyperGTest function only tests a single GO ontology at one time.

pvalueCutoff A numeric values between zero and one used as a p-value cutoff for p-values generated by the Hypergeometric test. When the test being performed is non-conditional, this is only used as a default value for printing and summarizing the results. For a conditional analysis, the cutoff is used during the computation to determine perform the conditioning: child terms with a p-value less than `pvalueCutoff` are conditioned out of the test for their parent term.

conditional A logical value. If `TRUE`, the test performed uses the conditional algorithm. Otherwise, a standard Hypergeometric test is performed. When 'conditional(p) == TRUE', the 'hyperGTest' function uses the structure of the GO graph to estimate for each term whether or not there is evidence beyond that which is provided by the term's children to call the term in question statistically overrepresented. The algorithm conditions on all child terms that are themselves significant at the specified p-value cutoff. Given a subgraph of one of the three GO ontologies, the terms with no child categories are tested first. Next the nodes whose children have already been tested are tested. If any of a given node's children tested significant, the appropriate conditioning is performed.

testDirection A string which can be either "over" or "under". This determines whether the test performed detects over or under represented GO terms.

```
> hgCutoff <- 0.001
> params <- new("GOHyperGParams",
+               geneIds=selectedEntrezIds,
+               universeGeneIds=entrezUniverse,
+               annotation="hgu95av2.db",
+               ontology="BP",
+               pvalueCutoff=hgCutoff,
+               conditional=FALSE,
+               testDirection="over")
>
```

We would also like to perform a conditional test. Instead of having to define a new *GO-HyperGParams* instance by hand, we can create a copy and update just the parameter of interest.

```
> paramsCond <- params
> conditional(paramsCond) <- TRUE
```

A similar approach would work to create a parameter object for testing a different GO ontology or to create an object for testing under representation.

3   Outputs and Result Summarization

The `hyperGTest` function returns an instance of class *GOHyperGResult*. When the input parameter object is a *KEGGHyperGParams* or *PFAMHyperGParams* instance, the result will

instead be a *HyperGResult* object. Most of the reporting and summarization methods demonstrated here will work the same, except for those that deal specifically with GO or the GO graph.

As shown below, printing the result at the R prompt provides a brief summary of the test performed and the number of significant terms found. Depending on how you pre-processed your gene list and gene universe, The hyperGTest function may have to do even more filtering on both of these for you. Genes that are not marked with a GO term from the ontology that you specified will have to be discarded, and so you might notice that your gene list and gene universe had shrank somewhat when you print the results.

```
> hgOver <- hyperGTest(params)
> hgCondOver <- hyperGTest(paramsCond)
>

> hgOver

Gene to GO BP  test for over-representation
6720 GO BP ids tested (96 have p < 0.001)
Selected gene set size: 647
    Gene universe size: 3224
    Annotation package: hgu95av2

> hgCondOver

Gene to GO BP Conditional test for over-representation
6720 GO BP ids tested (50 have p < 0.001)
Selected gene set size: 647
    Gene universe size: 3224
    Annotation package: hgu95av2
```

The `summary` function returns a *data.frame* summarizing the result. By default, only the results for terms with a p-value less than the cutoff specified in the parameter instance will be shown. However, you can set a new cutoff using the `pvalue` argument. You can also set a minimum number of genes for each term using the `categorySize` argument. For *GOHyperGResult* objects, the `summary` method also has a `htmlLinks` argument. When `TRUE`, the GO term names are printed as HTML links to the GO website.

```
> df <- summary(hgOver)
> names(df)                        # the columns

[1] "GOBPID"    "Pvalue"    "OddsRatio" "ExpCount"  "Count"     "Size"
[7] "Term"

> dim(summary(hgOver, pvalue=0.1))

[1] 1021    7

> dim(summary(hgOver, categorySize=10))
```

```
[1] 92  7
```

Now we demonstrate some of the accessor functions that can be used to extract detail from a result object. These functions are all detailed in their respective manual pages.

```
> pvalues(hgOver)[1:3]

  GO:0006955    GO:0007165    GO:0044700
1.714019e-09 4.781224e-09 5.826245e-09

> oddsRatios(hgOver)[1:3]

GO:0006955 GO:0007165 GO:0044700
  1.860013   1.668151   1.660997

> expectedCounts(hgOver)[1:3]

GO:0006955 GO:0007165 GO:0044700
  126.4299   265.9042   280.9553

> geneCounts(hgOver)[1:3]

GO:0006955 GO:0007165 GO:0044700
       182        331        346

> universeCounts(hgOver)[1:3]

GO:0006955 GO:0007165 GO:0044700
       630       1325       1400

> length(geneIds(hgOver))

[1] 647

> length(geneIdUniverse(hgOver)[[3]])

[1] 1400

> ## GOHyperGResult _only_
> ## (NB: edges go from parent to child)
> goDag(hgOver)

A graphNEL graph with directed edges
Number of Nodes = 6720
Number of Edges = 15784

> geneMappedCount(hgOver)

[1] 647
```

```
> universeMappedCount(hgOver)

[1] 3224

> conditional(hgOver)

[1] FALSE

> testDirection(hgOver)

[1] "over"

> testName(hgOver)

[1] "GO" "BP"

>
```

To make it easy for non-technical users to review the results, the `htmlReport` function generates an HTML file that can be viewed in any web browser. The output generated by `htmlReport` as called below is output to your current working directory.

```
> htmlReport(hgCondOver, file="ALL_hgco.html")
```

4   GOstats Capabilities

In the Hypergeometric model, each term is treated as an independent classification. Each gene is cross-classified according to whether or not it has been selected and whether or not it is annotated, not necessarily specifically annotated, at a particular term. A Hypergeometric probability is computed to assess whether the number of selected genes associated with the term is larger than expected.

The `hyperGTest` function provides an implementation of the commonly applied Hypergeometric calculation for over or under-representation of GO terms in a specified gene list. This computation ignores the structure of the GO terms, treating each term as independent from all other terms.

Often an analysis for GO term associations results in the identification of directly related GO terms with considerable overlap of genes. This is because each GO term inherits all annotations from its more specific descendants. To alleviate this problem, we have implemented a method which conditions on all child terms that are themselves significant at a specified *p*-value cutoff. Given a subgraph of one of the three GO ontologies, we test the leaves of the graph, that is, those terms with no child terms. Before testing the terms whose children have already been tested, we remove all genes annotated at significant children from the parent's gene list. This continues until all terms have been tested.

```
> toLatex(sessionInfo())
```

- R version 3.4.2 (2017-09-28), x86_64-pc-linux-gnu

- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`

- Running under: `Ubuntu 16.04.3 LTS`

- Matrix products: default

- BLAS: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRblas.so`

- LAPACK: `/home/biocbuild/bbs-3.6-bioc/R/lib/libRlapack.so`

- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, stats4, utils

- Other packages: ALL 1.19.0, AnnotationDbi 1.40.0, AnnotationForge 1.20.0, Biobase 2.38.0, BiocGenerics 0.24.0, Category 2.44.0, GO.db 3.4.2, GOstats 2.44.0, GSEABase 1.40.0, IRanges 2.12.0, KEGG.db 3.2.3, Matrix 1.2-11, RColorBrewer 1.1-2, RSQLite 2.0, Rgraphviz 2.22.0, S4Vectors 0.16.0, XML 3.98-1.9, annotate 1.56.0, genefilter 1.60.0, graph 1.56.0, hgu95av2.db 3.2.3, org.Hs.eg.db 3.4.2, xtable 1.8-2

- Loaded via a namespace (and not attached): DBI 0.7, RBGL 1.54.0, RCurl 1.95-4.8, Rcpp 0.12.13, bit 1.1-12, bit64 0.9-7, bitops 1.0-6, blob 1.1.0, compiler 3.4.2, digest 0.6.12, lattice 0.20-35, memoise 1.1.0, pkgconfig 2.0.1, rlang 0.1.2, splines 3.4.2, survival 2.41-3, tibble 1.3.4, tools 3.4.2

References

Adrian Alexa, Jorg Rahnenfuhrer, and Thomas Lengauer. Improved scoring of functional groups from gene expression data by decorrelating GO graph structure. *Bioinformatics*, 22 (13):1600–7, 2006.

S. Chiaretti, X Li, R Gentleman, A Vitale, M. Vignetti, F. Mandelli, J. Ritz, , and R. Foa. Gene expression profile of adult t-cell acute lymphocytic leukemia identifies distinct subsets of patients with different response to therapy and survival. *Blood*, 103:2771–2778, 2004.

The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25:25–29, 2000.