Gene Expressions Corrleation-R Package

Felipe Barraza, Jon Blichar, Emmanuel Edu

2021-04-09

Overview

This developing vignette is for the `geneExpCor` Rpackage which is capable of receiving tissue data and outputting correlation values corresponding to matching genes. Several functions are included that grant the user the capability to filter by variance percentage and by ligands. The 'geneExpCor' package will produce histograms and scatter plots corresponding to a biweight midcorrelation statistical analysis conducted on the inputted gene data sets. The built-in gene data sets, sourced from the GTEx consortium portal, are included to examine the capabilites of the functions within the package.

The objective of this software package is to identify putative endocrine interactions in a user-friendly way, and to compare organ communication discrepincies between males and females. The goal of this work is to build upon analyzing inter-tissue endocrine interactions utilizing bioinformatics methodologies.

Dependencies

- `devtools`: https://cran.r-project.org/web/packages/devtools/index.html
- `testthat`: https://testthat.r-lib.org/ Version 3.0 or above
- `WGCNA`: https://horvath.genetics.ucla.edu/html/CoexpressionNetwork/Rpackages/WGCNA/
- `iemisc`: https://cran.rstudio.com/web/packages/iemisc/index.html
- `dpylr`: https://www.tidyverse.org/
- `parallel`: https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf

Installation

1. Installing From Github

After dependencies have been installed and loaded, installing `geneExpCor` from github requires the following line:

```
devtools::install_github("Eziedu/Gene_Expression_Cors_JEF")
```

1.2 Addressing Installation Errors

If there is an error with loading `devtools`, it is most likely due to a version issue. The devtools package uses the `testthat`package, and in order for devtools to run appropriately the loaded `testhat` package needs to be of version 3.0 or higher.

If a simple update does not solve the problem, run the following line to update `testthat`:

```
install.packages("rlang", dependencies = c("Depends", "Imports", "LinkingTo", "Suggests"))
```

Follow the installation process as you see fit, afterwards the `testthat` package should update to version 3.0 or above.

2. Loading Necessary Data

To properly use the following functions, the inputted data needs to be in a specific format. To allow for easy testing, the package comes with two preinstalled tissue datasets: `subqAdipose_tissue_data` and `stomach_tissue_data`.

To properly access both datasets, we recommend attributing each set to a variable in the global enviornment. For example, run the following lines in the console:

```
library(geneExpCor)

tissue_1_data <- geneExpCor::subqAdipose_tissue_data
tissue_2_data <- geneExpCor::stomach_tissue_data
```

Addtionally a gene mapping dataset will need to be loaded to properly label gene names during analysis. This map is referred to as 'gene_map' and we recommend to load this data into the global enviornment.

```
gene_map <- geneExpCor::gencode_gene_map
```

The `gene_map` variable can now be used for susequent functions with the `ensembl_map` parameter.

Now that the gene map and both tissue sets have been loaded in the global environment, they can be used as arguments to fulfill the `tissue_matrix` parameters for all functions in the package.

Formatting Data

Filtering Data to Identify Matching Genes

To begin analysis the `gene_names` function will be used first. `gene_names()` takes in two tissue data sets and attempts to match the genes within both tissues and returns two matrices for each tissue with the matching genes. The `ensembl_map` parameter takes in a gene coding map to properly label the gene names in the dataset.

```
gene_names(tissue_1_matrix = tissue_1_data, tissue_2_matrix = tissue_2_data, ensembl_map = gene_map)
```

The outputted matrices are loaded in the global environment containing the matching donor and gene data present in both tissues. Therefore, both matricies have smaller dimensions than the original tissue data sets.

```
# Checking Dimensions
dim(final_tissue_1.matrix)
#> [1] 22699   244
```

The resulting matricies will have entries similar to the table below, where the rows represent the genes and columns are the samples.

```
final_tissue_1.matrix[1:3, 1:2]
#>               GTEX-111CU-1826-SM-5GZYN GTEX-111YS-2426-SM-5GZZQ
#> WASH7P                        2.029276                 2.037734
#> RP11-34P13.15                 2.652601                 2.000721
#> RP11-34P13.16                 3.655352                 3.003962
```

After the raw data files have been formatted to relabel gene names and filtered for matching genes amongst both tissue sets, the `final_tissue.matrix` matricies are in the proper form and can be used for subsequent functions within the package.

Conducting Gene Analysis

```
library(iemisc)
library(WGCNA)
#> Loading required package: dynamicTreeCut
#> Loading required package: fastcluster
#>
#> Attaching package: 'fastcluster'
#> The following object is masked from 'package:stats':
#>
#>     hclust
#>
#>
#> Attaching package: 'WGCNA'
#> The following object is masked from 'package:stats':
#>
#>     cor
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following object is masked from 'package:iemisc':
#>
#>     n
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
library(parallel)
```

Filtering for Most Variable Genes

To filter tissue matricies according to variance, the `var_filter` function should be utilized. This function takes two tissue matrices and a variance percentage and outputs matrices of the most variable genes for each tissue set. This function is primarily utilized with tissue sets containing matching genes and donors: `final_tissue_1.matrix`.

This function will return two lists of the most variable genes based on the `variance_percentage` parameter.

```
var_filter(final_tissue_1.matrix, final_tissue_2.matrix, 0.3)
```

In the example above, the function is sifting through both gene data sets and filtering for the top 30 percent of most variable genes.

The outputted matricies `variable.tissue_1.genes` and `variable.tissue_2.genes` contain the most variable genes amongst both tissue sets and will therefore have smaller dimensions than the inputted tissue sets.

```
#Check the Dimensions of the output matricies
dim(variable.tissue_1.genes)
#> [1] 6810  244
```

```
# View a section of the variance filtered matrix
variable.tissue_1.genes[3:5, 3:4]
#>                GTEX-1122O-2026-SM-9YFMG GTEX-117YX-2226-SM-5EGJJ
#> RP11-34P13.16                3.0511981                1.4651909
#> RP11-34P13.18                2.5883249                2.9747128
#> AP006222.2                   0.5162169                0.3422134
```

Conducting Biweight Midcorrelation Analysis

To conduct the biweight midcorrelation analysis on the tissue sets the `tissue_pair_gene_expression` function. This function takes in two tissue matrices and return two large matrices containing biweight midcorrelation coefficients and pvalues associated with both tissue sets.

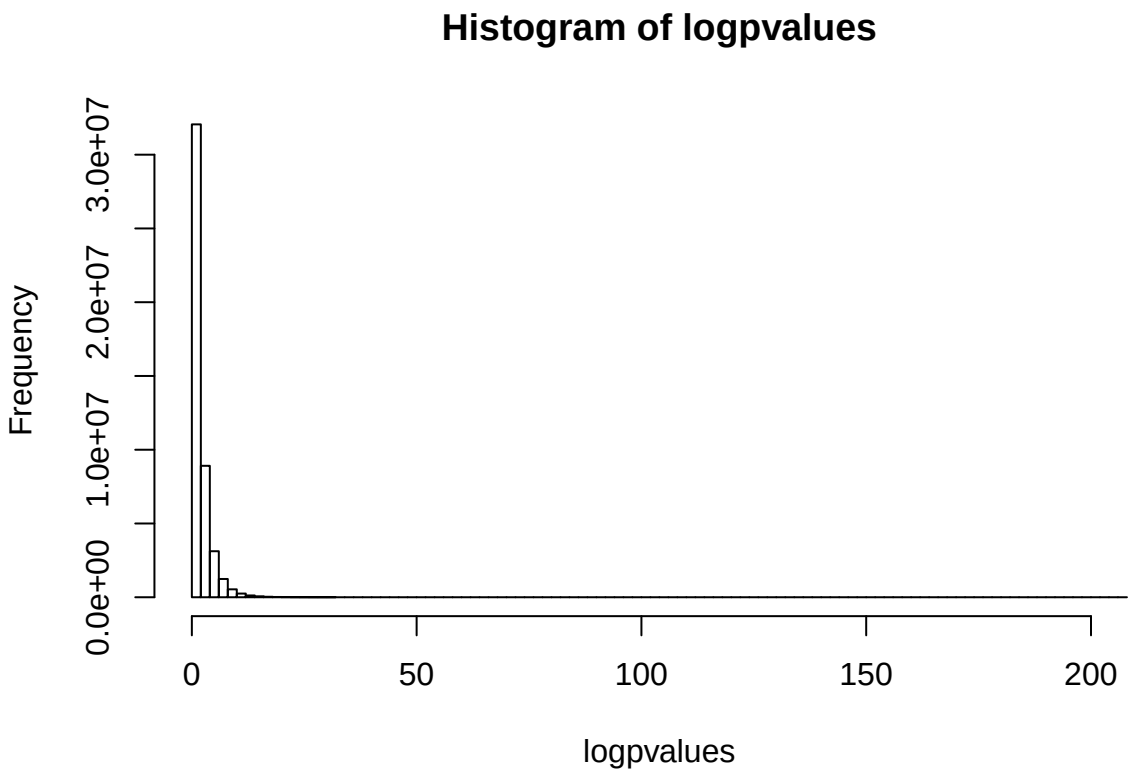
```
tissue_pair_gene_expression(variable.tissue_1.genes, variable.tissue_2.genes)
#> Warning in bicor(t(as.matrix(tissue_1_matrix)), t(as.matrix(tissue_2_matrix))):
#> bicor: zero MAD in variable 'x'. Pearson correlation was used for individual
#> columns with zero (or missing) MAD.
#> Warning in bicor(t(as.matrix(tissue_1_matrix)), t(as.matrix(tissue_2_matrix))):
#> bicor: zero MAD in variable 'y'. Pearson correlation was used for individual
#> columns with zero (or missing) MAD.
#> Warning in bicor(x, y, use = use, ...): bicor: zero MAD in variable 'x'. Pearson
#> correlation was used for individual columns with zero (or missing) MAD.
#> Warning in bicor(x, y, use = use, ...): bicor: zero MAD in variable 'y'. Pearson
#> correlation was used for individual columns with zero (or missing) MAD.
```

The outputted variables, `gene_biweight_midcorrelation` and `pvalues_biweight` are used to identify the corrlelations of the genes in both tissues matricies and are utilized to develop a series of plots to visualize the data.

Generating Plots

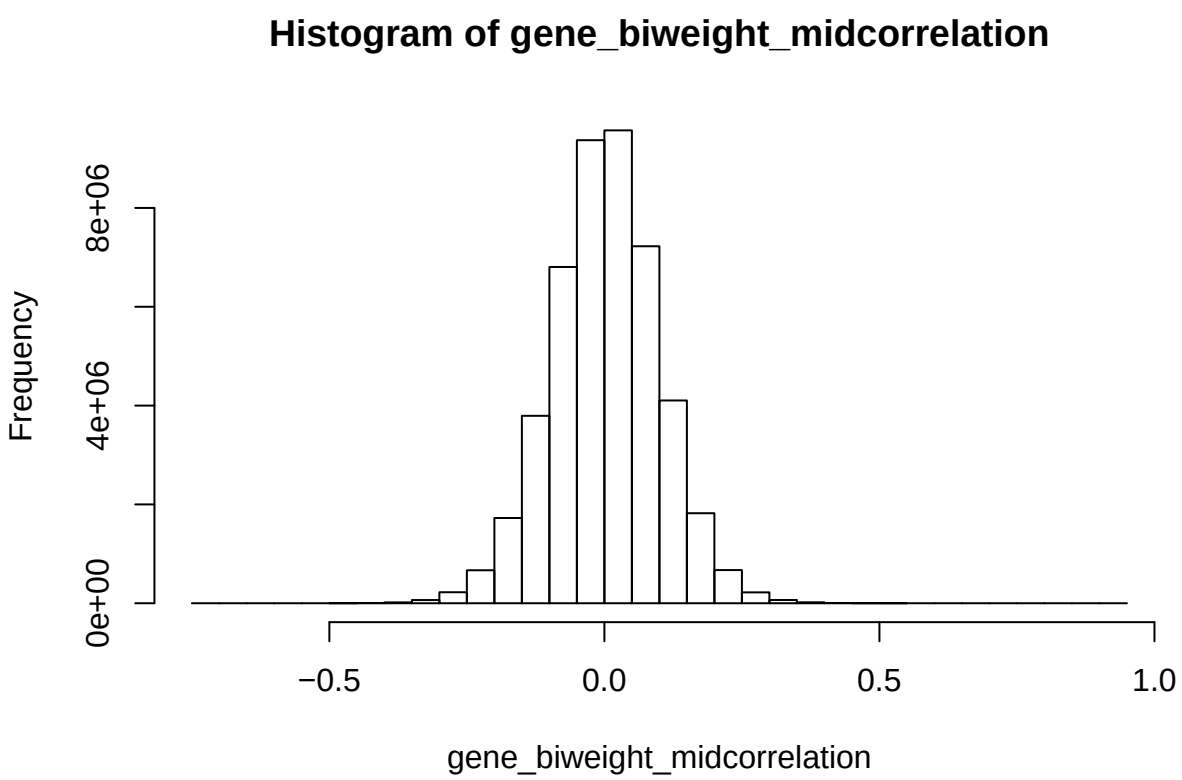
To properly create visuals for the analysis the matrix the pvalue matrix and biweight correlations from the `tissue_pair_gene_expression` function. The following code block produces a histogram using the inverse log pvalue data.

```
# Create a matrix of inverse log base 10 p-values
logpvalues <- -log(pvalues_biweight)

# Create a histogram of inverse log base 10 p-values
hist(logpvalues,100)
```



A histogram of the biweight correlations using the `gene_biweight_midcorrelation` matrix

```
# Create a histogram of correlation coefficients
hist(gene_biweight_midcorrelation)
```



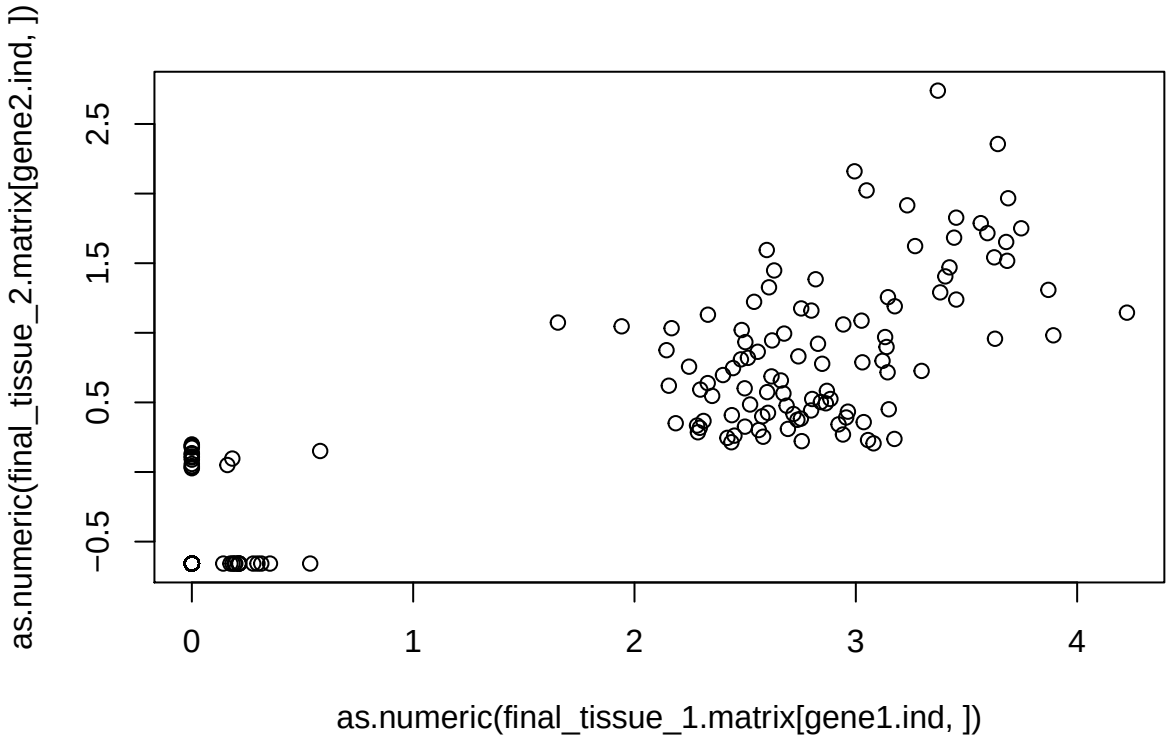
More customized plots can be developed by identifying certain pvalues that lie above a particular threshold. In this case, the cutoff considers log pvalues above 150. This value is arbitrary and can be altered to find a specific range of pvalues.

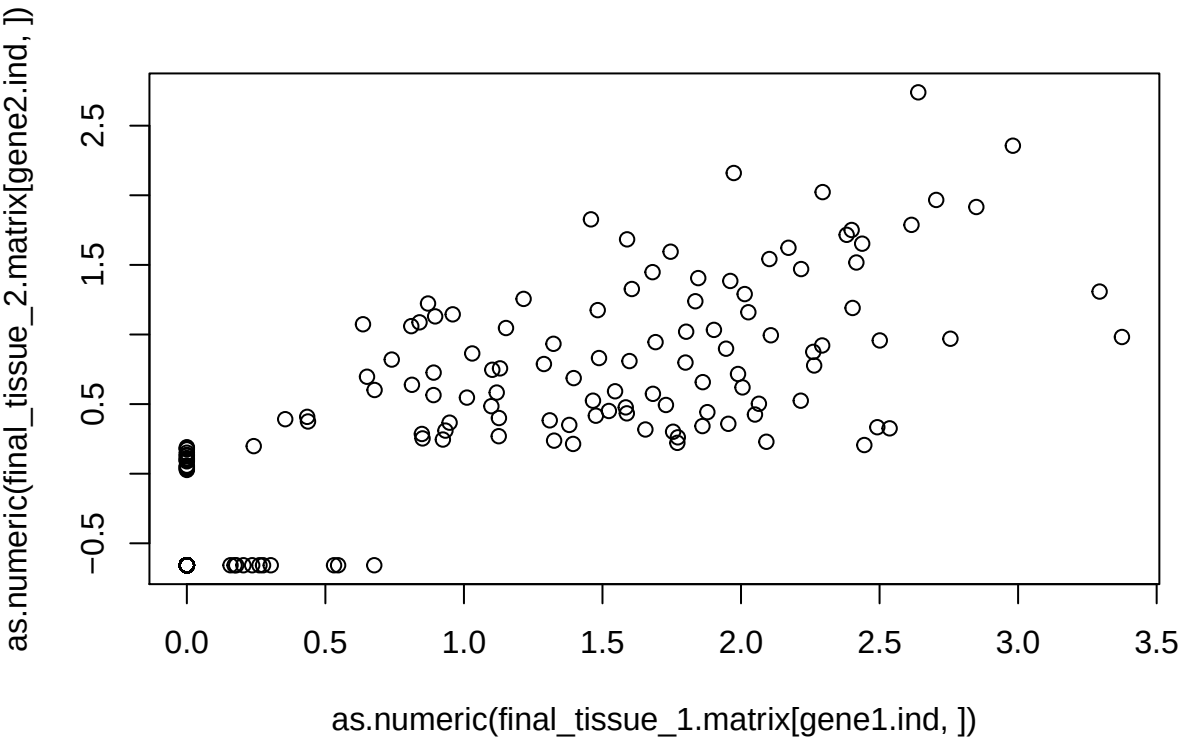
```
# Find the indices for the p-values greater than some cutoff value
cutoff.ind <- which(logpvalues >= 150, arr.ind = T)
```

The following code block generates a series of scatter plots according the size and values of the pvalue cutoff variable `cutoff.ind`. These scatter plots depict the pvalues between multiple gene pairs within the `final_tissue` matricies produced from the `gene_names` function.

```
for (i in 1:(size(cutoff.ind,1) - 2)){
  # Find the indices for the individual gene pairs for each run through the for loop
  ind.variance <- which(pvalues_biweight == pvalues_biweight[cutoff.ind[i,1],cutoff.ind[i,2]], arr.ind =

  # Create variables containing the rownames and columnames to match gene names
  rows <- rownames(pvalues_biweight)
  columns <- colnames(pvalues_biweight)

  # Get the gene names for the highly correlated gene pair of interest
  gene1 = rows[ind.variance[1]]
  gene2 = columns[ind.variance[2]]

  # Find the indices of the highly correlated genes in our matrix of gene expressions
  gene1.ind = which(rownames(final_tissue_1.matrix) == gene1)
  gene2.ind = which(rownames(final_tissue_2.matrix) == gene2)
```

```
  # Plot the expressions of the two genes with high degree of correlation
  plot(as.numeric(final_tissue_1.matrix[gene1.ind,]),as.numeric(final_tissue_2.matrix[gene2.ind,]))
}
```




as.numeric(final_tissue_2.matrix[gene2.ind,])
−0.5
0.5
1.5
2.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
as.numeric(final_tissue_1.matrix[gene1.ind,])