cnvGSA: Gene-Set Analysis of (Rare) Copy Number Variants

Joseph Lugo, Daniele Merico and Robert Ziman
The Centre for Applied Genomics
joseph.r.lugo@gmail.com, daniele.merico@gmail.com,
robert.ziman@gmail.com

June 7, 2015

Contents

1 Overview

`cnvGSA` is an `R` package for testing the rare gene-set variant burden in case-control studies of copy number variation (CNV). Earlier versions of `cnvGSA`, or highly similar burden tests, have been used in several high profile studies of rare CNV in neuropsychiatric disorders [1][2][3][4].

Only rare CNV (e.g. at frequency <1%) should be present in the input data: burden tests are not an appropriate method for gene-set analysis of common variants. Gene-sets need to be pre-compiled based on user-curated data or publicly available gene annotations like **Gene Ontology** or pathways.

"Competitive" gene-set over-representation tests are commonly used to analyze differentially gene expressed genes (e.g. Fisher's Exact Test, GSEA), but they are not suitable for rare CNV [1][5]; the most appropriate choice for rare CNV is a "self-contained" burden test with global burden correction implemented by `cnvGSA` or other tools [1].

Global burden correction is very important. For many disorders (including autism and schizophrenia), the disease-affected subjects (i.e. cases) are enriched in large, recurrent CNV [2][3][4][6]. Those CNV are not observed (or observed at very low frequency) in controls and only a minority of their genes may contribute to disease risk. In the absence of global burden correction, many gene-sets would present a biologically unspecific burden, uniquely driven by those larger and recurrent CNV [1]. Global burden correction thus helps to identify specific pathways and functional categories implicated in disease risk by rare CNV.

In `cnvGSA`, subjects are treated as statistical sampling units. Subject-level covariates that may act as confounders can be provided by the user (e.g. sex, ethnicity, CNV genotyping platform, CNV genotyping site, array quality metrics, etc.). The gene-set burden is tested using a logistic regression approach. Two logistic regression models are fit: model A includes the subject-level covariates and a variable quantifying global CNV burden for each subject (total CNV length, or total number of CNV-overlapped genes per subject, etc.); model B includes all variables present in model A, plus the number of CNV-overlapped genes that are members of the gene-set being tested. Presence of significantly higher burden in cases compared to controls for the gene-set of interest is then tested by comparing the two models using a deviance chi-square test, as implemented by `anova.glm`.

In addition, `cnvGSA` provides functions to visualize gene-set burden results and also to export them to the Cytoscape plugin Enrichment Map [7].

2 Workflow outline

The typical analysis workflow consists of:

1. Data import and internal formatting

2. Running the statistical analysis
3. Visualizing the results
4. Exporting to other applications

In the next sections there will be an overview of these steps and associated objects. More detailed documentation can be found in the following sections.

2.1 Data import

Data are typically imported from text files into the object `CnvGSAInput`. Of course directly constructing the objects, or loading them from a R workspace is also possible. The following is a brief description of all the text files that are required.

- `cnvFile` - This file contains all the CNV. Each row contains data for one CNV.
- `geneIDFile` - This file contains primary and secondary geneIDs (e.g. Entrez-gene ID, official symbols).
- `klLociFile` - This file contains all the loci with CNV known to be contributing to disease risk. Used for filtering the data-set before the statistical analysis.
- `klGeneFile` - This file contains genes with CNV known to be contributing to disease risk.
- `phFile` - This file contains the subject information including covariates. Each row contains data for one subject.
- `gsFile` - This file contains a list containing all the gene-sets and their corresponding genes.
- `configFile` - This file contains all parameters for the pre-processing and statistical analysis.

The `cnvGSAIn` function provides a convenient wrapper to import data from text files. A configFile is first loaded; the configFile contains paths to the other files; these paths are read by `cnvGSAIn`, the corresponding files are loaded, tested for consistency, and internal formatting is performed. Here is an example of the usage of `cnvGSAIn` where `cnvGSA.in` is and object of class `CnvGSAInput`. A more detailed example can be found in section 5.1.

Note: `cnvGSA.in` is expecting to find all the required files with their full paths in the configFile.

```
> configFile <- "INPUT PATH TO UPDATED configFile"
> cnvGSA.in  <- CnvGSAInput()
> cnvGSA.in  <- cnvGSAIn(configFile,cnvGSA.in)
```

`CnvGSAInput` is a S4 class acting as a container data structure with slots for each of these required inputs:

```
> library("cnvGSA")
> library("cnvGSAdata")
> slotNames("CnvGSAInput")
[1] "cnvData.ls" "phData.ls" "gsData.ls" "config.ls" "params.ls" "geneID.ls"
```

The input slots should hold the following:

- `cnvData.ls` - CNV data
- `phData.ls` - Phenotype/Covariate data
- `gsData.ls` - Gene-set data
- `config.ls` - paths and file names
- `params.ls` - Test parameters
- `geneID.ls` - Gene ID data

An example input object is available in the companion data package for this vignette:

```
> library("cnvGSA")
> library("cnvGSAdata")
> data("cnvGSA_input_example")
> ls()

[1] "cnvGSA.in"

> class(cnvGSA.in)

[1] "CnvGSAInput"
attr(,"package")
[1] "cnvGSA"

> slotNames(cnvGSA.in)
[1] "config.ls"  "params.ls"  "cnvData.ls" "phData.ls"  "gsData.ls"  "geneID.ls"
```

Each of the slots can be accessed using an accessor function with the same name (e.g. `cnvData.ls(cnvGSA.in)` gets `cnvData.ls`, `gsData.ls(cnvGSA.in)` gets `gsData.ls`, etc.).

2.2 Statistical analysis

The input object is used by `cnvGSA`'s main function, `cnvGSAlogRegTest`, to perform the case-control gene-set burden test and populate the `CnvGSAOutput` object. A full example is shown in section 5:

```
> cnvGSA.out <- CnvGSAOutput()
> cnvGSAlogRegTest(cnvGSA.in,cnvGSA.out)
```

There are many parameters that are mentioned in section 3.6, here are the few that are essential to the statistical analysis:

- `covariates` - The covariates the user would like to include in the regression model. This will be a string listing all the desired covariates with commas in between (ex. SEX,CNV_platform). The user can choose to not include covariates by putting `NONE`.
- `corrections` -The correction models that the user would like to include in the test. Each correction model will provide different results and comparisons can be made to see the difference between all the corrections. More than one correction can be included in the analysis. The corrections included in `cnvGSA` are `no_corr`, `uni_gc`, `tot_l`, and `cnvn_ml` which are all explained in 3.6.2.
- `KL` - Whether or not the user would like to include subjects with CNV overlapping known loci and genes in the statistical analysis. If the user chooses `ALL`, the test will be run twice: once including the subjects with CNV overlapping known loci, and once without them.
- `cnvType` - The type of CNV that the user would like to use for testing.

Note: larger gene-sets have more statistical power; this can be taken into account by separately testing gene-sets with different sizes.

2.3 Visualizing the results

`cnvGSAlogRegTest` produces an object of class `CnvGSAOutput` as its output – likewise a simple S4 class that has slots for each of the output elements.

```
> library(cnvGSA)
> library(cnvGSAdata)
> data("cnvGSA_output_example")
> ls()

[1] "cnvGSA.in" "cnvGSA.out"

> class(cnvGSA.out)

[1] "CnvGSAOutput"
attr(,"package")
[1] "cnvGSA"

> slotNames(cnvGSA.out)
[1] "res.ls"      "gsTables.ls" "gsData.ls"   "phData.ls"
```

The output slots contain the following:

- `res.ls` - List with the results from the statistical test; depending on the parameters in the configFile, this may hold either one data frame, or two data frames, with results including all subjects and/or only subjects without CNV overlapping known loci.
- `gsTables.ls` - Detailed information about the CNV (and their gene annotations) in each tested gene-set.
- `gsData.ls` - Gene-set data, identical to `gsData.ls` in `CnvGSAInput` , except it does not include gene-set that were filtered based on the parameters in the configFile.
- `phData.ls` - Phenotype/covariate data, identical to `phData.ls` in `CnvGSAInput`, except it does not include subjects that were filtered based on the parameters in the configFile.

As with the slots in the input object, each of these can likewise be accessed using an accessor function of the same name (`gsTables.ls()` gets `gsTables.ls`, etc.).

Once the main function (`cnvGSAlogRegTest`) is run, the object `res.ls` will contain all of the results from logistic regression tests (deviance p-values & regression coefficients) along with other useful summary statistics. For a more detailed explanation, see section 4.

2.4 Exporting to other applications

Using the results in `CnvGSAOutput`, you can format and export the output to other applications using functions in the `cnvGSA` package. One application that can be used to aid in the interpretation of the results is the Enrichment Map plugin in Cytoscape. This is used to create network maps and observe clusters of gene-sets. See section 5.2.4 for more information.

3 Loading input data

This section describes the components of the object `CnvGSAInput` in more detail and the text files from which they can be imported. As previously described, the components of the `CnvGSAInput` object – i.e. `cnvData.ls`, `phData.ls`, `gsData.ls`, `config.ls`, `params.ls`, and `geneID.ls` – can be easily filled using the function `cnvGSAIn`:

```
# Create input object
> cnvGSA.in <- CnvGSAInput()
> cnvGSA.in <- cnvGSAIn(configFile,cnvGSA.in)
```

3.1 CNV data

The input CNV data for `cnvGSAIn` relies on providing data with the correct structure and following the specifications described in this section. The text file for the CNV data should have the following columns and should be named accordingly; `SID`, `Chr`, `BP1`, `BP2`, `TYPE`, and `geneID`. All text files that are input should be tab separated. Order does not matter as long

as all the columns are named appropriately. The code generates a data frame in which the columns are named after the columns in the text file.

The following is an example of what the CNV data frame will look like. Here are the columns of the data frame in more detail:

- `$cnv.df` - A data frame containing the CNV. Each row contains data for one CNV:
    - `SID` - ID assigned to the subject in which the CNV was found. It is assumed that the correspondence for each is always 1-to-1 with a subject.
    - `Chr` - Chromosome on which the CNV is located.
    - `BP1` - Start position of the CNV on the chromosome. Genome coordinates should be 1-based.
    - `BP2` - End position of the CNV on the chromosome. Genome coordinates should be 1-based.
    - `TYPE` - CNV type (`"1"` or `"3"` used in example, but can be any other label indicating deletions and gains).
    - `geneID` - Genes overlapped by the CNV, stored in a delimited format inside a character string; e.g. `"54777;255352;84435"` for semicolon-delimited Entrez-Gene identifiers (recommended gene ID system). The gene ID system needs to match the Gene ID object and corresponding file. CNV that are not genic should have an empty string (i.e. `""`) in this column. The user can specify the value separator in `$params.ls`(3.6.2).

For demonstration purposes, the following code will bypass the automated loader and only load the single file. This will also be done for the other required files.

```
> cnvFile <- system.file("extdata","cnv_AGP_demo.txt",package = "cnvGSAdata")
> cnv.df <- read.table (cnvFile, header = T, sep = "\t", quote = "\"",
+ stringsAsFactors = F)
> str(cnv.df,strict.width="cut")
'data.frame': 10666 obs. of  6 variables:
 $ SID   : chr  "1020_4" "1020_4" "1020_4" "1030_3" ...
 $ CHR   : chr  "4" "6" "3" "10" ...
 $ BP1   : int  34802932 35606076 4110452 56265896 64316996 24115481 83206919..
 $ BP2   : int  35676439 35673400 4145874 56361311 64593616 24202712 83239473..
 $ TYPE  : int  3 3 1 1 1 3 1 3 3 3 ...
 $ geneID: chr  NA "2289" NA NA ...
```

After running `cnvGSAIn`, existing components of `$cnvData.ls` are updated and some new components are created:

- `$cnv.df` - Existing data frame containing the CNV. The following columns are added:
    - `CnvKey` - Made from combining CHR, BP1, BP2, TYPE with @ as separator.

- `OlpKL_CNV` - Marks what CNV overlap known loci or genes (see section 3.3), using the package `GenomicRanges` to find the overlaps. For overlap with known loci, it will only mark those CNV that have overlap above the threshold named `$klOlp`, which is described in more detail in section 3.6.2.
- `OlpKL_SID` - Using the information from `$OlpKL_CNV`, marks the subjects that have at least one CNV overlapping a known locus or gene.
- `SubjCnvKey` - Unique identifier of a subject's CNV made from combining `$SID` and `$Cnvkey` using @@ as the separator.
- `CnvLength_ALL` - Total CNV length for all CNV types for each `$SID`.
- `CnvLength_TYPE` - Total CNV length for the user-specified CNV type for each `$SID`.
- `CnvCount_TYPE` - CNV count for the user-specified CNV type for each `$SID`.

- `$cnv2gene.df` - A data frame mapping the CNV to the corresponding genes.

```
# cnvData.ls after running cnvGSAIn
> str(cnvData.ls(cnvGSA.in),strict.width="cut")
List of 2
 $ cnv.df     :'data.frame': 10666 obs. of  15 variables:
  ..$ SID           : chr [1:10666] "1020_4" "1020_4" "1020_4" "1030_3" ...
  ..$ CHR           : chr [1:10666] "4" "6" "3" "10" ...
  ..$ BP1           : int [1:10666] 34802932 35606076 4110452 56265896 6431699..
  ..$ BP2           : int [1:10666] 35676439 35673400 4145874 56361311 64593616..
  ..$ TYPE          : int [1:10666] 3 3 1 1 1 3 1 3 3 3 ...
  ..$ geneID        : chr [1:10666] NA "2289" NA NA ...
  ..$ CnvKey        : chr [1:10666] "4@34802932@3567643@3" "6@35606076@35673400@3"..
  ..$ SEX           : chr [1:10666] "Male" "Male" "Male" "Male" ...
  ..$ Condition     : int [1:10666] 1 1 1 1 1 1 1 1 1 1 ...
  ..$ OlpKL_CNV     : num [1:10666] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ OlpKL_SID     : num [1:10666] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ SubjCnvKey    : chr [1:10666] "1020_4@@4@34802932@3567643@3" "1020_4@@6@356..
  ..$ CnvLength_ALL : num [1:10666] 873508 67325 35423 95416 276621 ...
  ..$ CnvLength_TYPE: num [1:10666] 0 0 35423 95416 276621 ...
  ..$ CnvCount_TYPE : num [1:10666] 0 0 1 1 1 0 1 0 0 0 ...
$ cnv2gene.df:data.frame: 22322 obs. of  9 variables:
  ..$ SubjCnvKey : Factor w/ 10666 levels "1020_4@@3@4110452@4145874@1",..: 1 2 3 4 5 5
  ..$ geneID     : chr [1:22322] NA NA "2289" NA ...
  ..$ SEX        : chr [1:22322] "Male" "Male" "Male" "Male" ...
  ..$ CHR        : chr [1:22322] "3" "4" "6" "10" ...
  ..$ BP1        : int [1:22322] 4110452 34802932 35606076 56265896 64316996 ...
  ..$ BP2        : int [1:22322] 4145874 35676439 35673400 56361311 64593616 ...
  ..$ SID        : chr [1:22322] "1020_4" "1020_4" "1020_4" "1030_3" ...
  ..$ TYPE       : int [1:22322] 1331113333...
  ..$ geneID_TYPE: chr [1:22322] NA NA NA NA ...
```

3.2 Gene ID file

One convenient feature of this package is that the user can use any gene ID system. The user just has to provide a gene ID file that will specify primary identifiers (e.g. EntrezGene IDs) and secondary identifiers (e.g. official symbols); the secondary identifiers are just used as human-readable labels in the output files, but all data-matching operations rely on the primary identifiers.

- `$ann_eg.df` - A data frame with primary and secondary gene identifiers:
  - `geneID` - The gene identifier for the user specified system.
  - `Symbol` - Symbol for the gene.
  - `Name` - Full gene name.

Here is an example of how to make the gene ID file with the Entrez ID system for humans and how to save the data as a tab delimited text file:

```
> library (org.Hs.eg.db)

> ann_eg2sy.df   <- stack (as.list (org.Hs.egSYMBOL));
> names (ann_eg2sy.df)   <- c ("Symbol", "geneID");
> ann_eg2sy.df   <- as.data.frame (lapply (ann_eg2sy.df,   as.character),
+ stringsAsFactors = F)

> ann_eg2name.df <- stack (as.list (org.Hs.egGENENAME));
> names (ann_eg2name.df) <- c ("Name",   "geneID");
> ann_eg2name.df <- as.data.frame (lapply (ann_eg2name.df, as.character),
+ stringsAsFactors = F)

> ann_eg.df <- merge (ann_eg2sy.df, ann_eg2name.df, by = "geneID", all = T)

> str(ann_eg.df,strict.width="cut")
'data.frame': 47721 obs. of  3 variables:
 $ geneID: chr  "1" "10" "100" "1000" ...
 $ Symbol: chr  "A1BG" "NAT2" "ADA" "CDH2" ...
 $ Name  : chr  "alpha-1-B glycoprotein" "N-acetyltransf"..

## Create text file and place path in configFile
> write.table (ann_eg.df, col.names=T, row.names=F, quote=F, sep="\t",
+ file = "geneID_demo.txt")

## There is also a pre-made object found in the data package
> geneIDFile <- system.file( "extdata", "gene_ID_demo.txt", package="cnvGSAdata" )
```

3.3 Known Loci and Genes

This object provides a list of CNV loci in the genome that have a known implication in disease risk, and a list of genes for which CNV has a known implication in disease risk. This object is used to mark CNV and subjects, so that subjects with at least one CNV marked can be removed from the analysis, based on the parameter settings (see `$KL` in section 3.6.2); CNV are marked when they have a matching type and at least one gene in the known genes set, and/or when they have a matching type and a minimum overlap with a locus in the known loci set. Removing marked subjects from the analysis is useful to evalute the "strictly novel" burden signal. This object can be alternatively used as a general purpose mask for genomic loci and genes.

Two text files are required to mark CNV overlapping known loci and known genes. The text files should both be tab separated. One file describes all the loci known to have CNV implicated in disease risk and it should have the following columns and be named accordingly; `CHR`, `BP1`, `BP2`, `TYPE`. The second file contains the genes that are known to be implicated in disease risk. This file should include the following columns and be named accordingly; `geneID`, `TYPE`. Both of these text files will be used to generate data frames with the same columns.

Here is a more detailed explanation of the columns for both of the data frames that will be created:

- `$kl_loci.df` - Data frame that contains all the loci with CNV known to be implicated in disease risk. It should include the following columns:
  - `CHR` - Chromosome on which the CNV is located.
  - `BP1` - Start position of the CNV on the chromosome. Genome coordinates should be 1-based.
  - `BP2` - End position of the CNV on the chromosome. Genome coordinates should be 1-based.
  - `TYPE` - CNV type (example data uses "1" to represent deletions and "0" to represent gains, but be any other label indicating deletions and gains can be used).

```
> klLociFile <- system.file("extdata","kl_loci_AGP_demo.txt",package = "cnvGSAdata")
> kl_loci.df <- read.table (klLociFile,  sep = "", header = T, comment.char = "",
+ quote = "\"", stringsAsFactors = F)
> str(kl_loci.df,strict.width="cut")
'data.frame': 56 obs. of  4 variables:
 $ CHR   : chr  "2" "2" "15" "9" ...
 $ BP1   : int  50539877 51002576 21190624 98998 50990306 50968208 28705540..
 $ BP2   : int  50730546 51157742 26203954 3682923 51222043 51214171 30436163..
 $ TYPE  : int  1 1 3 1 1 1 1 3 1 3 ...
```

- `$kl_gene.df` - Data frame that contains the genes known to be implicated in disease risk:
    - `geneID` - The gene IDs for the genes known to be implicated in disease risk.
    - `TYPE` - CNV type (example data uses "1" to represent deletions and "0" to represent gains, but be any other label indicating deletions and gains can be used).

```
> klGeneFile <- system.file("extdata","kl_gene_AGP_demo.txt",package = "cnvGSAdata")
> kl_gene.df <- read.table (klGeneFile, sep = "", header = T, comment.char = "",
+ quote = "\"", stringsAsFactors = F)
> str(kl_gene.df,strict.width="cut")
'data.frame': 1 obs. of  2 variables:
 $ geneID: int 9378
 $ TYPE  : int 1
```

3.4 Phenotype/Covariate Data

After the CNV data has been loaded the next data structure that needs to be loaded is `phData`, which contains the phenotype/covariate data. Just like the CNV data there is an example of this type of file in the package `cnvGSAdata`.

The text file for the phenotype/covariate data should include the following columns and be named accordingly; `SID`, `Covariates`, and `Condition`. The `Covariates` will depend on what the user includes in the input data. These column names can be whatever is in the data as long as the user specifies them in the configFile. The covariate found in the example data `cnvGSA.in` is "SEX" which describes the biological sex of the subject. This should be a tab separated file and will be used to generate a data frame with the same columns.

Here is a look at the data frame that will be generated as well as a more detailed explanation of the columns that will be included:

- `$ph.df` - A data frame containing the subject information including covariates. Each row contains data for one subject:
    - `SID` - ID assigned to the subject in which the CNV was found. It is assumed that the correspondence for each is always 1-to-1 with a subject.
    - `Covariates` - The covariates that will be included in the analysis. A set of columns, of type character, numeric, integer or factor.
    - `Condition` - Identifies the sample as a case(1) or control(0).

```
> phFile <- system.file("extdata","ph_AGP_demo.txt",package = "cnvGSAdata")
> ph.df <- read.table (phFile, header = T, sep = "\t", quote = "\"",
+ stringsAsFactors = F)
> str(ph.df)
```

```
'data.frame': 10666 obs. of  3 variables:
 $ SID      : chr  "1020_4" "1020_4" "1020_4" "1030_3" ...
 $ SEX      : chr  "Male" "Male" "Male" "Male" ...
 $ Condition: int  1 1 1 1 1 1 1 1 1 1 ...
```

After running `cnvGSAIn`, existing components of `$ph.df` are updated and some new components are created:

- `$ph.df` - Existing data frame containing the subject information including covariates. Additional covariates are generated at corresponding columns:
  - `OlpKL_SID` - Marks the subjects that have at least one CNV overlapping a known locus or gene..
  - `CnvCount_TYPE` - Count for the number of CNV of the user specified type that has occurred for each `$SID`.
  - `CnvTotLength_TYPE` - Total length of all the CNV of the user specified type that occurred for each `$SID`.
  - `CnvMeanLength_TYPE` - Mean length of all the CNV of the user specified type that occurred for each `$SID`.
- `$ph_TYPE.df` - A data frame with the same columns as `$ph.df` plus columns corresponding to gene-set gene counts (each of these columns will show how many genes from the gene-set are overlapped by a CNV of the user-selected type).

```
> str(cnvGSA.in@phData.ls,max.level=2,list.len=10)
List of 2
 $ ph.df:'data.frame': 4227 obs. of  7 variables:
  ..$ SID               : chr [1:4227] "1020_4" "1030_3" "1045_3" "1050_3" ...
  ..$ SEX               : chr [1:4227] "Male" "Male" "Male" "Female" ...
  ..$ Condition         : int [1:4227] 1 1 1 1 0 0 0 0 0 0 ...
  ..$ OlpKL_SID         : num [1:4227] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ CnvCount_TYPE     : num [1:4227] 1 2 1 1 0 0 2 1 3 0 ...
  ..$ CnvTotLength_TYPE : num [1:4227] 35423 372037 32555 41098 0 ...
  ..$ CnvMeanLength_TYPE: num [1:4227] 11808 186018 10852 6850 0 ...
 $ ph_TYPE.df:'data.frame': 4227 obs. of  59 variables:
  ..$ SID               : chr [1:4227] "1020_4" "1030_3" "1045_3" "1050_3" ...
  ..$ SEX               : chr [1:4227] "Male" "Male" "Male" "Female" ...
  ..$ Condition         : int [1:4227] 1 1 1 1 0 0 0 0 0 0 ...
  ..$ OlpKL_SID         : num [1:4227] 0 0 0 0 0 0 0 0 0 0 ...
  ..$ CnvCount_TYPE     : num [1:4227] 1 2 1 1 0 0 2 1 3 0 ...
  ..$ CnvTotLength_TYPE : num [1:4227] 35423 372037 32555 41098 0 ...
  ..$ CnvMeanLength_TYPE: num [1:4227] 11808 186018 10852 6850 0 ...
  ..$ GS1               : int [1:4227] 0 1 0 1 0 0 1 0 1 0 ...
  ..$ GS10              : int [1:4227] 0 1 0 0 0 0 1 0 1 0 ...
  ..$ GS11              : int [1:4227] 0 0 0 1 0 0 0 0 2 0 ...
  .. [list output truncated]
```

3.5 Gene sets

The gene-set data should be one list containing all the gene sets that the user wants to include in the analysis. This list should have gene-set names, which will act as the `$GsIDs` for the later tables and will be the IDs for a certain gene set.

Instead of using a text file to read in the data, it is recommended to save the data as an .RData file. This is because the gene-sets will not be equal in length and cannot be read as a normal data frame. The object containing the gene-sets must be called `gs_all.ls` and will be a list with each entry being the gene-set named and a vector or the corresponding genes in the gene-set. There should also be a separate object that will contain the gene set names. This object must be named `gsid2name.chv` and will be a vector simply containing all the gene-set names.

To create these files it is common to read a **Gene Matrix Transposed file format** (GMT file). The GMT file format is a tab delimited file format that describes gene-sets. In the GMT format, each row represents a gene-set. Here the user can see how to create the gene-set objects using a GMT file:

```
> tmp         <- read.table(filename, sep = "@", comment.char = "", quote = "",
+ stringsAsFactors = F)
> tmp$V1      <- gsub("%KEGG%", "\tKEGG:", tmp$V1)
> tmp$V1      <- gsub("Reactome Pathway", "REACT:", tmp$V1)
> tmp$V1      <- gsub("http://www.broadinstitute.org/gsea/msigdb/cards/BIOCARTA_
+ |BIOCARTA_", "BIOC:", tmp$V1)
> tmp         <- lapply(1:nrow(tmp), function(x) unlist(strsplit(tmp[x,], "\t")))
> names.v     <- unlist(lapply(1:length(tmp), function(x) tmp[[x]][1]))
> IDs         <- unlist(lapply(1:length(tmp), function(x) tmp[[x]][2]))
> genes.ls    <- lapply(1:length(tmp), function(x) tmp[[x]][-c(1,2)])
> genes.ls    <- lapply(genes.ls, setdiff, NA)
> genes.ls    <- lapply(genes.ls, unique)
> gs_demo.ls  <- list()
> gs_demo.ls$gsid2name.chv <- names.v
> gs_demo.ls$gs_all.ls     <- genes.ls
> names(gs_demo.ls$gsid2name.chv) <-  IDs
> names(gs_demo.ls$gs_all.ls)     <-  IDs
> gsid2name.chv <- gs_demo.ls$gsid2name.chv
> gs_all.ls     <- gs_demo.ls$gs_all.ls
```

Here is an example of pre-compiled gene-set lists found in the data package:

```
> library(cnvGSAdata)
> data("gs_data_example") # gs_all.ls and gsid2name.chv will be loaded
```

```
> str(gs_all.ls,max.level=1,list.len=5)
List of 51
 $ BspanVHM_PreNat             : int [1:3038] 7105 4800 81887 6405 1595 3927 ...
 $ BspanVHM_EqlNat             : int [1:3038] 8813 54467 9957 23072 8379 56603 ...
 $ BspanVHM_PstNat             : int [1:3131] 2519 2729 90529 57185 22875 572 ...
 $ BspanVH_lg2rpkm4.74         : int [1:4600] 7105 4800 57185 1595 572 9957 3927 ...
 $ BspanHM_lg2rpkm3.21         : int [1:4605] 8813 2519 2729 90529 81887 22875 ...
  [list output truncated]
> str(gsid2name.chv)
 Named chr [1:51] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" ...
 - attr(*, "names")= chr [1:51] "BspanVHM_PreNat" "BspanVHM_EqlNat" ...

 ## Location of the pre-compiled gene-set file
 > gsFile <- system.file( "data", "gs_data_example.RData", package="cnvGSAdata" )
```

After `cnvGSAIn` is run the `gsData.ls` slot will have components added to it. To see an example of `gsData.ls`, load the `CnvGSAInput` object included in the data package. The list elements are:

- `$gs_info.df` - A data frame with the columns `$GsKey`, `$GsID` ,`$GsName`, `$GsSize`; `$GsID` is the primary gene-set identifier outside `cnvGSA`, while `$GsKey` is the primary gene-set identifier inside `cnvGSA`.
- `$gs_sel_U.df` - A data frame with the columns `$GsKey`, `$geneID`, `$GsName` mapping each gene-set `$GsKey` with the corresponding `$geneID`. It splits all the genes apart so that there is a one to one relationship. This data frame is derived from `$gs_all.ls`.
- `$gs_colnames_TYPE.chv` - A single character vector that contains all the `$GsKey` of the gene-set selected by the user for the statistical test.
- `$geneCount.tab` - A table object, with gene counts for subjects with different `$Condition` (e.g. cases and controls).
- `$gs_all.ls` - A list containing all the gene-sets and their corresponding genes. The names for this list are the `$GsID`s. This list is identical to the list found in the gene-set data the user inputs.

```
> library(cnvGSAdata)
> data("cnvGSA_input_example")
> str(cnvGSA.in@gsData.ls,strict.width="cut",list.len = 10)
List of 5
 $ gs_info.df          :'data.frame': 52 obs. of  4 variables:
  ..$ GsKey : chr [1:52] "GS1" "GS2" "GS3" "GS4" ...
  ..$ GsID  : chr [1:52] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" ..
  ..$ GsName: chr [1:52] "BspanVHM_PreNat" "BspanVHM_EqlNat" "BspanVHM_PstNat" ..
```

```
 ..$ GsSize: int [1:52] 3037 3037 3131 4600 4605 4596 4601 4131 4185 4190 ...
$ gs_sel_U.df         :'data.frame': 120486 obs. of  4 variables:
 ..$ GsKey : Factor w/ 52 levels "GS1","GS10","GS11",..: 1 1 1 1 1 1 1 1 1 1 ...
 ..$ geneID   : chr [1:120486] "7105" "4800" "81887" "6405" ...
 ..$ GsID  : chr [1:120486] "BspanVHM_PreNat" "BspanVHM_PreNat" "BspanVHM_PreNat" ..
 ..$ GsName: chr [1:120486] "BspanVHM_PreNat" "BspanVHM_PreNat" "BspanVHM_PreNat" ..
$ gs_colnames_TYPE.chv: chr [1:51] "GS1" "GS10" "GS11" "GS12" ...
$ geneCount.tab       : 'table' int [1:6203, 1:2] 1 1 0 0 0 0 0 0 0 0 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ geneID   : chr [1:6203] "1" "10" "10003" "100033413" ...
 .. ..$ Condition: chr [1:2] "0" "1"
$ gs_all.ls           :List of 51
 ..$ BspanVHM_PreNat             : int [1:3038] 7105 4800 81887 6405 1595 3927 ...
 ..$ BspanVHM_EqlNat             : int [1:3038] 8813 54467 9957 23072 8379 56603 ...
 ..$ BspanVHM_PstNat             : int [1:3131] 2519 2729 90529 57185 22875 ...
 ..$ BspanVH_lg2rpkm4.74         : int [1:4600] 7105 4800 57185 1595 572 9957 ...
 ..$ BspanHM_lg2rpkm3.21         : int [1:4605] 8813 2519 2729 90529 81887 ...
 ..$ BspanML_lg2rpkm0.93         : int [1:4596] 57147 55732 2268 3075 889 5893 ...
 ..$ BspanLA_lg2rpkm.MIN         : int [1:4601] 64102 1080 26 51384 6542 ...
 ..$ GI_q4                       : chr [1:4131] "1" "8086" "65985" "343066" ...
 ..$ GI_q3                       : chr [1:4185] "29974" "53947" "13" "344752" ...
 ..$ GI_q2                       : chr [1:4190] "2" "195827" "79719" "14" ...
 .. [list output truncated]
```

3.6 Parameters and settings (configFile)

Running the association tests requires that the user provide a configFile to control how the tests will be run. The script will go through the configFile that the user inputs and use he parameters to specify how different functions in the package will run. There are four blocks in the configFile that serve different purposes. The following section will go through each block and explain how to fill in each value.

3.6.1 config.ls configuration

The `config.ls()` accessor function can be used to access the data in the slot. The main association test procedure accepts several files and this list will contain all the paths to the necessary input data.

The fields for `config.ls` are as follows:

- `cnvFile` - File name for the CNV file user would like to read in.
- `phFile` - File name for the phenotype/covariate file user would like to read in.

- `geneIDFile` - File name for the file describing the ID system used for the data.
- `klGeneFile` - File name for the file containing the genes with CNV known to be implicated in disease risk.
- `klLociFile` - File name for the file with containing the loci with CNV known to be implicated in disease risk.
- `gsFile` - File name for the gene-set data the user would like to read in.
- `outputPath` - Path for the folder where the user would like to output all the results.
- `geneListFile` - Path for the file containing the list of genes that the user would like to restrict the statistical burden analysis to. All genes are used if this is not provided.

```
> str( config.ls(cnvGSA.in) )
List of 9
 $ cnvFile     : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/
                 cnv_AGP_demo.txt"
 $ phFile      : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/
                 ph_AGP_demo.txt"
 $ geneIDFile  : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/
                 geneID_demo.txt"
 $ klGeneFile  : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/
                 kl_gene_AGP_demo.txt"
 $ klLociFile  : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/
                 kl_loci_AGP_demo.txt"
 $ gsFile      : chr "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/
                 gs_data_demo.RData"
 $ outputPath  : chr "/Users/josephlugo/Documents/R/PGC2_test/Restructure_
                 Results/8.5/testResults/"
 $ geneListFile: chr ""
 $ config.df   :'data.frame': 37 obs. of  2 variables:
  ..$ param: chr [1:37] "cnvFile" "phFile" "geneIDFile" "klGeneFile" ...
  ..$ value: chr [1:37] "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/ ...
```

3.6.2 params.ls configuration

The params.ls() accessor function can be used to access the data in the slot. The main association test procedure accepts several parameters which will specify how the user would like to run the logistic regression tests.

The fields for `params.ls` are as follows:

- `Kl` - Whether or not the user would like to include subjects marked for known loci and genes in the statistical analysis. If the user chooses `ALL`, the test will be run twice: once including the subjects marked for known loci, and once without them. The default value for this parameter is `ALL`. Possible values are:
    - `YES` - Include subjects marked as having CNV overlapping known loci or known genes
    - `NO` - Do not include subjects marked as having CNV overlapping known loci or known genes
    - `ALL` - Produce results from both `YES` and `NO`. Will create two separate data frames.
- `projectName` - Suffix used in the output file names.
- `gsUset` - Gene-set representing the universe-set for the corresponding global burden correction. If not provided, all genes mapped at least once to a CNV are used to define this set.
- `cnvType` - The type of CNV that the user would like to use for testing. Possible values are:
    - Any value in the `$TYPE` column which may include ("1", "0", "LOSS", "GAIN", etc.). The user may have multiple levels for `$TYPE` but they should only choose one for the analysis.
    - `ALL` - Includes all types of CNV in the data.
- `covariates` - The covariates the user would like to include in the regression model. This will be a character vector listing all the desired covariates with commas in between (ex. "SEX,CNV_platform"). The user can choose not to include covariates by putting `NONE`.
- `klOlp` - The minimum percentage overlap between an observed CNV and the interval corresponding to a known locus.
- `corrections` - The correction models that the user would like to include in the logistic regression tests. Each correction model will provide different results and comparisons can be made to see the difference between all the different corrections. More than one correction can be included in the analysis. Enter as a list of corrections with commas in between (ex. "no_corr,uni_gc"). The default value for this is to use all the correction models. Possible values are:
    - `no_corr` - Do not include any correction in the analysis.
    - `uni_gc` - Add the universe gene count correction model.
    - `tot_l` - Add the CNV total length correction model.
    - `cnvn_ml` - Add the mean CNV length and CNV count correction model.
    - `ALL` - Include all the correction models.

- `geneSep` - The separator that will be used to separate the gene identifiers in the `$geneID` column of the `cnv.df` object. The default value for this is ";".
- `geneSetSizeMax` - The maximum number of genes that the gene sets are allowed to have. The default value for this is 1500.
- `geneSetSizeMin` - The minimum number of genes that the gene sets are allowed to have. The default value for this is 25.
- `filtGs` - Once the `$geneSetSizeMax` and `$geneSetSizeMax` parameter's have been set the user can choose whether or not to include this filter in the analysis. The default value for this is `NO`. Possible values are:
    - `YES` - Include the filter in the analysis.
    - `NO` - Do not include the filter in the analysis.
- `covInterest` - What covariate will be used to report stratified case and control subject counts.
- `eventThreshold` - When reporting subject counts split by covariate value and condition, what is the minimum number of genes of the gene-set that need to be overlapped by a type-matched CNV for the subject to be counted. The default value for this is 1.
- `flevels` - The maximum number of levels the factor `$covInterest` is allowed to have. The default value for this is 10.
- `cores` - User will set the number of cores they want to use for the parallelization. The default value for this is the maximum number of cores on the machine.
- `parallel` - Option to run the association test in parallel or not. The default value is `YES`. The options include:
    - `YES` - Allow parallelization in the logistic regression tests.
    - `NO` - Do not allow parallelization in logistic regression tests.
- `CNVevents` - The minimum number of overlapped genes that a gene-set needs to be included in the analysis. Checks for gene-sets that have at least this many (user specified) overlapped genes.

`NOTE:` Parallelization is known to have issues on Windows machines. If there are errors, the user must run the logistic regression tests sequentially (i.e choose `NO` for `parallel` parameter).

```
> str( params.ls(cnvGSA.in) )
List of 17
 $ Kl             : chr "ALL"
 $ projectName    : chr "NS"
```

```
$ gsUSet         : chr ""
$ cnvType        : chr "1"
$ covariates     : chr "SEX"
$ klOlp          : num 0.5
$ corrections    : chr [1:4] "no_corr" "uni_gc" "tot_l" "cnvn_ml"
$ geneSep        : chr ";"
$ geneSetSizeMin: num 25
$ geneSetSizeMax: num 1500
$ filtGs         : chr "NO"
$ covInterest    : chr "SEX"
$ eventThreshold: num 1
$ fLevels        : num 10
$ cores          : num 4
$ parallel       : chr "YES"
$ CNVevents      : chr "5"
```

3.6.3 Visualization configuration

This package includes the ability to create visualizations from the output of the main association test. It is recommended to use this functionality for a small number of selected gene-sets (i.e. 5-20). The function `f.makeViz` is described in the workflow example and it takes in these fields:

- `FDRThreshold` - Will only include those gene-sets that have a FDR below this threshold
- `plotHeight` - Sets the bottom margin of the plot by using the `mar` option in the `par` function included in R. This is useful to ensure that the gene-set names fit in the plot. The default value is 13.
- `gsList` - Text file containing list of gene-sets that the user is interested in looking at plotting. Should be comma separated with no space in between (ex. "PSD_BayesGrant_fullset, FMR1_Targets_Darnell").
- `outputPathViz` - Path for the output of the visualization script.
- `labelSize` - How long the user would like the labels to be on the plots. This is the expansion factor for the axis names that uses the `cex` function in R. The default value is 0.7.
- `correctionViz` - Corrections used in creating the plots. Multiple corrections may be selected and additional plots will be created accordingly. The default value for this is `ALL`. The valid options include:
  - `no_corr` - Do not include any correction model.
  - `uni_gc` - Add the universe count correction model.

- `tot_l` - Add the total count correction model.
- `cnvn_ml` - Add the mean length and number correction model.
- `ALL` - Include all the correction models.

- `cnvGSAViz` also takes in some fields that are defined in `params.ls`. It is not necessary to restate the following fields:
  - `Kl` - Look at description in previous section 3.6.2.
  - `cnvType` - Look at description in previous section 3.6.2.

3.6.4 Enrichment configuration

Along with the visualization function, there is also a function that will create the files necessary to create an Enrichment Map. This function takes in these fields:

- `pVal` - The p-value the user would like to use for the enrichment map. The default value is `Pvalue_U_dev`.
- `FDR` - The FDR value the user would like to use for the enrichment map. The default value is `FDR_BH_U`.
- `coeff` - The coefficient the user would like to use for the enrichment map. The default value is `Coeff_U`.
- `keepCoeff` - Whether or not the user would like to keep the gene-sets with negative regression coefficients. Without global burden correction, negative regression coefficients are present when gene-sets have a higher burden in controls than cases; after global burden correction negative regression coefficients can just indicate "unspecific" burden. The default value is `YES`.
- `outputPathEnr` - Path for the output of the `f.enrFiles` function.
- `minCaseCount` - The minimum number of cse subjects that must have a CNV overlapping the gene. The default value is 1.
- `maxControlCount` - The maximum number of controls each gene is allowed to have. The default value is 0.
- `minRatio` - The minimum case/control ratio each gene is allowed to have. The default value is 0.
- `filtGsEnr` - Whether or not the user would like to filter the gene-sets allowed in the GMT file created. The default value is `YES`.
- `f.enrFiles` also takes in some fields that are defined in `params.ls`. It is not necessary to restate the following fields:
  - `Kl` - Look at description in previous section 3.6.2.
  - `cnvType` - Look at description in previous section 3.6.2.

3.6.5 Loading test parameters from a file

To make it easier to integrate the association test into a larger bioinformatics pipeline, it is convenient to read in the parameters from an external source such as a text file. There is a template file that the user can look at which is included in the package. One such implementation is to record each parameter on its own line using the tab separated syntax:

```
param value
###############
# LogReg Config
###############
## CONFIG.LS ##
cnvFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/cnv_AGP_demo.txt"
phFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/ph_AGP_demo.txt"
geneIDFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/geneID_demo.txt"
klGeneFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/kl_gene_AGP_demo.txt"
klLociFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/kl_loci_AGP_demo.txt"
gsFile "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/gs_data_demo.RData"
outputPath "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/testResults/
geneListFile ""
## PARAMS.LS ##
Kl "ALL"
projectName "NS"
gsUSet ""
cnvType 1
covariates "SEX"
klOlp 0.5
corrections "no_corr,uni_gc,tot_l,cnvn_ml"
geneSep ";"
keySep "@"
geneSetSizeMax 1500
geneSetSizeMin 25
filtGs "NO"
covInterest "SEX"
eventThreshold 1
fLevels 10
cores   4
parallel    "YES"
CNVevents   5
############
# Viz Config
############
FDRThreshold 0.1
plotHeight 13
```

```
gsList "/Users/josephlugo/Documents/R/PGC2_test/cnvGSAData/gsList.txt"
outputPathViz "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/Viz/"
labelSize 0.7
correctionViz   "uni_gc,tot_l,cnvn_ml"
############
# ENR Config
############
pVal "Pvalue_U_dev"
FDR "FDR_BH_U"
coeff "Coeff_U"
keepCoeff "YES"
filtGsEnr "NO"
outputPathEnr "/Users/josephlugo/Documents/R/PGC2_test/Restructure_Results/8.5/enr/"
minCaseCount 0
minControlCount 0
minRatio 0

## Find a template of the configFile
> configFileTmp <- system.file( "extdata", "configFile.txt", package="cnvGSA" )
```

To update the parameters in the input object `CnvGSAInput` the user simply needs to change the configFile and then run the function below. Once the function is run, all the changed parameters will be updated in the input object `CnvGSAInput`.

```
> cnvGSA.in <- f.readConfig(configFile,cnvGSA.in)
```

4 Interpreting the results from the statistical analysis

4.1 Description of res.ls results

After the main function `cnvGSAlogRegTest` is run, there will either be one data frame in `res.ls`, or two data frames in `res.ls`. There will be one data frame if the user sets `$KL` to `YES` or `NO`. There will be two data frames if the user sets `$KL` to `ALL`. As previously described in section 3.6.2, the `$KL` value will specify whether or not the user would like to include subjects marked for known loci and genes in the statistical analysis.

For the example data that is included in this `cnvGSAdata`, the `$KL` parameter was set to ALL so there are two objects in the list. Taking a look first at `res.ls`:

```
> library(cnvGSAdata)
> data(cnvGSA_output_example)
> str(cnvGSA.out@res.ls,max.level=1,list.len = 2)
List of 2
```

```
$ covAll_chipAll_1_KLy.df:'data.frame': 50 obs. of  39 variables:
 .. [list output truncated]
$ covAll_chipAll_1_KLn.df:'data.frame': 50 obs. of  39 variables:
 .. [list output truncated]
```

The data frames `$covAll_chipAll_1_KLy.df` and `$covAll_chipAll_1_KLn.df` contain the actual logistic regression results. `$covAll_chipAll_1_KLy.df` has the results for the tests including subjects that are marked for known loci and genes, whereas `$covAll_chipAll_1_KLn.df` has the results without subjects that are marked for known loci and genes. The CNV type is specified by the user and will vary for different data sets. The types used in the sample data are "1"(LOSS) and "3"(GAIN).

Each row contains results for a single gene-set. The columns are as follows:

- `GsID`, `GsName`, and `GsSize` show the gene-set's identifier, name, and number of member genes respectively.
- `Coeff`, `Coeff_U`, `Coeff_TL`, and `Coeff_CNML` show the gene-set's regression coefficients for each of the different correction models used. This represents the log odds of being a case in the exposed vs. unexposed, adjusted for the covariates and whichever correction models were specified.
- `Pvalue_glm`, `Pvalue_U_glm`, `Pvalue_TL_glm`, and `Pvalue_CNML_glmL` show the gene-set's p-values for each of the different correction models used. These p-values were obtained using the coefficient significance from the default test of the function `glm` in R.
- `Pvalue_dev`, `Pvalue_U_dev`, `Pvalue_TL_dev`, and `Pvalue_CNML_dev` show the gene-set's p-values for each of the different correction models used.These p-values were obtained using a chi-square deviance test.
- `Pvalue_dev_s`, `Pvalue_U_dev_s`, `Pvalue_TL_dev_s`, and `Pvalue_CNML_dev_s` are obtained by taking the log of the deviance p-values and multiplying them by the sign of the coefficient (these are useful for sorting).
- `FDR_BH`, `FDR_BH_U`, `FDR_BH_TL`, and `FDR_BH_CNML` show FDR values using four methods. `FDR_BH` is the Benjamini-Hochberg (BH) false discovery rate (FDR) for the p-values calculated using the no correction model. `FDR_BH_U`, `FDR_BH_TL`, and `FDR_BH_CNML` are the same FDR but with the `uni_gc`, `tot_l`, and `cnvn_ml` correction respectively.
- `CASE_g1n`, `CASE_g2n`, `CASE_g3n`, `CASE_g4n`, and `CASE_g5n` show the percentage of samples from the cases that each gene-set has at least 1,2,3,4 and 5 events in.
- `CTRL_g1n`, `CTRL_g2n`, `CTRL_g3n`, `CTRL_g4n`, and `CTRL_g5n` show the percentage of samples from the controls that each gene-set has at least 1,2,3,4 and 5 events in.
- `CASE_gTT` and `CTRL_gTT` show the total number of cases and controls there are.
- `CASE_Female` and `CASE_male` show the percentage of samples from the cases that must be >= a user specified number of events and also is part of the SEX covariate. This will change depending on which covariate the user would like to see in the output.

The structure of both data frames is shown in the listing below:

```
> str( cnvGSA.out@res.ls$covAll_chipAll_1_KLy.df )
'data.frame': 50 obs. of  39 variables:
 $ GsID              : chr  "FMR1_Targets_Darnell" "PSD_BayesGrant_fullset" ...
 $ GsName            : chr  "FMR1_Targets_Darnell" "PSD_BayesGrant_fullset" ...
 $ GsSize            : int  840 1407 4600 1230 3131 4605 927 1424 3037 116 ...
 $ Coeff             : num  1.461 1.03 0.41 0.821 0.392 ...
 $ Pvalue_glm        : num  9.19e-09 2.15e-08 5.92e-07 2.09e-07 8.70e-07 ...
 $ Pvalue_dev        : num  8.67e-13 2.08e-12 4.45e-10 9.15e-09 1.96e-09 ...
 $ Pvalue_dev_s      : num  12.06 11.68 9.35 8.04 8.71 ...
 $ FDR_BH            : num  4.33e-11 5.19e-11 7.41e-09 7.62e-08 2.29e-08 ...
 $ Coeff_U           : num  1.328 0.935 0.366 0.647 0.337 ...
 $ Pvalue_U_glm      : num  9.33e-07 2.81e-06 2.93e-04 2.88e-04 6.15e-04 ...
 $ Pvalue_U_dev      : num  4.19e-08 1.73e-07 7.35e-05 1.71e-04 2.82e-04 ...
 $ Pvalue_U_dev_s    : num  7.38 6.76 4.13 3.77 3.55 ...
 $ FDR_BH_U          : num  2.09e-06 4.33e-06 9.19e-04 1.71e-03 1.91e-03 ...
 $ Coeff_TL          : num  1.349 0.944 0.358 0.71 0.344 ...
 $ Pvalue_TL_glm     : num  2.33e-07 7.15e-07 4.46e-05 2.49e-05 3.67e-05 ...
 $ Pvalue_TL_dev     : num  4.63e-10 2.00e-09 5.31e-07 6.53e-06 1.86e-06 ...
 $ Pvalue_TL_dev_s   : num  9.33 8.7 6.27 5.19 5.73 ...
 $ FDR_BH_TL         : num  2.31e-08 4.99e-08 8.85e-06 5.44e-05 2.32e-05 ...
 $ Coeff_CNML        : num  1.375 0.967 0.366 0.753 0.353 ...
 $ Pvalue_CNML_glm   : num  9.72e-08 2.55e-07 1.66e-05 5.06e-06 1.20e-05 ...
 $ Pvalue_CNML_dev   : num  8.52e-11 2.51e-10 1.02e-07 8.20e-07 3.03e-07 ...
 $ Pvalue_CNML_dev_s: num  10.07 9.6 6.99 6.09 6.52 ...
 $ FDR_BH_CNML       : num  4.26e-09 6.27e-09 1.70e-06 6.83e-06 3.79e-06 ...
 $ CASE_g1n          : num  3.86 5.34 9.25 4.92 7.24 ...
 $ CTRL_g1n          : num  0.899 1.627 4.88 2.269 4.409 ...
 $ CASE_g2n          : num  0.74 1.16 2.7 1.22 2.7 ...
 $ CTRL_g2n          : num  0.0428 0.1284 0.8134 0.0856 0.8134 ...
 $ CASE_g3n          : num  0.529 0.687 1.639 0.687 1.745 ...
 $ CTRL_g3n          : num  0 0 0.342 0 0.171 ...
 $ CASE_g4n          : num  0.37 0.582 1.163 0.106 1.322 ...
 $ CTRL_g4n          : num  0 0 0.1712 0 0.0856 ...
 $ CASE_g5n          : num  0.106 0.423 0.793 0.106 1.005 ...
 $ CTRL_g5n          : num  0 0 0.1284 0 0.0856 ...
 $ CASE_gTT          : num  1891 1891 1891 1891 1891 ...
 $ CTRL_gTT          : num  2336 2336 2336 2336 2336 ...
 $ CASE_Female       : num  6.3 6.3 10.37 5.19 7.04 ...
 $ CASE_Male         : num  3.45 5.18 9.07 4.87 7.28 ...
 $ CTRL_Female       : num  0.962 1.603 4.567 2.484 5.048 ...
 $ CTRL_Male         : num  0.827 1.654 5.239 2.022 3.676 ...
```

5 Full workflow example: case-control analysis of rare CNV from the Pinto et al. 2014 ASD study

5.1 Loading the data and running the association test

The following code performs an analysis of approximately 5500 CNV from 2000 subjects. The CNV data-set consists of rare CNV as described in Pinto et al., AJHG 2014 [4], and the gene-sets are a collection imported from the Gene Ontology, Biocarta, KEGG, NCI, Reactome.

The workflow is based on the data found in the data package, refer to section 3 to find the paths to the example data. The user can use these paths to fill in the template configFile found in this package.

```
> library( "cnvGSA" )
> library( "cnvGSAdata" )

##
## Update the configFile
##

## Find a template of the configFile
> configFileTmp <- system.file( "extdata", "configFile.txt", package="cnvGSA" )

##
## Create the input object
##

> configFile <- "INPUT PATH TO UPDATED configFile"
> cnvGSA.in  <- CnvGSAInput()
> cnvGSA.in  <- cnvGSAIn(configFile,cnvGSA.in)

## If this error is seen:
> Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings,  :
  line 2 did not have 2 elements
## Check the configFile to make sure that everything but the comments are tab
## separated.

##
## Run association test and save the output
##

> cnvGSA.out <- CnvGSAOutput()
```

```
> cnvGSA.out <- cnvGSAlogRegTest(cnvGSA.in, cnvGSA.out)
```

5.2 Reviewing the results

As stated in the workflow outline, the output object is a simple S4 class containing a slot for each output data structure:

```
> slotNames(cnvGSA.out)
[1] "res.ls"      "gsTables.ls" "gsData.ls"   "phData.ls"
```

Similar to the input, each of components are a list structure containing further data structures: `res.ls` contains the the results of the logistic regression tests, and `gsTables.ls` contains the gene-set tables for each gene-set. `gsData.ls` contains all the gene-set data, and `phData.ls` contains all the phenotype/covariate data.

5.2.1 Association test results

All the results from the association test are stored in the `res.ls` object after the main function `cnvGSAlogRegTest` is run. For an in depth look at the `res.ls` object results, refer to section 4.1.

5.2.2 Detailed analysis of gene-set CNV and genes

As a further aid in understanding the CNV and genes contributing to the association results, there exists `gsTables.ls` – a list of data frames, one for each of the gene-sets, containing information about the CNV and corresponding genes affecting the gene-set:

```
## There is a pre-made object available in the data package
> gsTables.ls <- gsTables.ls(cnvGSA.out)
> str( gsTables.ls(cnvGSA.out), max.level=1, list.len=5 )
List of 52
 $ BspanHM_lg2rpkm3.21         :'data.frame': 1409 obs. of  11 variables:
  .. [list output truncated]
 $ BspanLA_lg2rpkm.MIN         :'data.frame': 2700 obs. of  11 variables:
  .. [list output truncated]
 $ BspanML_lg2rpkm0.93         :'data.frame': 2107 obs. of  11 variables:
  .. [list output truncated]
 $ BspanVH_lg2rpkm4.74         :'data.frame': 955 obs. of  11 variables:
  .. [list output truncated]
 $ BspanVHM_EqlNat             :'data.frame': 834 obs. of  11 variables:
  .. [list output truncated]
  [list output truncated]
```

To see `gsTables.ls` the user can either look at the data included in the `cnvGSAdata` package or run the `cnvGSAgsTables` function provided in this package. This function will create all the gene-set tables for each gene-set:

```
## This will populate the gsTables.ls slot in cnvGSA.out
> cnvGSA.out  <- cnvGSAgsTables(cnvGSA.in,cnvGSA.out)
> gsTables.ls <- cnvGSA.out@gsTables.ls
```

The structure of the data frame for a particular gene-set is similar to that of `cnvData.ls$cnv.df` in the input with the exception of the `$Symbol` and `$Symbol_TYPE` columns, which show the gene symbols that match the primary identifier:

```
> str(gsTables.ls[[2]],strict.width="cut",max.level=2)
'data.frame': 2700 obs. of  11 variables:
 $ SID        : chr  "110036001873_" "110036001873_" "110036001873_" ...
 $ CHR        : chr  "1" "17" "2" "13" ...
 $ BP1        : int  1193801 30708148 87214673 69378175 14286493 335515 13523286..
 $ BP2        : int  1295424 30792312 87801158 69546838 14373337 428662 13903204..
 $ TYPE       : int  3 1 3 1 1 3 1 1 3 1 ...
 $ geneID     : chr  "116983;118424;126789;1855;54587;54973;6339;80772;83756"..
 $ geneID_TYPE: chr  NA "146857;55106;91607" NA "57626" ...
 $ Symbol     : chr  "ACAP3;UBE2J2;PUSL1;DVL1;MXRA8;CPSF3L;SCNN1D;CPTP;TAS1R3"..
 $ Symbol_TYPE: chr  NA "SLFN13;SLFN12;SLFN11" NA "KLHL1" ...
 $ GsID       : chr  "BspanLA_lg2rpkm.MIN" "BspanLA_lg2rpkm.MIN" "BspanLA_"..
 $ GsName     : chr  "BspanLA_lg2rpkm.MIN" "BspanLA_lg2rpkm.MIN" "BspanLA_"..
```

Examining the data frame for a particular gene-set may reveal that its association due to certain genes may actually be better explained by *other* genes (those that have a clearer functional impact or that have previously been associated with the cases under consideration).

5.2.3 Visualizations

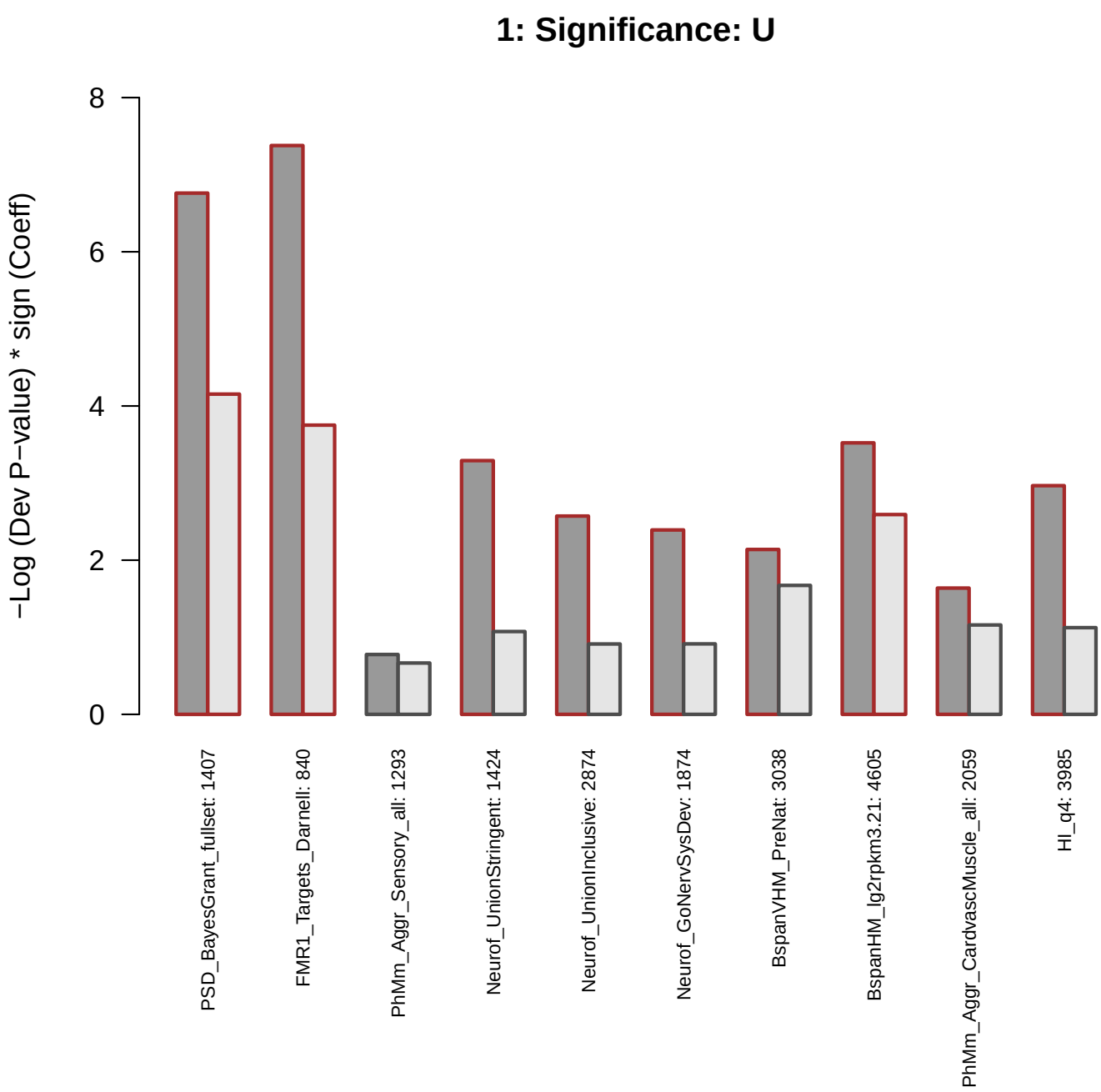
Once the `CnvGSAOutput` and `CnvGSAInput` objects are made, the user can run `f.makeViz` to create a collection of visualizations. An example is shown below:

```
> f.makeViz(cnvGSA.in,cnvGSA.out)
```

There are three different types of plots that will be made by this function. They show the significance, effect, and support of the gene-sets that are input by the user. Depending on which corrections the user chooses to include there can be up to 9 plots made. Only the support plots do not vary with the correction choice. The gene-sets that will be displayed on the plots should only include ones that are particularly interesting. Valid gene-set choices include all those that are in the gene-sets in the `$res.ls` data frame list elements. The user must input a comma separated file with no spaces containing all the gene-sets that they would like to look at. The file should resemble this:

```
PSD_BayesGrant_fullset,FMR1_Targets_Darnell,PhMm_Aggr_Sensory_all,
Neurof_UnionStringent,Neurof_UnionInclusive,Neurof_GoNervSysDev,BspanVHM_PreNat,
BspanHM_lg2rpkm3.21,PhMm_Aggr_CardvascMuscle_all,HI_q4
```

For the plots, anything with an FDR below the user specified cut off will have a brown border. This way the user can easily see how reliable the results are. An example of one of the plots with an FDR cut off of 0.1, the universe correction, and the gene-sets mentioned above is included below:


1: Significance: U
−Log (Dev P−value) * sign (Coeff)
0
2
4
6
8
PSD_BayesGrant_fullset: 1407
FMR1_Targets_Darnell: 840
PhMm_Aggr_Sensory_all: 1293
Neurof_UnionStringent: 1424
Neurof_UnionInclusive: 2874
Neurof_GoNervSysDev: 1874
BspanVHM_PreNat: 3038
BspanHM_lg2rpkm3.21: 4605
PhMm_Aggr_CardvascMuscle_all: 2059
HI_q4: 3985

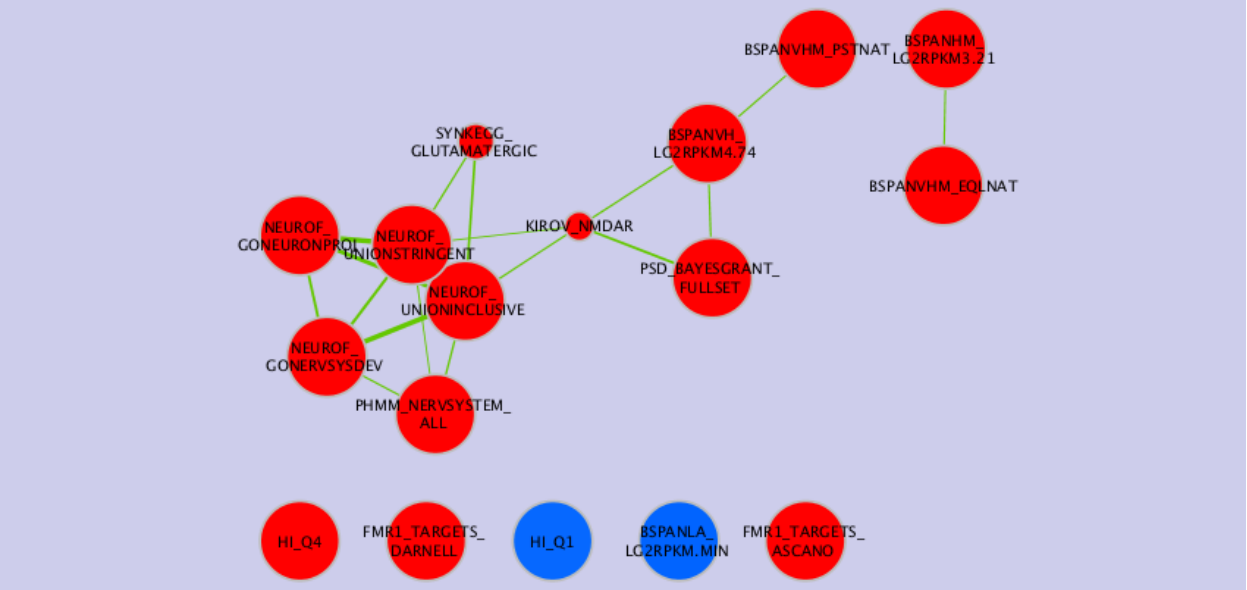
5.2.4 Enrichment Map pre-processing

To conduct further analysis of the results, this package has the function `f.enrFiles` that will do all the pre-processing for the files necessary to make an enrichment map. The enrichment map is created using Cytoscape and displays a network of gene-sets and how they overlap with one another. It will produce clusters that will convey the data in a meaningful way. The `$MinCaseCount`, `$MinControlCount`, and `$MinRatio` parameters in the configFile will influence the generation of the files produced by the `f.enrFiles` function. They provide cuttoffs that specify the minimum amount of case/control events each gene is allowed to have and the minimum case/control ratio each gene is allowed to have.

The `f.enrFiles` function will create two files (GMT and generic) needed to make the enrichment map. For more information on these types of files, see the **Enrichment Map User Guide**. Once the function is run, these two files will be output and will be available to use with the **Cytoscape** software. It is recommended to use the Jaccard+Overlap combined option when creating the map. There is a basic example below.

```
## To create the two files, simply run this function
> f.enrFiles(cnvGSA.in,cnvGSA.out)
```

The output will allow the user to use Cytoscape to create a network map that will look similar to this:



The map represents the gene-sets as nodes (circles) and links them together with edges (lines connecting circles). The closer the gene-sets are, the more similar the gene-sets are with one another. This creates clusters which make it easy to determine how the gene-sets compare to one another. The size of the node represents the size of the gene-set and the thickness of the edges is proportional to the overlap between the gene-sets. This overlap

refers to the number of genes two gene-sets share with one another. In a two-class experiment such as this one, red nodes display high enrichment in one class and blue nodes display high enrichment in the second class. For a more in depth look into how to interpret these Enrichment Maps, refer to the Enrichment Map paper [7].

If the user would like to use their own files to make the enrichment map they should follow the examples below. This can be found on the **Enrichment Map User Guide** as described above.

The GMT file should follow somehthing like this:

- Each row of the geneset file represents one gene-set and consists of:
    - gene-set name (–tab–) description (–tab–) a list of tab-delimited genes that are part of that gene-set.

The generic file should follow this format:

- The generic results file is a tab delimited file with enriched gene-sets and their corresponding p-values (and optionally, FDR corrections)
- The Generic Enrichment Results file needs:
    - gene-set ID (must match the gene-set ID in the GMT file),
    - gene-set name or description
    - p-value
    - FDR correction value
    - Phenotype: +1 or -1, to identify enrichment in up- and down-regulation, or, more in general, in either of the two phenotypes being compared in the two-class analysis (+1 maps to red and -1 maps to blue).

6 References

[1] Raychaudhuri S et al. Accurately assessing the risk of schizophrenia conferred by rare copy-number variation affecting genes with brain function. *PLoS Genet.* 2010 Sep 9; **6**(9): e1001097.

[2] Pinto, D et al. Functional impact of global rare copy number variation in autism spectrum disorders. *Nature.* 2010 Jul 15; **466**(7304): 368–72.

[3] Kirov G et al. De novo CNV analysis implicates specific abnormalities of postsynaptic signalling complexes in the pathogenesis of schizophrenia. *Mol Psychiatry.* 2012 Feb; **17**(2): 142–53.

[4] Pinto, D et al. Convergence of Genes and Cellular Pathways Dysregulated in Autism Spectrum Disorders. *Am J Hum Genet.* 2014 May 1; **94**(5): 677–694.

[5] Goeman JJ, Bhlmann P. Analyzing gene expression data in terms of gene sets: methodological issues. *Bioinformatics.* 2007 Apr 15; **23**(8): 980–7.

[6] Marshall C. R. et al. Structural Variation of Chromosomes in Autism Spectrum Disorder. *Am J Hum Genet.* 2008 Feb 8; **82**(2): 477–488.

[7] Merico D et al. Enrichment map: a network-based method for gene-set enrichment visualization and interpretation. *PLoS One.* 2010 Nov 15; **5**(11): e13984.