FunctionalNetworks: An Algorithm for Gene and Gene Set Network Inference.

Alejandro Quiroz-Zárate[1], Benjamin Haibe-Kains[2], Hugo Aerts[3], and John Quackenbush[1]

[1]Biostatistics and Computational Biology, Dana-Farber Cancer Institute, Boston, Massachusetts, United States of America.
[2]Bioinformatics and Computational Genomics Laboratory, Institut de Recherches Cliniques de Montréal, Montreal, Quebec, Canada.
[3]Harvard Medical School, Boston, Massachusetts, United States of America

November 26, 2013

Contents

1 Introduction

The *FunctionalNetwork* package provides an algorithm to infer networks at a gene and gene set level. This package includes (i) functions to perform network inference (ii) examples to extract and visualize the results of such comparisons.
The *FunctionalNetwork* package provides functions to implement the network algorithm to infer gene and gene set networks on datasets with cross-sectional or time series design.

1.1 Installation

FunctionalNetworks requires *R* ($>= 2.10.0$) installed. To install *FunctionalNetworks*, source biocLite from bioconductor:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("FunctionalNetworks")
```

Load the *FunctionalNetworks*, into your current workspace:

```
> library(FunctionalNetworks)
```

1.2 Further help

To view the *FunctionalNetworks* description and a summary of all the functions within *FunctionalNetworks*, type the following:

```
> library(help=FunctionalNetworks)
```

1.3 Citing

We are delighted if you use this package. Please do email us if you find a bug or have a suggestion. We would be very grateful if you could cite:
Quiroz-Zarate A, Haibe-Kains B and Quackenbush J (2013). *Manuscript in preparation*.

2 An application in Breast cancer.

We will very briefly demonstrate the use of some functions in *FunctionalNetworks* by providing its application on a cross-sectional dataset.
We use the *breastCancerVDX* data library from Bioconductor for demonstration purposes under a cross-sectional design. This data set corresponds to the data set from [1]. Minn, AJ and colleagues used Affymetrix U133A Gene Chips to profile gene expression in 286 fresh-frozen tumor samples from patients with lymph-node-negative breast cancer who were treated during $1980 - 95$, but who did not receive systemic neoadjuvant or adjuvant therapy. These samples correspond from the data set used in [3] with GEO reference accession number

GSE2034, from the tumor bank at the Erasmus Medical Center in Rotterdam, Netherlands. An additional 58 estrogen receptor-negative samples were added from [1] GEO (GSE5327). In total 209 tumor samples are classified as ER+ and 135 as ER-. Even though this data set comes from a 5-year follow-up design, the way the data is conceived for this analysis is cross-sectional.

2.1 Example: Data analysis under a cross-sectional setting.

This is an example on how to perform an analysis with the proposed method in [2] for a data set with cross-sectional design. This example is divided in two parts. The data preparation and the execution of the network algorithm.

2.1.1 Data preprocessing stage

The original gene expression data set Minn AJ and colleagues [1] has a U133A Affymetrix platform. The normalized data set was saved to the variable vdx in the *breastCancerVDX* data library from Bioconductor.

```
> library(breastCancerVDX)
> library(Biobase)
> data(vdx)
> gene.data=exprs(vdx)   # Gene expression of the package
> vdx.annot=fData(vdx)   # Annotation associated to the dataset
> vdx.clinc=pData(vdx)   # Clinical information associated to the dataset
> # Identifying the sample identifiers associated to ER+ and ER- breast cancer
> er.pos=which(vdx.clinc$er==1)
> er.neg=which(vdx.clinc$er==0)
> # Checking if the probeset are ordered with respect to the dataset
> all(rownames(gene.data)==as.character(vdx.annot[,1]))

[1] TRUE

> # Checking if the sample identifiers are order with respect to the dataset
> all(colnames(gene.data)==as.character(vdx.clinc[,1]))

[1] TRUE

> # Changing the row identifiers to the gene identifiers of interest
> rownames(gene.data)=as.character(vdx.annot[,2])
> vec.gene=NULL
> vec.ids=NULL
> for(i in 1:dim(vdx.annot)[1])
+ {
+   aux=strsplit(gsub("[^[:alnum:] ]"," ",vdx.annot[i,3])," ")
+         aux.num=unlist(lapply(aux,nchar))
+         aux=aux[[1]][which(aux.num!=0)]
+         aux.ids=rep(i,length(aux))
+          vec.ids=c(vec.ids,aux.ids)
+             vec.gene=c(vec.gene,aux)
```

```
+ }
> which.erase=which(is.na(vec.gene)==T)
> vec.gene=vec.gene[-which.erase]
> vec.ids=vec.ids[-which.erase]
> unique.genes=sort(unique(vec.gene))
> # Erase the first 26, because they are not reported in GO
> unique.genes=unique.genes[-seq(1,26)]
> #= Because we have several measurements for a gene, we filter the genes
> # Function to obtain the genes with highest variabilty
> indices=unlist(lapply(unique.genes,function(x){quienes=vec.gene==x;aux=which(quienes==T)
+              aux.2=vec.ids[aux]
+          if(length(aux)>1){
+                      aux.2=vec.ids[aux]
+                                  var.r = apply(gene.data[aux.2,],1,var)
+                                 aux.2=aux.2[which.max(var.r)]};return(aux.2)}))
> gene.data=gene.data[indices,]   # Final genes to keep
> rownames(gene.data)=unique.genes # Assign the gene symbols
```

In order to implement the Network inference algorithm, there are some datasets that need to be generated from the gene expression data we have just created: `gene.data`. At this stage we will use the function `data.generation` from the *FunctionalNetwork* package. For this example it is assumed that the gmt file containing the Molecular Functions (MF) ontology from GO is stored in the path: "/Users/MyLapTop/Netwokrs/c5.mf.v4.0.symbols.gmt". The "c5.mf.v4.0.symbols.gmt" file can be dowloaded from the MSigDB web site: "http://www.broadinstitute.org/gsea/msigdb/index.jsp"

```
> gmt.file.path="/Users/MyLapTop/Netwokrs/c5.mf.v4.0.symbols.gmt"
> min.gene.set.size=5
> data4network=data.generation(gmt.file.path,gene.data,er.neg,min.gene.set.size)
```

Before we estimated the networks, there is a crucial step: the computation of the Bayesian Information Criteria (BIC) for all the possible associations among genes and among gene sets [2]. This is done before the network computation step

```
> gene.data=data4network$gene.data
> affy.loc=data4network$affy.loc
> set.data=data4network$set.data
> bic4network=bic.generation(gene.data,affy.loc,set.data)
```

2.1.2 Network estimation

To perform the network estimation, the results of the functions `data.generation` and `bic.generation` are needed. It has to be noted that the calculation of the BIC at the gene and gene set level is very time consuming. So only for purposes of this example we generated toy datasets. These toy datasets are based on a random selection of 5 of the genes from the original `gene.data`. These toy datasets from the application of the functions `data.generation` and `bic.generation` are stored in `data.4.toy.example` and `bic.4.toy.network` on this package. In total there are 216 genes considered for the gene network, (see `affy.loc` in `bic.4.toy.network`) and a toltal of 62 gene sets for the gene set network (see `set.data` in `data.4.toy.network`)

```
> library(FunctionalNetworks)
> data(data.4.toy.network,package="FunctionalNetworks")
```

```
> data(bic.4.toy.network,package="FunctionalNetworks")
> nsim=1000
> burn=100
> network.toy.estimation=network.estimation(nsim,burn,data.4.toy.network,bic.4.toy.network)

Percentage of iterations completed: 1
Percentage of iterations completed: 2
Percentage of iterations completed: 3
Percentage of iterations completed: 4
Percentage of iterations completed: 5
Percentage of iterations completed: 6
Percentage of iterations completed: 7
Percentage of iterations completed: 8
Percentage of iterations completed: 9
Percentage of iterations completed: 10
Percentage of iterations completed: 11
Percentage of iterations completed: 12
Percentage of iterations completed: 13
Percentage of iterations completed: 14
Percentage of iterations completed: 15
Percentage of iterations completed: 16
Percentage of iterations completed: 17
Percentage of iterations completed: 18
Percentage of iterations completed: 19
Percentage of iterations completed: 20
Percentage of iterations completed: 21
Percentage of iterations completed: 22
Percentage of iterations completed: 23
Percentage of iterations completed: 24
Percentage of iterations completed: 25
Percentage of iterations completed: 26
Percentage of iterations completed: 27
Percentage of iterations completed: 28
Percentage of iterations completed: 29
Percentage of iterations completed: 30
Percentage of iterations completed: 31
Percentage of iterations completed: 32
Percentage of iterations completed: 33
Percentage of iterations completed: 34
Percentage of iterations completed: 35
Percentage of iterations completed: 36
Percentage of iterations completed: 37
Percentage of iterations completed: 38
Percentage of iterations completed: 39
Percentage of iterations completed: 40
Percentage of iterations completed: 41
Percentage of iterations completed: 42
Percentage of iterations completed: 43
Percentage of iterations completed: 44
Percentage of iterations completed: 45
Percentage of iterations completed: 46
```

```
Percentage of iterations completed: 47
Percentage of iterations completed: 48
Percentage of iterations completed: 49
Percentage of iterations completed: 50
Percentage of iterations completed: 51
Percentage of iterations completed: 52
Percentage of iterations completed: 53
Percentage of iterations completed: 54
Percentage of iterations completed: 55
Percentage of iterations completed: 56
Percentage of iterations completed: 57
Percentage of iterations completed: 58
Percentage of iterations completed: 59
Percentage of iterations completed: 60
Percentage of iterations completed: 61
Percentage of iterations completed: 62
Percentage of iterations completed: 63
Percentage of iterations completed: 64
Percentage of iterations completed: 65
Percentage of iterations completed: 66
Percentage of iterations completed: 67
Percentage of iterations completed: 68
Percentage of iterations completed: 69
Percentage of iterations completed: 70
Percentage of iterations completed: 71
Percentage of iterations completed: 72
Percentage of iterations completed: 73
Percentage of iterations completed: 74
Percentage of iterations completed: 75
Percentage of iterations completed: 76
Percentage of iterations completed: 77
Percentage of iterations completed: 78
Percentage of iterations completed: 79
Percentage of iterations completed: 80
Percentage of iterations completed: 81
Percentage of iterations completed: 82
Percentage of iterations completed: 83
Percentage of iterations completed: 84
Percentage of iterations completed: 85
Percentage of iterations completed: 86
Percentage of iterations completed: 87
Percentage of iterations completed: 88
Percentage of iterations completed: 89
Percentage of iterations completed: 90
Percentage of iterations completed: 91
Percentage of iterations completed: 92
Percentage of iterations completed: 93
Percentage of iterations completed: 94
Percentage of iterations completed: 95
Percentage of iterations completed: 96
Percentage of iterations completed: 97
```

```
Percentage of iterations completed: 98
Percentage of iterations completed: 99
Percentage of iterations completed: 100
```

The result of the network estimation function is stored in `network.toy.estimation`. This object has 3 items: `Algorithm.results`, `Set.names` and `Gene.names`. The object `Algorithm.results` containes 6 different results: `Gene.network`, `Set.network`, `BIC.gene`, `BIC.set`, `RSS.gene` and `RSS.set`. The result `Gene.network` correspond to the matrix associated to the gene network. The rows correspond to the objective nodes and the columns to the source nodes. The entries of this matrix are the number of times each source node was a predictor of the objective node. The result `Set.network` correspond to the matrix associated to the gene set network. The rows correspond to the objective nodes and the columns to the source nodes. The entries of this matrix are the number of times each source node was a predictor of the objective node. `BIC.gene` corresponds to the overall BIC of the gene network for each iteration. `BIC.set` corresponds to the overall BIC of the gene set network for each iteration. `RSS.gene` corresponds to the overall residual sum of squares (RSS) of the gene network for each iteration. `RSS.set` corresponds to the overall RSS of the gene set network for each iteration. Finally `Set.names` and `Gene.names` corrspon to the names of the nodes on the gene set and gene network respectively.

2.1.3 Network estimation

To analyze the performance of the network one can plot the `RSS.gene` or the `BIC.set`. The following are examples of such commands:

```
> library(FunctionalNetworks)
> data(data.4.toy.network,package="FunctionalNetworks")
> data(bic.4.toy.network,package="FunctionalNetworks")
> nsim=1000
> burn=100
> network.toy.estimation=network.estimation(nsim,burn,data.4.toy.network,bic.4.toy.network)
```

```
Percentage of iterations completed: 1
Percentage of iterations completed: 2
Percentage of iterations completed: 3
Percentage of iterations completed: 4
Percentage of iterations completed: 5
Percentage of iterations completed: 6
Percentage of iterations completed: 7
Percentage of iterations completed: 8
Percentage of iterations completed: 9
Percentage of iterations completed: 10
Percentage of iterations completed: 11
Percentage of iterations completed: 12
Percentage of iterations completed: 13
Percentage of iterations completed: 14
Percentage of iterations completed: 15
Percentage of iterations completed: 16
Percentage of iterations completed: 17
```

```
Percentage of iterations completed: 18
Percentage of iterations completed: 19
Percentage of iterations completed: 20
Percentage of iterations completed: 21
Percentage of iterations completed: 22
Percentage of iterations completed: 23
Percentage of iterations completed: 24
Percentage of iterations completed: 25
Percentage of iterations completed: 26
Percentage of iterations completed: 27
Percentage of iterations completed: 28
Percentage of iterations completed: 29
Percentage of iterations completed: 30
Percentage of iterations completed: 31
Percentage of iterations completed: 32
Percentage of iterations completed: 33
Percentage of iterations completed: 34
Percentage of iterations completed: 35
Percentage of iterations completed: 36
Percentage of iterations completed: 37
Percentage of iterations completed: 38
Percentage of iterations completed: 39
Percentage of iterations completed: 40
Percentage of iterations completed: 41
Percentage of iterations completed: 42
Percentage of iterations completed: 43
Percentage of iterations completed: 44
Percentage of iterations completed: 45
Percentage of iterations completed: 46
Percentage of iterations completed: 47
Percentage of iterations completed: 48
Percentage of iterations completed: 49
Percentage of iterations completed: 50
Percentage of iterations completed: 51
Percentage of iterations completed: 52
Percentage of iterations completed: 53
Percentage of iterations completed: 54
Percentage of iterations completed: 55
Percentage of iterations completed: 56
Percentage of iterations completed: 57
Percentage of iterations completed: 58
Percentage of iterations completed: 59
Percentage of iterations completed: 60
Percentage of iterations completed: 61
Percentage of iterations completed: 62
Percentage of iterations completed: 63
Percentage of iterations completed: 64
Percentage of iterations completed: 65
Percentage of iterations completed: 66
Percentage of iterations completed: 67
Percentage of iterations completed: 68
```

```
Percentage of iterations completed: 69
Percentage of iterations completed: 70
Percentage of iterations completed: 71
Percentage of iterations completed: 72
Percentage of iterations completed: 73
Percentage of iterations completed: 74
Percentage of iterations completed: 75
Percentage of iterations completed: 76
Percentage of iterations completed: 77
Percentage of iterations completed: 78
Percentage of iterations completed: 79
Percentage of iterations completed: 80
Percentage of iterations completed: 81
Percentage of iterations completed: 82
Percentage of iterations completed: 83
Percentage of iterations completed: 84
Percentage of iterations completed: 85
Percentage of iterations completed: 86
Percentage of iterations completed: 87
Percentage of iterations completed: 88
Percentage of iterations completed: 89
Percentage of iterations completed: 90
Percentage of iterations completed: 91
Percentage of iterations completed: 92
Percentage of iterations completed: 93
Percentage of iterations completed: 94
Percentage of iterations completed: 95
Percentage of iterations completed: 96
Percentage of iterations completed: 97
Percentage of iterations completed: 98
Percentage of iterations completed: 99
Percentage of iterations completed: 100
```

```
> par(mfrow=c(2,1))
> plot(network.toy.estimation$Algorithm.results$BIC.set,xlab="Iterations"
+      ,ylab="BIC",main="Overall BIC of the gene set network",type="l",lwd=3)
> usr=par("usr")
> rect(1000,usr[3],100,usr[4],col="gray80")
> lines(network.toy.estimation$Algorithm.results$BIC.set,type="l",lwd=3)
> legend(400,18000,legend=c("BIC","Iterations considered"),col=c("black","gray80")
+        ,lwd=c(3,0),fill=c("white","gray80"),border=c("white","black"),bg="white")
> plot(network.toy.estimation$Algorithm.results$RSS.gene,xlab="Iterations"
+      ,ylab="RSS",main="Overall RSS of the gene network",type="l",lwd=3)
> usr=par("usr")
> rect(1000,usr[3],100,usr[4],col="gray80")
> lines(network.toy.estimation$Algorithm.results$RSS.gene,type="l",lwd=3)
> legend(400,4600,legend=c("RSS","Iterations considered"),col=c("black","gray80")
+        ,lwd=c(3,0),fill=c("white","gray80"),border=c("white","black"),bg="white")
```

3 Session Info

- R version 3.0.2 (2013-09-25), `x86_64-apple-darwin10.8.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: Biobase 2.20.1, BiocGenerics 0.6.0, breastCancerVDX 1.0.5, FunctionalNetworks 0.99.6
- Loaded via a namespace (and not attached): tools 3.0.2

References

[1] Minn AJ, Gupta GP, Padua D, Bos P, Nguyen DX, Nuyten D, Kreike B, Zhang Y, Wang Y, Ishwaran H, Foekens JA, Van de Vijver M and Massagué J: Lung Metastasis Genes Couple Breast Tumor Size and Metastatic Spread. *PNAS*, **104(16)**, 6740-6745. 2007.

[2] Quiroz-Zarate A and Quackenbush J XXXX: Biological functional networks *Journal* **Vol(Num):Page 1-Page N**. 2013.

[3] Wang Y, Klijn JGM, Zhang Y, Sieuwerts AM, Look MP, Yanh F, Talantov D, Timmermans M, Gelder, MEMG, Yu J, Jatkoe T, Berns EMJJ, Atkins D and Foekens JA: Gene-expression Profiles to Predict Distant Metastasis of Lymph-Node-Negative Primary Breast Cancer. *Lancet*, **365**, 671-679. 2005.