MarkRank Tutorial

Duanchen Sun and Ling-Yun Wu

2018-12-18

Contents

1 Introduction

MarkRank is a network-based gene ranking method for identifying the cooperative biomarkers for heterogenous diseases. MarkRank uses the gene cooperation network to explicitly model the gene cooperative effects. MarkRank suggests that explicit modeling of gene cooperative effects can greatly improve the performance of biomarker identification for complex diseases, especially for diseases with high heterogeneity. This tutorial could help the user to execute the markrank function compiled in the Corbi package.

We first import Corbi and other required packages:

```
rm(list=ls(all=TRUE))
library(Corbi)
library(Matrix)
```

```
##
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:Corbi':
##
##     nnzero
```

```
options(scipen=0)
```

2 MarkRank example

The inputs of markrank function include an expression dataset with labelled (e.g. disease/normal) samples and an adjacent matrix of biological network (e.g. PPI network). Here we use simulated dataset to illustrate the usage of markrank function.

2.1 Simulate dataset

First, we load in a small network using another function read_net compiled in Corbi.

```
net <- read_net("network.txt")
```

This network contains 100 genes. Then we set the number of preset differential expression genes.

```
size <- 10
```

We randomly extract a connected subnetwork with the preset size from the loaded network. Here we use the function search_net to implement.

```
source("search_net.R")
subnet  <- search_net(net, node_size = size, ori_name = TRUE)
deg_list <- as.character(unique(as.vector(subnet)))
```

The preset differentially expression genes are:

```
deg_list
```

```
##  [1] "5"  "46" "7"  "12" "11" "4"  "13" "99" "3"  "74"
```

Now we simulate the expression matrix. The sample number is set as

```
sample_num <- 50
```

The number of disease samples and normal samples are equal.

```
disease_num <- 25
```

The code of simulating the expression dataset is as follows. We up-regulated the expression values of preset differentially expression gene set. The detailed description of this process can be found in the Supplementary Materials in our manuscript.

```
library(matrixcalc)
library(MASS)
l <- net$size
p <- length(deg_list)
exp_dataset <- matrix(0, sample_num, l, dimnames = list(paste("sample", 1:sample_num, sep=""), net$node
vars  <- 1
sigma <- matrix(0)
while(!is.positive.definite(sigma)){
  vars  <- vars + 1
  sigma <- as.matrix(as(net$matrix,'dgCMatrix'))
  sigma[which(sigma == 1)] <- rnorm(length(which(sigma == 1)), 4, 1)
  sigma[which(sigma == 0)] <- rnorm(length(which(sigma == 0)), 2, 1)
  diag(sigma) <- rnorm(l, vars, 1)
  sigma <- (sigma + t(sigma))/2
}
sample_mean <- rnorm(l, 5, 1)
exp_dataset <- mvrnorm(sample_num, sample_mean, sigma)
exp_dataset[1:disease_num, deg_list] <- exp_dataset[1:disease_num, deg_list] * rnorm(disease_num*p, 2, 
```

The final simulated gene expression dataset contains 50 samples and 100 genes. The number of preset marker genes is 10.

```
dim(exp_dataset)
```

```
## [1]  50 100
```

The sample label is

```
label <- c(rep(0, disease_num), rep(1, sample_num-disease_num))
```

The adjacent matrix of the network is

```
adj_matrix <- as.matrix(net$matrix)
adj_matrix <- adj_matrix[colnames(exp_dataset), colnames(exp_dataset)]
adj_matrix <- adj_matrix + t(adj_matrix)
```

2.2 Run markrank

With the above simulated datasets as inputs, we now execute the markrank function to test whether MarkRank could prioritize the preset genes. We use the default parameter combination as alpha=0.8 and lambda=0.2 to run the markrank.

```
time1 <- system.time(
  result1 <- markrank(exp_dataset, label, adj_matrix, alpha=0.8, lambda=0.2, trace=TRUE)
  )
```

```
## [1] "Computing discriminative potential network ..."
## [1] 10
## [1] 20
## [1] 30
## [1] 40
## [1] 50
## [1] 60
## [1] 70
## [1] 80
## [1] 90
```

The output result of markrank contains the following variables:

```
names(result1)
```

```
## [1] "score"        "steps"        "NET2"         "initial_pars"
## [5] "dis"
```

The scores of top 10 markrank genes are:

```
s1 <- sort(result1$score, decreasing=TRUE)
s1[1:10]
```

```
##         46          3         12          4          5         99
## 0.11432243 0.09590014 0.09561663 0.05411746 0.04243733 0.03815478
##         13         74         15         19
## 0.02716366 0.02318233 0.01932772 0.01904058
```

The scores of pre-set differential expression genes are:

```
result1$score[deg_list]
```

```
##          5         46          7         12         11          4
## 0.04243733 0.11432243 0.01665109 0.09561663 0.01697651 0.05411746
##         13         99          3         74
## 0.02716366 0.03815478 0.09590014 0.02318233
```

The false discovery genes are:

```
setdiff(names(s1[1:10]), deg_list)
```

```
## [1] "15" "19"
```

The iteration steps in the random walk iteration is

```
result1$steps
```

```
## [1] 115
```

The user could find the input parameters by using the following code:

```
result1$initial_pars
```

```
## $alpha
## [1] 0.8
##
## $lambda
## [1] 0.2
##
## $eps
## [1] 1e-10
```

3 Reuse gene cooperation network

The computation of gene cooperation network is time-consuming. To reduce the redundant computation, we can reuse the gene cooperation network computed in previous step. The computed gene cooperation network is stored in

```
NET2 <- result1$NET2
```

Using the parameter Given_NET2, we could tune other parameters without the repeated computation of gene cooperation network. For example, we use the alpha=0.8 and lambda=0.5 to recompute the result:

```
time2 <- system.time(
  result2 <- markrank(exp_dataset, label, adj_matrix, alpha=0.8, lambda=0.5, trace=FALSE, Given_NET2=NE
  )
```

The running time of two results is:

```
time1
```

```
##    user  system elapsed
##   4.347   0.006   3.796
```

```
time2
```

```
##    user  system elapsed
##   0.161   0.000   0.160
```

The running time of result2 is far less than result1, because the result2 just contains the step of random walk algorithm. Now the new scores of top 10 markrank genes are:

```
s2 <- sort(result2$score, decreasing=TRUE)
s2[1:10]
```

```
##         46          3         12          5          4         99
## 0.07426386 0.06418216 0.06359423 0.04711116 0.04192298 0.02926330
##         13         18         74         11
## 0.02782042 0.02448161 0.02394709 0.02272578
```

The scores of pre-set differential expression genes are:

```
result2$score[deg_list]
```

```
##          5         46          7         12         11          4
## 0.04711116 0.07426386 0.01471943 0.06359423 0.02272578 0.04192298
##         13         99          3         74
## 0.02782042 0.02926330 0.06418216 0.02394709
```

The false discovery genes are:

```
setdiff(names(s2[1:10]), deg_list)
```

```
## [1] "18"
```

4 Fast construction of gene cooperation network

By using the input parameter d, markrank could reduce the computation time for constructing the gene cooperation network. Only the gene pairs, whose shortest distances in the biological network are less than d, participate in computation. For example, we could run

```
time3 <- system.time(
  result3 <- markrank(exp_dataset, label, adj_matrix, trace=F, d=2)
  )
```

The running time of two results is:

```
time1
```

```
##    user  system elapsed
##   4.347   0.006   3.796
```

```
time3
```

```
##    user  system elapsed
##   1.537   0.011   1.127
```

In this situation, the distance information of each gene pair can be found in output variable dis. For example, the distance matrix of gene 1 to 10 is:

```
result3$dis[1:10,1:10]
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
##  [1,]    0    1    2    3    2    3    3    3    3     4
##  [2,]    1    0    1    2    1    2    2    2    2     3
##  [3,]    2    1    0    1    1    2    2    2    2     3
##  [4,]    3    2    1    0    1    2    2    2    2     3
##  [5,]    2    1    1    1    0    1    1    1    1     2
##  [6,]    3    2    2    2    1    0    2    1    1     2
##  [7,]    3    2    2    2    1    2    0    2    1     3
##  [8,]    3    2    2    2    1    1    2    0    1     1
##  [9,]    3    2    2    2    1    1    1    1    0     2
## [10,]    4    3    3    3    2    2    3    1    2     0
```

The user should balance the computation depth with computation time to achieve a acceptable result.