GeneGeneInteR vignette
Gene-based gene-gene interaction analysis in case-controls association studies

Mathieu Emily

May 19, 2016

1 Introduction

The package **GeneGeneInteR** aims at providing a collection of statistical methods to search for interaction between genes in case-control association studies. These methods are dedicated to the analysis of biallelic SNP (Single Nucleotide Polymorphism) genotype data. This vignette describes the complete analysis pipeline of a set $k > 2$ genes, going from data importation to results visualization via data manipulation and statistical analysis.

In the remainder of this vignette, we illustrate our pipeline through the analysis of a case-control dataset publicly available in the NCBI repository GSE39428 series [CXW+13]. The dataset contains the genotypes of 312 SNPs from 17 genes in a total of 429 patients (266 individuals affected by Rheumatoid Arthritis and 163 Health controls) and is attached to our package **GeneGeneInteR** as an external file in the ped format.

2 Data importation and manipulation

2.1 Importation of genotype data

At first, the path for the files containing genotype data and information regarding the SNP set are loaded:

```
ped <- system.file("extdata/example.ped", package="GeneGeneInteR")
info <- system.file("extdata/example.info", package="GeneGeneInteR")
## Information about position of the snps
posi <- system.file("extdata/example.txt", package="GeneGeneInteR")
```

The importation is performed with the `importFile` function:

```
R> data <- importFile(file=ped, snps=info, pos=posi, pos.sep="")
```

The object `data` is a list of 3 elements: `status`, `snpX` a `SnpMatrix` object and `genes.info` a `data.frame`. The `status` is available only if the imported format is ped.

```
R> summary(data)
              Length  Class       Mode
  status      429     factor      numeric
  snpX        133848  SnpMatrix   raw
  genes.info  4       data.frame  list
```

We can check that the `snpX` object contains the genotype of 429 individuals for 312 SNPs.

```
R> data$snpX
  A SnpMatrix with 429 rows and 312 columns
  Row names:  H97 ...  RA345
  Col names:  rs1002788 ...  rs9502656
```

The genes.info is a data.frame with exactly four columns that are named as follows: Chromosome, Genenames, SNPnames and Position.

```
R> summary(data$genes.info)
```

```
      Chromosome        Genenames          SNPnames             Position
  Min.      :  1.000   PCSK6     : 74   rs1002788   :   1   Min.      :   7881078
  1st Qu.   :  6.000   TXNDC5    : 69   rs1005753   :   1   1st Qu.   :  11712674
  Median    :  8.000   DNAH9     : 41   rs1006273   :   1   Median    :  47863803
  Mean      :  9.962   CA1       : 38   rs10152164  :   1   Mean      :  52968484
  3rd Qu.   : 15.000   VDR       : 19   rs10184179  :   1   3rd Qu.   :  97191582
  Max.      : 17.000   Gc        : 12   rs1032551   :   1   Max.      : 123157722
                       (Other)   : 59   (Other)     : 306
```

2.2 Phenotype importation

Similar to functions introduced to analyze a single pair of genes (see vignette "Statistical analysis of the interaction between a pair of genes."), the case-control status is stored in a numeric or a factor vector with exactly two distinct values. If the phenotype is saved in a separate file in table form, it can thus be imported simply by using the read.table such as for example:

```
R> Y <- read.table(system.file("/extdata/response.txt",package="GGItest"),sep=";")
```

If the case-control status is provided in the ped file, it can be uploaded as follows:

```
R> Y <- data$status
```

2.3 Data filtering

Before performing the statistical analysis, it is very common to remove some SNPs in order to improve the quality of the data. Such a cleaning step can be performed in our **GeneGeneInteR** package by using the function snpMatrixScour. snpMatrixScour aims at modifying a SnpMatrix object by removing SNPs that does not meet criteria regarding the Minor Allele Frequency (MAF), deviation to Hardy-Weinberg Equilibrium (HWE) and the proportion of missing values. In the following example, SNPs with MAF lower than 0.05 or SNPs with p-value for HWE lower than 0.001 or SNPs with a call rate lower than 0.9 are removed from the object data.

```
R> data <- snpMatrixScour(data$snpX,genes.info=data$genes.info,min.maf=0.05,
        min.eq=1e-3,call.rate=0.9)
```

The following lines show that the dataset now contains only 209 SNPs, meaning that 103 SNPs have been filtered out.

```
R> data$snpX
A SnpMatrix with 429 rows and 209 columns
Row names:  H97 ...  RA345
Col names:  rs10510123 ...  rs4328262
```

Since the use of stringent filters could lead to the elimination of all SNPs within a gene, care has to be taken during the filtering step. However, in such a situation the gene without SNPs is removed from the dataset and a warning message is provided for the user.

In other situations, the user might be interested in performing the analysis on a predefined subset of SNPs. For that purpose, the select.snps function provides three options to extract of collection of SNPs by specifying the argument select that should be one of the following:

- a numeric vector with only the column number in the snpMatrix (or row number for genes.info) of each selected SNP. The following line allow the extraction of the 10 first SNPs:
  `R> selec <- select.snps(data$snpX, data$genes.info, select=1:10)`
- a character vector with the names of each selected SNP or each selected gene. The following example is used to extract genes DNAH9 and TXNDC5:
  `R> selec <- select.snps(data$snpX, data$genes.info, c("DNAH9","TXNDC5"))`

- a character vector which elements are position bounds of genes. Each element of the vector is either of the form "begin:end", or "chr:begin:end" if you have to precise the chromosome of the gene. The following code allow to select SNPs from position 101342000 to 101490000 on chromosome 15:
  `R> selec <- select.snps(data$snpX, data$genes.info, c("15:101342000:101490000"))`

2.4 Imputation

Since our pipeline of analysis does not handle with missing values, SNPs filtering as well as SNPs selection can help removing missing data. This can be done easily by applying the `snpMatrixScour` with `NA.rate=0` argument. However, in that case, SNPs with an acceptable call rate are also removed and the lost information is likely to be critical. Genotype imputation is then commonly performed to keep most of the informative SNPs in the dataset. Since our genotype data are stored into `SnpMatrix` object, we implement the `imputeSnpMatrix` function that wraps `snp.imputation` and `impute.snps` functions from **snpStats** package. Our `imputeSnpMatrix` function mimics a Leave-One-Out process where missing SNP are imputed for an individual based on a model trained on all other individuals.

In our example, the following lines show that after the filtering step, 844 missing values still remain in the dataset.

```
R> sum(is.na(data$snpX))
[1] 844
```

To impute those missing values, we simply used our `imputeSnpMatrix` function as follows:

```
R> data <- imputeSnpMatrix(data$snpX,data$ genes.info
|------------------------------------------------------------| 100%
```

A simple check of the dataset show that all missing values have been imputed:

```
R> sum(is.na(data$snpX))
[1] 0
```

When the amount of missing values is so important that `snp.imputation` is not able to find a rule of imputation, some missing values may remain. In that case, the user can specify the action to be done thanks to the `om.rem` arguments:

- `om.rem="none"`: leave the dataset as it is,
- `om.rem="SNP"`: remove all SNPs with remaining missing values,
- `om.rem="ind"`: remove all individuals with remaining missing values.

It is noteworthy that removing all SNP is often more parsimonious than removing individuals and allows to get a dataset without any missing values with minimum information-loss.

Although, function `snp.imputation` can calculate accurate rules for imputation, we encouraged the user to first input missing genotype with an external software (such as IMPUTE2 [HDM09]) prior to the importation step.

3 Statistical analysis

The statistical analysis of a set of genes, as implemented in the `GGI` function, consist in performing all possible pairwise tests between two genes. Pairwise tests are conducted by using the `method` argument with one of the ten method detailed in Section **??**. The `GGI` function takes two further mandatory arguments: `Y` the vector of case-control status and `snpX`, a `SnpMatrix` object that store the genotypes for all SNPs. It is assumed that SNPs within the same gene are consecutive in the `snpX` argument. Furthermore, gene information, such as gene ordering and the number of SNPs within each gene, has to be provided either in the `genes.length` or in the `gene.info` argument.

The following line allow the computation of all pairwise tests between the 17 genes of our example dataset with the PCA-based method.

`GGI.res <- GGI(Y=Y, snpX=data`$snpX, genes.info = data`genes.info,method=PCA)`

A REVOIR

The output of the `GGI` function is a squared matrix with M rows and M columns where M is the number of genes in the dataset. As example, the `GGI.res` object generated in the previous example if a 17 matrix. Each

cell of the output matrix is the p-value of the corresponding pairwise test. The pairwise p-values obtained for the 4 first genes (bub3, CA1, CDSN, DNAH9) of our dataset can be observed as follows:

```
R> round(GGI.res$p.value[1:4,1:4],digits=4)
          bub3     CA1    CDSN   DNAH9
bub3    0.0000  0.1684  0.3179  0.1851
CA1     0.1684  0.0000  0.0697  0.0000
CDSN    0.3179  0.0697  0.0000  0.4539
DNAH9   0.1851  0.0000  0.4539  0.0000
```

FAIRE UN SUMMARY!

4 Visualization

Given a $M \times M$ squared matrix of p-values, as obtained from the analysis of M genes with our `GGI` function, results can be visualized through two types of representation: an heatmap-like visualization with the `GGI.plot` function and a network-like representation with the `draw.network` function.

4.1 Heatmap-like visualization

The `GGI.plot` function can be simply used with the matrix of p-values as the single input argument. Figure 1 (a) and (b) show the graphical representation where all pairwise interactions are plotted (Fig. 1 (a)) or only the interaction between the 3 genes CA1, Gc and PADI1 (Fig. 1 (b)).

(a) `R> plot(GGI.res)`

(b)
```
R> plot(GGI.res[c("CA1","Gc","PADI1"),
          c("CA1","Gc","PADI1")])
```

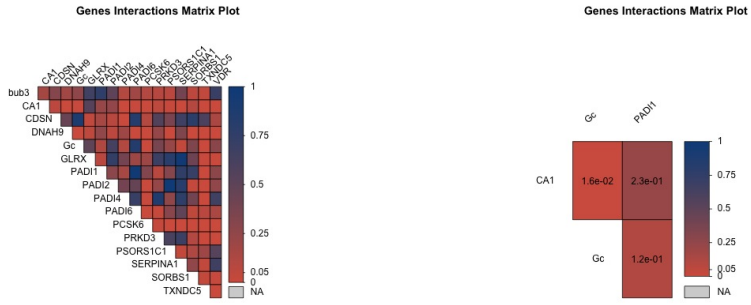

Figure 1: Default output of the function `GGI.plot` considering (a) the whole set of genes or (b) a restricted set of 3 genes.

When the number of genes is below 15, p-values and names are drawn to make matrix reading easier (see Figure 1(b)). However, when the number of genes is larger than 15, p-values are not drawn and gene names are kept while if the number of genes is larger than 25, none of the lvalues or the gene names are displayed (see Figure 1(a)). In that case, the default behavior of the function is to start an interactive process where user can click on a cell of interest to open a tooltip displaying which genes are involved in the selected interaction and the p-value of the interaction test. Tooltips can be closed if user clicks anywhere else than on a cell. This process stops when the user presses the escape button (or terminates the locator procedure in general) or when the user clicks on any place other than a cell when no tooltip window is open.

Several arguments can further be specified to customize the output graphics such as colors (arguments `col` and `NA.col`), width of the bar for colors (argument `colbar.width`), and titles (argument `title`). User can also decide whether p-values (argument `draw.pvals`) and gene names (argument `draw.names`) should be drawn and is allowed to disable the interactivity of plot (argument `interact`). To further improve plot clarity and hence allowing a better interpretation of the results, (1) genes can be ordered according to a hierarchical

clustering (argument `hclust.order`), (2) p-values can be reported in -log10 scale (argument `use.log`) and (3) a threshold can be applied to the p-values in order to distinguish between significant and non-significant interactions (argument `threshold`).

Figures 2 and 3 provide two plots resulting from different sets of arguments passed to the `GGI.plot` function.

```
R> plot(GGI.res,col=c("black","cyan","white"),colbar.width=0.25,title="Interaction
between 17 genes",hclust.order=TRUE,use.log=TRUE,threshold=NULL,NA.col
="#D3D3D3",draw.pvals=FALSE,draw.names=TRUE,interact=FALSE)
```

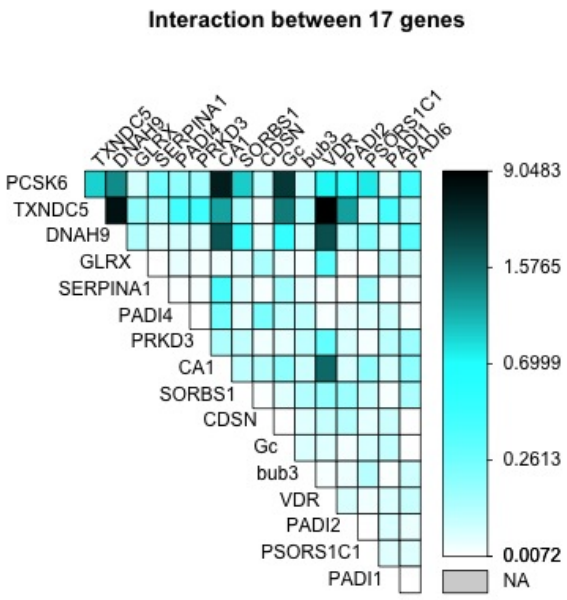

Figure 2: Example of the use of `GGI.plot` arguments when no threshold is applied to the p-values.

4.2 Network-like visualization

The `draw.network` function aims at drawing a graph between genes where two genes are adjacent if the pvalues between these two genes is below a given threshold (argument `threshold` with a default value equal to 0.05). As mandatory argument, the `draw.network` function takes the matrix of p-values as obtained from the `GGI` function (see Section **??**). The display of the network is performed by utilizing the `graph_from_data_frame` from the **igraph** R package [CN06].

Two additional arguments can be used to customize the network. First, user can focus on a specific subset of genes with the argument `genes` and gene not linked to other genes can be removed from the graph with argument `plot.nointer`.

Figure 4 displays the default network obtained with all genes. In figure 5, a subset of only 12 genes have been selected to be the vertices of the graph. However, genes bub3 and PADI1 does not have a pvalue below the threshold of 0.05 with any of the other selected genes. Since the argument `plot.nointer` is set to `TRUE`, the two genes bub3 and PADI1 are not drawn in the resulting network.

In the resulting network, genes bub3 and PADI1 are not drawn

```
R> plot(GGI.res,col=c("black","cyan","white"),colbar.width=0.05,title="Interaction
between 17 genes",hclust.order=TRUE,use.log=FALSE,threshold=0.05,NA.col
="#D3D3D3",draw.pvals=FALSE,draw.names=TRUE,interact=FALSE)
```

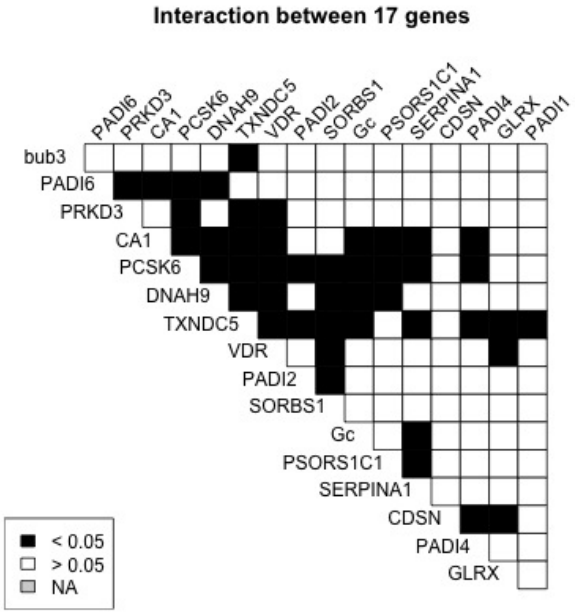

Figure 3: Example of the use of `GGI.plot` arguments when a threshold of 0.05 is applied to the p-values.

References

[CN06] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[CXW+13] X. Chang, B. Xu, L. Wang, Y. Wang, Y. Wang, and S. Yan. Investigating a pathogenic role for txndc5 in tumors. *International Journal of Oncology*, 43(43):1871–1884, 2013.

[HDM09] Bryan N. Howie, Peter Donnelly, and Jonathan Marchini. A flexible and accurate genotype imputation method for the next generation of genome-wide association studies. *PLoS Genetics*, 5(6):e1000529, 06 2009.

```
R> set.seed(1234)
R> plot(GGI.res,method="network")
```

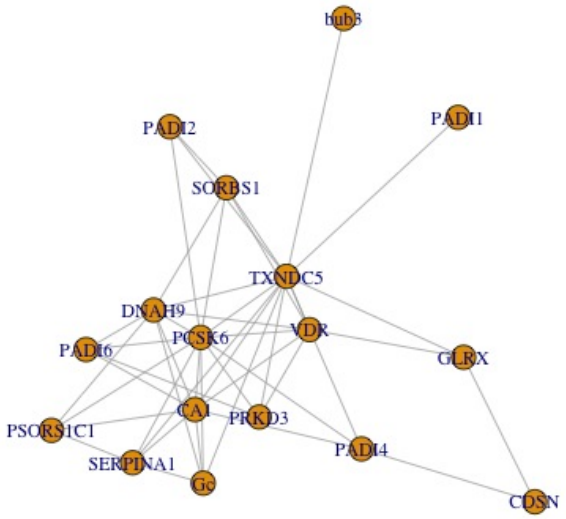

Figure 4: Default output of `draw.network`

```
R> set.seed(1234)
R> plot(GGI.res,method="network",genes=c("bub3","CDSN","Gc","GLRX","PADI1","PADI2","PADI4",
          "PADI6","PRKD3","PSORS1C1","SERPINA1","SORBS1"),
          threshold=0.05,plot.nointer=FALSE)
```

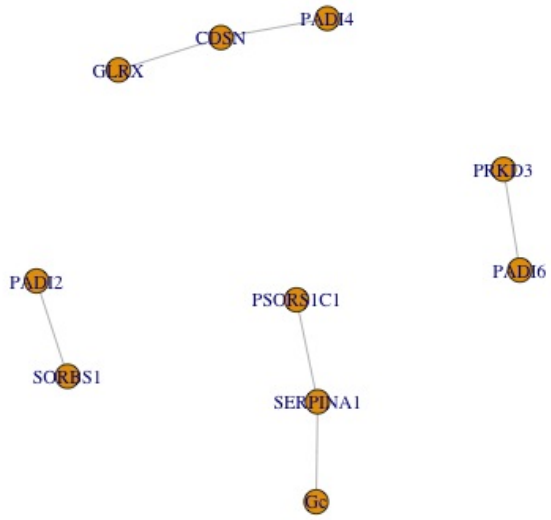

Figure 5: Output of `draw.network` with a customized set of arguments.