GeneExpCor_Vignette

Felipe Barraza, Jon Blichar, Emmanuel Edu

2021-03-28

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 including that grant the user the capability to filter by variance percentage and by ligands. Main Functions:

- tissue_pair_expression()
- gene_names()
- var_filter()

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/

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:

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

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

Functions

tissue_pair_gene_expression() Version 1

This function takes in two tissue matrices. The current version of this function only accepts tissue matrices(`tissue_1_matrix` and `tissue_2_matrix`) as tables that contain the gene identifiers as rows and the donor identifiers as columns. The current version of this function is only able to produce viable outputs for biweight midcorrelations.

```
tissue_pair_gene_expression <- function(tissue_1_matrix,tissue_2_matrix) {}
```

The following lines of code seperate the columns names from the inputted tissue data matrices and separates them to match the donor identifiers. This is stored into two separate variables which will be used to identify the matching genes between both tissues.

```
tissue_pair_gene_expression <- function(tissue_1_matrix,tissue_2_matrix) {
## Create a list of column names for downstream matching of donors
  tissue_1.samples = colnames(tissue_1_matrix)
  tissue_2.samples = colnames(tissue_2_matrix)

  ## Separate strings of column names to match donor tag identifiers
  tissue_1.subjects = sapply(tissue_1.samples,function(x){
    subj = strsplit(x,'[-]')[[1]]
    out = paste0(subj[1],'_',subj[2])
    return(out)
  })
  tissue_2.subjects = sapply(tissue_2.samples,function(x){
    subj = strsplit(x,'[-]')[[1]]
    out = paste0(subj[1],'_',subj[2])
    return(out)
  })
#combined_tissue.subjects is a list of the matching genes from both tissues
```

```
and thus should be a smaller size than both tissue_subjects variables.
  combined_tissue.subjects = intersect(tissue_1.subjects,tissue_2.subjects)
...}
```

After the matching donors are placed into a list, two `sapply` functions are used to sift through the combined list to identify the donor data that belongs to their respective tissues. The matched donors of each tissue are placed into two separate lists under `donor_matched_tissue_1.matrix` and `donor_matched_tissue_2.matrix`.

```
tissue_pair_gene_expression <- function(tissue_1_matrix,tissue_2_matrix) {
  ...
 ## Index the matching donors so as to remove any excess information
  ind.tissue_1 = sapply(combined_tissue.subjects,function(x){
    out = which(tissue_1.subjects == x)
    return(out)
  })
  ind.tissue_2 = sapply(combined_tissue.subjects,function(x){
    out = which(tissue_2.subjects == x)
    return(out)
  })
  donor_matched_tissue_1.matrix = tissue_1_matrix[,ind.tissue_1]
  donor_matched_tissue_2.matrix = tissue_2_matrix[,ind.tissue_2]
  ...}
```

Next, the genes from each tissue are selected and compared to store matching genes in a list. The list of matching genes are attributed to the matching donors in the variables `final_tissue_1.matrix` and `final_tissue_2.matrix`.

```
tissue_pair_gene_expression <- function(tissue_1_matrix,tissue_2_matrix) {
  ...
 ## Perform similar data truncation with the gene information
  tissue_1.genes <- rownames(tissue_1_matrix)
  tissue_2.genes <- rownames(tissue_2_matrix)
  combined_tissue.genes <- intersect(tissue_1.genes,tissue_2.genes)
  ind.gene_tissue_1 = sapply(combined_tissue.genes,function(x){
    out = which(tissue_1.genes == x)
    return(out)
  })
  ind.gene_tissue_2 = sapply(combined_tissue.genes,function(x){
    out = which(tissue_2.genes == x)
    return(out)
  })
  final_tissue_1.matrix = donor_matched_tissue_1.matrix[ind.gene_tissue_1,]
  final_tissue_2.matrix = donor_matched_tissue_2.matrix[ind.gene_tissue_2,]
...}
```

Now that the matching donors and genes are known comparisons can be made for analysis. Each gene pair between both tissues are examined for their variance and the values are stored as a list in `gene_expression_variance`

```
 ## Solve for the variance within genes across donors
tissue_pair_gene_expression <- function(tissue_1_matrix,tissue_2_matrix) {
  ...
  library(iemisc)
  gene_expression_variance <- c()
  for (i in 1:size(as.matrix(final_tissue_1.matrix),1)) {
    gene_expression_variance[i] <-
var(as.numeric(final_tissue_1.matrix[i,]),as.numeric(final_tissue_2.matrix[i,
]))
  }

...}
```

The variances are then filtered according to variance percentage which is based on user input.

```
tissue_pair_gene_expression <- function(tissue_1_matrix,tissue_2_matrix) {
  ...
## Filter for the most variable genes (based on input variance_percentage)
  ind.variance <- which(gene_expression_variance >=
quantile(gene_expression_variance, probs = 1 - .2)) #variance_percentage =
0.2

  variable.tissue_1.genes <- final_tissue_1.matrix[ind.variance,]
  variable.tissue_2.genes <- final_tissue_2.matrix[ind.variance,]
...}
```

Lastly, the biweight midcorrelation analysis is run on the filtered genes producing a matrix of correlation values and a second list of pvalues.

```
tissue_pair_gene_expression <- function(tissue_1_matrix,tissue_2_matrix) {
  ...
  library(WGCNA)
  gene_biweight_midcorrelation <-
bicor(t(as.matrix(variable.tissue_1.genes)),t(as.matrix(variable.tissue_2.gen
es)))
  pvalues_biweight <-
bicorAndPvalue(t(as.matrix(variable.tissue_1.genes)),t(as.matrix(variable.tis
sue_2.genes)))$p
  assign("gene_biweight_midcorrelation",gene_biweight_midcorrelation, envir =
globalenv())
  assign("pvalues_biweight",pvalues_biweight, envir = globalenv())
...}
```

gene_names()

Takes in two tissue data sets and attempts to match the genes within both tissues. Returns two matrices for each tissue with the matching genes. This function is similar to the previous, `tissue_pair_expressions` but lacks the correlation and pvalue analysis. This function is strictly for acquiring two matricies of the matche genes across both tissues.

```r
gene_names <- function(tissue_matrix_1,tissue_matrix_2){
  ## Create a list of column names for downstream matching of donors
  tissue_1.samples = colnames(tissue_1_matrix)
  tissue_2.samples = colnames(tissue_2_matrix)

  ## Separate strings of column names to match donor tag identifiers
  tissue_1.subjects = sapply(tissue_1.samples,function(x){
    subj = strsplit(x,'[-]')[[1]]
    out = paste0(subj[1],'_',subj[2])
    return(out)
  })
  tissue_2.subjects = sapply(tissue_2.samples,function(x){
    subj = strsplit(x,'[-]')[[1]]
    out = paste0(subj[1],'_',subj[2])
    return(out)
  })
  combined_tissue.subjects = intersect(tissue_1.subjects,tissue_2.subjects)

  ## Index the matching donors so as to remove non-matching information
  ind.tissue_1 = sapply(combined_tissue.subjects,function(x){
    out = which(tissue_1.subjects == x)
    return(out)
  })
  ind.tissue_2 = sapply(combined_tissue.subjects,function(x){
    out = which(tissue_2.subjects == x)
    return(out)
  })
  donor_matched_tissue_1.matrix = tissue_1_matrix[,ind.tissue_1]
  donor_matched_tissue_2.matrix = tissue_2_matrix[,ind.tissue_2]

  ## Perform similar data truncation with the gene information
  tissue_1.genes <- rownames(tissue_1_matrix)
  tissue_2.genes <- rownames(tissue_2_matrix)
  combined_tissue.genes <- intersect(tissue_1.genes,tissue_2.genes)
  ind.gene_tissue_1 = sapply(combined_tissue.genes,function(x){
    out = which(tissue_1.genes == x)
    return(out)
  })
  ind.gene_tissue_2 = sapply(combined_tissue.genes,function(x){
    out = which(tissue_2.genes == x)
    return(out)
  })
  final_tissue_1.matrix = donor_matched_tissue_1.matrix[ind.gene_tissue_1,]
  final_tissue_2.matrix = donor_matched_tissue_2.matrix[ind.gene_tissue_2,]

  assign("final_tissue_1.matrix",final_tissue_1.matrix, envir = globalenv())
  assign("final_tissue_2.matrix",final_tissue_2.matrix, envir = globalenv())

}
```

The outputted matrices can be used for the `var_filter()` which returns matrices of the most variable genes based on user input. It is recommended that `gene_names()` be used prior to `var_filter()` so the necessary tissue matrices(`final_tissue_1.matrix` and `final_tissue_2.matrix`) are ready for input into the `var_filter` function.

var_filter()

Functions that takes tissue matrices and a given variance percentage and outputs matrices of the most variable genes for each tissue set. This function is primarily utilized with inputted tissue sets with matching donors and genes. Therefore the `var_filter` function serves as the supplement after the use of the function `gene_names` which outputs matching gene tissue matrices. This function will return two list of the most variable genes based on the `variance_percentage` parameter.

```
var_filter <- function(tissue_matrix_1,tissue_matrix_2,percentage_variance){

  ## Solve for the variance within genes across donors
  library(iemisc)
  gene_expression_variance <- c()
  for (i in 1:size(as.matrix(tissue_matrix_1),1)) {
    gene_expression_variance[i] <-
var(as.numeric(tissue_matrix_1[i,]),as.numeric(tissue_matrix_2[i,]))
  }

  gene_variance_test <- mcmapply(function(x)
var(as.numeric(tissue_matrix_1[x,]),as.numeric(tissue_matrix_2[x,])),1:nrow(t
issue_matrix_1))

  ## Filter for the most variable genes (based on input variance_percentage)
  ind.variance <- which(gene_expression_variance >=
quantile(gene_expression_variance, probs = 1 - variance_percentage))

  variable.tissue_1.genes <- tissue_matrix_1[ind.variance,]
  variable.tissue_2.genes <- tissue_matrix_2[ind.variance,]

  assign("variable.tissue_1.genes",variable.tissue_1.genes, envir =
globalenv())
  assign("variable.tissue_2.genes",variable.tissue_2.genes, envir =
globalenv())
}
```

Future Function updates

-Plan on improving `tissue_pair_expression()` to allow for 'pearson' and 'spearman' correlations.

-Adding more functions that increase flexibility of analysis: ligand filtration