INDEED R package for cancer biomarker discovery

Yiming Zuo and Kian Ghaffari

2019-09-30

Introduction

Differential expression (DE) analysis is commonly used to identify biomarker candidates that have significant changes in their expression levels between distinct biological groups. One drawback of DE analysis is that it only considers the changes on single biomolecular level. In differential network (DN) analysis, network is typically built based on the correlation and biomarker candidates are selected by investigating the network topology. However, correlation tends to generate over-complicated networks and the selection of biomarker candidates purely based on network topology ignores the changes on single biomolecule level. Thus, we have proposed a novel method INDEED, which considers both the changes on single biomolecular and network levels by integrating DE and DN analysis. INDEED has been published in Methods journal (PMID: 27592383). This is the R package that implements the algorithm.

This R package will generate a list of dataframes containing information such as p-values, node degree and activity score for each biomolecule. A higher activity score indicates that the corresponding biomolecule has more neighbors connceted in the differential network and their p-values are more statistically significant. It will also generate a network display to aid users' biomarker selection.

Installation

You can install INDEED from github with:

```r
# install.packages("devtools")
devtools::install_github("ressomlab/INDEED")
```

Load package

Load the package.

```r
# load INDEED
library(INDEED)
# Loading required package: glasso
```

Testing dataset

A testing dataset has been provided to the users to get familiar with INDEED R package. It contains the expression levels of 39 metabolites from 120 subjects (CIRR: 60; HCC: 60) with CIRR group named as group 0 and HCC group named as group 1.

```r
# Data matrix contains the expression levels of 39 metabolites from 120 subjects
# (6 metabolites and 10 subjects are shown)
head(Met_GU[, 1:10])
# Group label for each subject (40 subjects are shown)
Met_Group_GU[1:40]
# Metabolite KEGG IDs (10 metabolites are shown)
Met_name_GU[1:10]
```

non-partial correlation data analysis function `non_partial_cor()`

- (**data**) A dataframe that contains the expression level of individual biomolecule from two biologically disparate groups (p * n)
- (**class_label**) A binary array with group 1 labeled as 0 and group 2 as 1
- (**id**) An array that includes the corresponding ID for each biomolecule
- (**method**) Type of correlation, user can choose between pearson and spearman correaltion, default pearson
- (**permutation**) A positive integer number of permutation, default 1000
- (**p_val**) An optional dataframe containing p-values obtained from DE analysis

In non partical correlation method, user only need to run `non_partial_cor()` function. No rho number will be needed. In this method, user input dataframe with expression level, class label, id, method, number of permutation and p-value similar to partical correaltion but in one step fashion. Result will be saved in a list of two dataframes: activity_score and diff_network. Activity_score contains a dataframe with biomolecules ranked by activity score calculated from p-value and node degree. Diff_network contains a dataframe of connection weight between two nodes.

The following example demonstrates how to use `non_partial_cor()` function:

```
non_partial_cor(data = Met_GU, class_label = Met_Group_GU, id = Met_name_GU, method = "spearman")
```

partial correlation data preprocessing function `select_rho_partial()`

To use the function `select_rho_partial()`, users will need to have these data sets in hand:

- (**data**) A dataframe that contains the expression level of individual biomolecule from two biologically disparate groups (p * n)
- (**class_label**) A binary array with group 1 labeled as 0 and group 2 as 1
- (**id**) An array that includes the corresponding ID for each biomolecule
- (**error_curve**) This is an option on whether a error curve plot will be provided to the user, user can choose "YES" or "NO". The default is YES

partial correlation data analysis function `partial_cor()`

- (**data_list**) A list of dataframes output from select_rho_partial step
- (**rho_group1**) Rule to choose rho for first group of data, user can input "min" for minimum rule, "ste" as one standard error rule, or user can input a number of choice, the default is min
- (**rho_group2**) Rule to choose rho for second group of data, user can input "min" for minimum rule, "ste" as one standard error rule, or user can input a number of choice, the default is min
- (**permutation**) A positive integer number of permutation, default 1000
- (**p_val**) A dataframe containing p-values obtained from DE analysis (optional)
- (**permutation_thres**) This is the threshold for permutation. The defalut is 0.05 to make 95 percent confidenc

In partical correlation method, user will need to preprocess the data using `select_rho_partial()` function, followed by using `partial_cor()` and choosing desired rho number and complete the analysis. User can also choose to provide p-value table and number of permutations in `partial_cor()` function. Result will be saved in a list of two dataframes: activity_score and diff_network. Activity_score contains a dataframe with biomolecules ranked by activity score calculated from p-value and node degree. Diff_network contains a dataframe of connection weight between two nodes.

The following example demonstrates how to use `select_rho_partial()` and `partial_cor()`function:

```
pre_data <- select_rho_partial(data = Met_GU, class_label = Met_Group_GU, id = Met_name_GU,
                               error_curve = "YES")
result <- partial_cor(data_list = pre_data, rho_group1 = 'min', rho_group2 = "min",
                      permutation = 1000, p_val = pvalue_M_GU, permutation_thres = 0.05)
```

In this case, the sparse differential network is based on partial correlation and p-value for each biomolecule is calculated for users. Also, **rho** is picked based on minimum rule and number of permutations is set to 1000.

Interactive Network Visualization function `network_display()`

- (**results**) This is the result from the calling either non_partial_cor() or partial_cor().
- (**nodesize**) This parameter determines what the size of each node will represent. The options are 'Node_Degree', 'Activity_Score', 'Z_Score', and 'P_Value'. The title of the resulting network will identify which parameter was selected to represent the node size. The default is Node_Degree.
- (**nodecolor**) This parameter determines what color each node will be, based on a yellow or red color gradient. The options are 'Node_Degree', 'Activity_Score', 'Z_Score', and 'P_Value'. A color bar will be created based on which parameter is chosen. The default is Activity_Score.
- (**edgewidth**) This is a 'YES' or 'NO' option as to if the edgewidth should be representative of the weight value corresponding to the correlation between two nodes. The default is NO.
- (**layout**) User can choose from a a handful of network visualization templates including:'nice', 'sphere', 'grid', 'star', and 'circle'. The default is nice.

An interactive tool to assist in the visualization of the results from INDEED functions non_partial_corr() or patial_corr(). The size and the color of each node can be adjusted by users to represent either the Node_Degree, Activity_Score, Z_Score, or P_Value. The color of the edge is based on the binary value of either 1 corresonding to a positive correlation change dipicted as green or a negative correlation change of -1 dipicted as red. The user also has the option of having the width of each edge be proportional to its weight value. The layout of the network can also be customized by choosing from the options: 'nice', 'sphere', 'grid', 'star', and 'circle'. Nodes can be moved and zoomed in on. Each node and edge will display extra information when clicked on. Secondary interactions will be highlighted as well when a node is clicked on.

The following example demonstrates how to use the `network_display()` function:

```
network_display(results = result, nodesize= 'Node_Degree', nodecolor= 'Activity_Score',
                edgewidth= 'NO', layout= 'nice')
```