Combining SNP P-Values in Gene Sets: the cpvSNP Package

***Caitlin McHugh*[1,2*], *Jason Hackney*[1], *and Jessica L. Larson*[1]**

[1em] [1] Department of Bioinformatics and Computational Biology, Genentech, Inc.
[2] Department of Biostatistics, University of Washington
[*]mchughc (at) uw.edu

May 19, 2021

Contents

1 Introduction

Genome-wide association studies (GWAS) have lead to the discovery of many disease-associated single nucleotide polymorphisms (SNPs). Researchers are often interested in extending these studies to determine the genetic association of a given pathway (i.e., a gene set) with a certain phenotype. Gene set methods allow users to combine SNP-level association p-values across multiple biologically related genes.

The `cpvSNP` package provides code for two gene set analysis methods [1-2] and accurately corrects for the correlation structure among observed SNPs. Both of the implemented methods translate a set of gene ids to their corresponding SNPs, and combine the p-values for those SNPs. Calculated statistics, degrees of freedom, and corresponding p-values are stored for each gene set.

This vignette describes a typical analysis workflow and includes some details regarding the statistical theory behind `cpvSNP`. For more technical details, please see references [1] and [2].

2 Example workflow for cpvSNP

2.1 Preparing a dataset for analysis

For our example, we will use a set of simulated data, the `geneSetAnalysis` dataset from the `cpvSNP` package. We begin by loading relevant libraries, subsetting the data, and running `createArrayData` on this data set.

```
> library(cpvSNP)
> data(geneSetAnalysis)
> names(geneSetAnalysis)
```

```
[1] "arrayData" "geneSets"  "ldMat"     "indepSNPs"
```

The `geneSetAnalysis` list holds four elements, each of which we will need for this vignette. The first object, `arrayData`, is a `data.frame` containing the p-values, SNP ids, genomic position, and chromosome of all the probes in our hypothetical GWAS. Our first step is to use the `cpvSNP` function `createArrayData` to convert this `data.frame` to a `GRanges` object.

```
> arrayDataGR <- createArrayData(geneSetAnalysis[["arrayData"]],
+     positionName="Position")
> class(arrayDataGR)
```

```
[1] "GRanges"
attr(,"package")
```

```
[1] "GenomicRanges"
```

The `geneSetAnalysis` list also contains a `GeneSetCollection` with two sets of interest. We can find the Entrez ids by accessing the `geneIds` slot of the `GeneSetCollection`.

```
> geneSets <- geneSetAnalysis[["geneSets"]]
> geneSets
```

```
GeneSetCollection
  names: set1, set2 (2 total)
  unique identifiers: 100505495, 11128, ..., 80243 (250 total)
  types in collection:
    geneIdType: NullIdentifier (1 total)
    collectionType: NullCollection (1 total)
```

```
> length(geneSets)
```

```
[1] 2
```

```
> head(geneIds(geneSets[[1]]))
```

```
[1] "100505495" "11128"     "2857"      "2002"      "84466"     "100506696"
```

```
> details(geneSets[[1]])
```

```
setName: set1
geneIds: 100505495, 11128, ..., 6857 (total: 200)
geneIdType: Null
collectionType: Null
setIdentifier: rescomp216:19144:2014-08-28 13:23:17:1192854957
description: Randomly sampled gene set 1
organism:
pubMedIds:
urls:
contributor:
setVersion: 0.0.1
creationDate: Fri Aug  8 13:47:58 2014
```

```
> head(geneIds(geneSets[[2]]))
```

```
[1] "9447"   "6741"   "647979" "7846"   "55350"  "285987"
```

Our next data formatting step is to convert the ids in our `GeneSetCollection` from Entrez gene ids to their corresponding SNP ids. In this example, our SNP positions are coded in the hg19 genome build. Please be careful when converting gene ids to SNPs, as mappings change between genome build updates. The

geneToSNPList function requires gene transcripts stored as a GRanges object, along with the GRanges object specific to our study. For this example, we will use the gene transcripts stored in the database TxDb.Hsapiens.UCSC.hg19.knownGene.

```
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
> genesHg19 <- genes(txdb)
> snpsGSC <- geneToSNPList(geneSets, arrayDataGR, genesHg19)
> class(snpsGSC)
```

```
[1] "GeneSetCollection"
attr(,"package")
[1] "GSEABase"
```

Note that the geneToSNPList function has a quiet option defaulted to TRUE, which suppresses all warnings that may arise when finding overlaps between the genes in our collection and our study SNPs. The default is set to TRUE because there are often warnings that are usually not an issue. However, please be aware that valid warnings may also be suppressed if the quiet option is set to TRUE.

We now have the two input files required to run GLOSSI [1] and VEGAS [2]: a GRanges object for the SNPs in our GWAS, and a GeneSetCollection with SNP ids for each gene in each set.

```
> arrayDataGR
```

```
GRanges object with 1478 ranges and 6 metadata columns:
         seqnames    ranges strand |         P         SNP  Position chromosome
            <Rle> <IRanges>  <Rle> | <numeric> <character> <integer>   <factor>
     [1]     chr1  12686368      * |  0.438553  rs10779772  12686368       chr1
     [2]     chr1  12686476      * |  0.967386   rs3010868  12686476       chr1
     [3]     chr1  12687753      * |  0.803474   rs4568844  12687753       chr1
     [4]     chr1  12691826      * |  0.768926   rs3010872  12691826       chr1
     [5]     chr1  12692907      * |  0.602467   rs3000873  12692907       chr1
     ...      ...       ...    ... .       ...         ...       ...        ...
  [1474]     chr1 223543792      * |  0.599405   rs6681438 223543792       chr1
  [1475]     chr1 223544114      * |  0.211034  rs12024361 223544114       chr1
  [1476]     chr1 223544169      * |  0.846048  rs12042076 223544169       chr1
  [1477]     chr1 223544430      * |  0.299470   rs2036497 223544430       chr1
  [1478]     chr1 223551121      * |  0.145221    rs596166 223551121       chr1
             Start       End
         <numeric> <numeric>
     [1]  12686368  12686368
     [2]  12686476  12686476
     [3]  12687753  12687753
     [4]  12691826  12691826
```

```
      [5]  12692907  12692907
      ...       ...       ...
   [1474] 223543792 223543792
   [1475] 223544114 223544114
   [1476] 223544169 223544169
   [1477] 223544430 223544430
   [1478] 223551121 223551121
   -------
   seqinfo: 27 sequences from an unspecified genome; no seqlengths
```

```
> snpsGSC
```

```
GeneSetCollection
  names: set1, set2 (2 total)
  unique identifiers: rs3789052, rs3789051, ..., rs3766392 (1478 total)
  types in collection:
    geneIdType: AnnotationIdentifier (1 total)
    collectionType: NullCollection (1 total)
```

2.2 Running GLOSSI

An assumption of GLOSSI [1] is that our SNPs (and thus p-values) are independent. In order to run `glossi`, we must subset our `arrayDataGR` p-values to those from independent SNPs.

In the `geneSetAnalysis` list, we have included a vector of independent SNPs from our GWAS experiment. This list was created using a standard 'LD-pruning' method from the PLINK software [3].

```
> indep <- geneSetAnalysis[["indepSNPs"]]
> head(indep)
```

```
          V1
1  rs2649588
2  rs3107157
3  rs1456465
4  rs7528494
5 rs12046130
6 rs11590026
```

```
> dim(indep)
```

```
[1] 302   1
```

We now subset `arrayDataGR` to contain only independent SNPs, and create a new vector of p-values with names corresponding to these independent SNPs.

```
> pvals <- arrayDataGR$P[is.element(arrayDataGR$SNP, indep$V1)]
> names(pvals) <- arrayDataGR$SNP[is.element(arrayDataGR$SNP, indep$V1)]
> head(pvals)
```

```
rs2172285 rs2430130 rs1572750
0.7191158 0.3508501 0.8763177
```

We now have the proper input to call `glossi`. We can consider all gene sets in our `GeneSetCollection`, or call `glossi` on a just some of the sets. Accessor functions for the resulting `GLOSSIResultCollection` allow us to view the results.

```
> gRes <- glossi(pvals, snpsGSC)
> gRes
```

```
An object of class "GLOSSIResultCollection"
[[1]]
GLOSSI results for set1
p-value = 0.876
observed statistic = 0.132
degrees of freedom = 1

[[2]]
GLOSSI results for set2
p-value = 0.6
observed statistic = 1.38
degrees of freedom = 2
```

```
> gRes2 <- glossi(pvals, snpsGSC[[1]])
> gRes2
```

```
GLOSSI results for set1
p-value = 0.876
observed statistic = 0.132
degrees of freedom = 1
```

```
> pValue(gRes)
```

```
$set1
[1] 0.8763177

$set2
[1] 0.5997541
```

```
> degreesOfFreedom(gRes)
```

```
$set1
[1] 1
```

```
$set2
[1] 2
```

```
> statistic(gRes)
```

```
$set1
[1] 0.1320265

$set2
[1] 1.377129
```

Using the `ReportingTools` package, we can publish these results to a HTML page for exploration. We first adjust for multiple testing.

```
> pvals <- p.adjust( unlist( pValue(gRes) ), method= "BH" )
> library(ReportingTools)
> report <- HTMLReport (shortName = "cpvSNP_glossiResult",
+         title = "GLOSSI Results", reportDirectory = "./reports")
> publish(geneSets, report, annotation.db = "org.Hs.eg",
+         setStats = unlist(statistic (gRes)),
+         setPValues = pvals)
> finish(report)
```

2.3 Running VEGAS

Unlike GLOSSI, which requires SNPs and p-values to be independent, VEGAS [2] accounts for correlation among SNPs and corresponding p-values. We thus need a matrix of correlation values for each SNP in our GWAS. Most commonly, this correlation matrix holds linkage disequilibrium (LD) values. Many R packages and online tools exist to calculate an LD matrix for observed raw data.

Here, we briefly show how to calculate an LD matrix for chromosome 1 using the publicly available HapMap data, the R package `snpStats`, and the PLINK software package [3]. This requires downloading PLINK file formatted data, extracting the probes on chromosome 1, and then calculating LD among SNPs in the `snpsGSC` elements.

```
> download.file(
+     url="http://hapmap.ncbi.nlm.nih.gov/genotypes/hapmap3_r3/plink_format/hapmap3_r3_b
+     destfile="hapmap3_r3_b36_fwd.consensus.qc.poly.ped.gz")
> download.file(
+     url="http://hapmap.ncbi.nlm.nih.gov/genotypes/hapmap3_r3/plink_format/hapmap3_r3_b
+     destfile="hapmap3_r3_b36_fwd.consensus.qc.poly.map.gz")
> system("gunzip hapmap3_r3_b36_fwd.consensus.qc.poly.ped.gz")
```

```
> system("gunzip hapmap3_r3_b36_fwd.consensus.qc.poly.map.gz")
> system("plink --file hapmap3_r3_b36_fwd.consensus.qc.poly --make-bed --chr 1")
> genos <- read.plink(bed, bim, fam)
> genos$genotypes
> head(genos$map)
> x <- genos[,is.element(genos$map$snp.name,c(geneIds(snpsGSC[[2]])))]
> ldMat <- ld(x,y=x,stats="R.squared")
```

We have performed these steps already, and can simply use the LD matrix included in our `geneSetAnalysis` list, `ldMat` to call `vegas`. Note that the vegas method calculates simulated statistics (see Methods section below for more details).

```
> ldMat <- geneSetAnalysis[["ldMat"]]
> vRes <- vegas(snpsGSC[1], arrayDataGR, ldMat)
> vRes
> summary(unlist(simulatedStats(vRes)))
> pValue(vRes)
> degreesOfFreedom(vRes)
> statistic(vRes)
```

2.4 Visualizing Results

There are two plotting functions available in `cpvSNP` to visualize the results from the GLOSSI and VEGAS methods.

The `plotPvals` function plots the calculated p-values against the number of SNPs in each gene set, for each set in the original `GeneSetCollection`and `GLOSSIResultCollection`. In this vignette we have only analyzed two gene sets, so this plot is not very informative. The plot is included simply to demonstrate the plotting functions available in the `cpvSNP` package.

```
> plotPvals(gRes, main="GLOSSI P-values")
```

The `assocPvalBySetPlot` function plots the GWAS p-values for each SNP in the original association study, as well as those for SNPs in a particular gene set. This visualization enables an easy comparison of the p-values within a particular gene set to all p-values from our GWAS. Gene sets that are highly associated with the phenotype of interest will have a different distribution than all SNPs in our study.

```
> pvals <- arrayDataGR$P
> names(pvals) <- arrayDataGR$SNP
> assocPvalBySetPlot(pvals, snpsGSC[[2]])
```

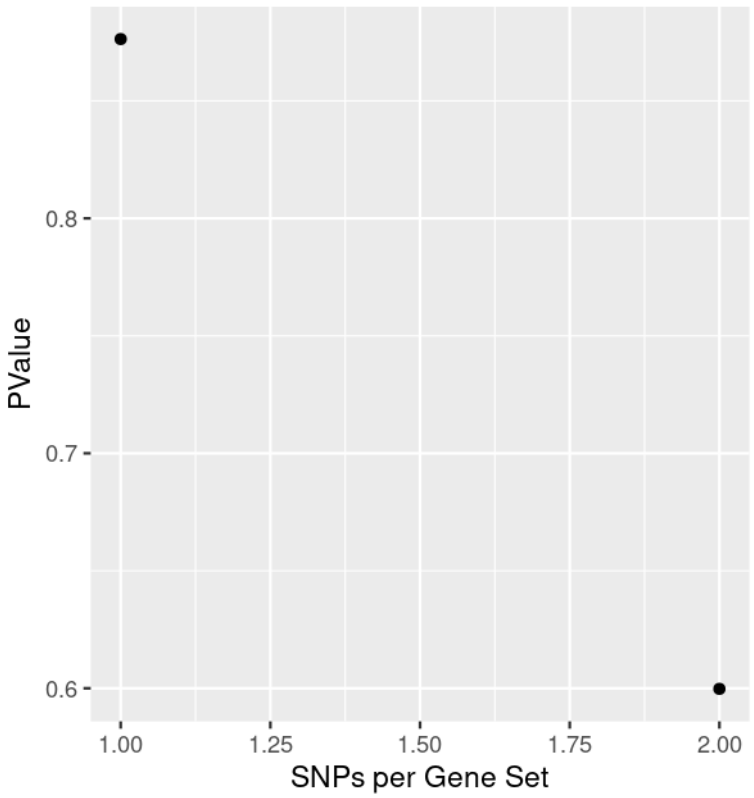

Figure 1: The number of SNPs per gene set versus the p-value, for the GLOSSI methods

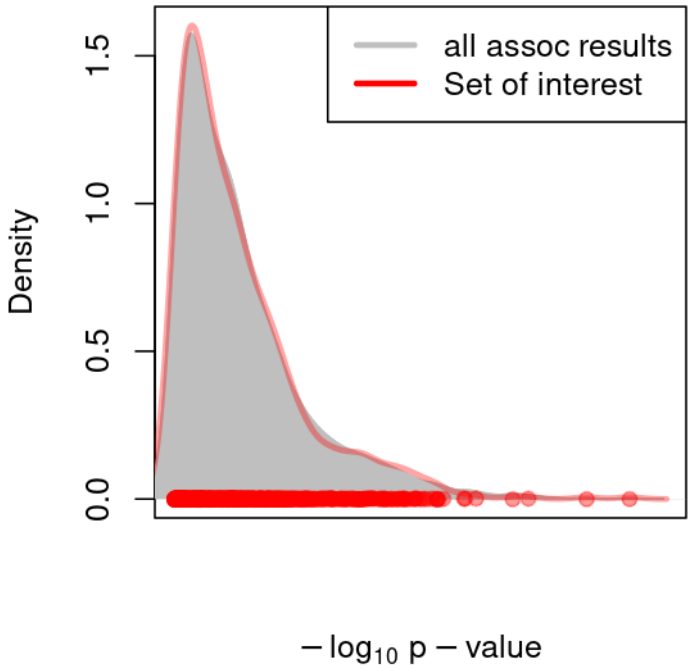

Figure 2: Density plots of all p-values, overlaid in red with p-values from the second gene set

3 Methods in brief

3.1 GLOSSI methods

The GLOSSI [1] method assumes that our p-values are independently distributed. Define J to be the total number of *independent* SNPs for which we have association p-values, such that each locus j has a corresponding p-value, p_j, $j \in \{1, \ldots, J\}$. For this vignette, $J = 302$. Let K be the total number of loci sets in which we are interested. For the example used in this vignette, $K = 2$.

We begin by defining an indicator variable g for each loci set k and for each locus j, such that

$$g_{jk} = \begin{cases} 1, & \text{if } j^{th} \text{ locus is in } k^{th} \text{ set} \\ 0, & \text{otherwise} \end{cases}$$

Note the sum of g_{jk} is the size of loci-set k

$$n_k = \sum_{j=1}^{J} g_{jk}$$

Our statistic for each loci-set k is defined as

$$S_{k_{obs}} = -2 \sum_{j=1}^{J} g_{jk} log(p_j)$$

We know from Fisher's transformation that if the $p_j \overset{iid}{\sim} Unif(0, 1)$ then $S_{k_{obs}} \sim \chi^2_{2n_k}$. Thus, to calculate the corresponding p-value for loci-set k, we simply use the corresponding χ^2 distribution for each set. Note the degrees of freedom in the null distribution takes into account the size of the loci-set, n_k.

3.2 VEGAS methods

The VEGAS [2] method does not require independent SNPs, but rather a matrix of correlation values among the SNPs being considered. These correlation values can be correlation coefficients, a composite LD measure, or similar. We denote the correlation matrix for a particular loci-set k as Σ_k, where each row and column corresponds to a SNP in k. This matrix must be square, symmetric, and have values of 1 on the diagonal.

To calculate a p-value for loci-set k that takes into account the correlation structure, we begin by simulating a vector $z \sim N(0,1)$ with length n_k. We take the Cholesky decomposition of Σ_k and multiply this by z to define a Multivariate Normal random variable $z' \sim MVN(0, \Sigma_k)$. To define a statistic from this null distribution that now has the same correlation structure as our observed data, we calculate

$$S_k = \sum_{i=1}^{n_k} [z_i chol(\Sigma_k)]^2$$

We simulate the vector z a total of n_{sims} times. We calculate the observed p-value as

$$\frac{\#(S_k > S_{k_{obs}}) + 1}{(n_{sims} + 1)}.$$

4 Session Info

- R version 4.1.0 (2021-05-18), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Running under: `Ubuntu 20.04.2 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.13-bioc/R/lib/libRblas.so`
- LAPACK: `/home/biocbuild/bbs-3.13-bioc/R/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.54.0, Biobase 2.52.0, BiocGenerics 0.38.0, GSEABase 1.54.0, GenomeInfoDb 1.28.0, GenomicFeatures 1.44.0, GenomicRanges 1.44.0, IRanges 2.26.0, S4Vectors 0.30.0, TxDb.Hsapiens.UCSC.hg19.knownGene 3.2.2, XML 3.99-0.6, annotate 1.70.0, cpvSNP 1.24.0, graph 1.70.0
- Loaded via a namespace (and not attached): BiocFileCache 2.0.0, BiocIO 1.2.0, BiocManager 1.30.15, BiocParallel 1.26.0, BiocStyle 2.20.0, Biostrings 2.60.0, DBI 1.1.1, DelayedArray 0.18.0, GenomeInfoDbData 1.2.6, GenomicAlignments 1.28.0, KEGGREST 1.32.0, Matrix 1.3-3, MatrixGenerics 1.4.0, R6 2.5.0,

RCurl 1.98-1.3, RSQLite 2.2.7, Rcpp 1.0.6, Rsamtools 2.8.0, SummarizedExperiment 1.22.0, XVector 0.32.0, assertthat 0.2.1, biomaRt 2.48.0, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.1, cachem 1.0.5, colorspace 2.0-1, compiler 4.1.0, corpcor 1.6.9, crayon 1.4.1, curl 4.3.1, dbplyr 2.1.1, digest 0.6.27, dplyr 1.0.6, ellipsis 0.3.2, evaluate 0.14, fansi 0.4.2, farver 2.1.0, fastmap 1.1.0, filelock 1.0.2, generics 0.1.0, ggplot2 3.3.3, glue 1.4.2, grid 4.1.0, gtable 0.3.0, hms 1.1.0, htmltools 0.5.1.1, httr 1.4.2, knitr 1.33, labeling 0.4.2, lattice 0.20-44, lifecycle 1.0.0, magrittr 2.0.1, matrixStats 0.58.0, memoise 2.0.0, munsell 0.5.0, pillar 1.6.1, pkgconfig 2.0.3, plyr 1.8.6, png 0.1-7, prettyunits 1.1.1, progress 1.2.2, purrr 0.3.4, rappdirs 0.3.3, restfulr 0.0.13, rjson 0.2.20, rlang 0.4.11, rmarkdown 2.8, rstudioapi 0.13, rtracklayer 1.52.0, scales 1.1.1, stringi 1.6.2, stringr 1.4.0, tibble 3.1.2, tidyselect 1.1.1, tools 4.1.0, utf8 1.2.1, vctrs 0.3.8, xfun 0.23, xtable 1.8-4, yaml 2.2.1, zlibbioc 1.38.0

5 References

1. Chai, High-Seng and Sicotte, Hughes et al. GLOSSI: a method to assess the association of genetic loci-sets with complex diseases. BMC Bioinformatics, 2009.

2. Liu, Jimmy Z. and Mcrae, Allan F. et al. A Versatile Gene-Based Test for Genome-Wide Association Studies. The American Journal of Human Genetics, 2010.

3. Purcell S., Neale B., and Sham P.C. et al. PLINK: a toolset for whole-genome association and population-based linkage analysis. American Journal of Human Genetics, 2007.