

Combinatorial Nested Effects Models

April 1, 2016

Contents

1	The package epiNEM	2
1.1	Core functions	2
2	Validation studies	2
2.1	Comparing against other methods	2
3	Case Study	3
3.1	Data overview	3
3.2	Running the main function	4
3.3	Plotting the result	7
3.4	Triplet screening	7
4	Session Info	9

1 The package epiNEM

This package is an extension of the classic Nested Effects Models provided in package *nem*. Nested Effects Models is a pathway reconstruction method, which takes into account effects of downstream genes. Those effects are observed for every knockout of a pathway gene, and the nested structure of observed effects can then be used to reconstruct the pathway structure. However, classic Nested Effects Models do not account for double knockouts. In this package *epiNEM*, one additional layer of complexity is added. For every two parents of a gene, the relationship is evaluated and added to the model as a logic gate. Genetic relationships are represented by the logics OR (no relationship), AND (functional overlap), NOT (masking or inhibiting) and XOR (mutual prevention from acting on gene C).

1.1 Core functions

epiNEM() The main function *epiNEM* can be called using several different parameters. First of all, a filename is specified, containing a table with binary data. If no such data available, *random* needs to be specified. The vector *random* consists of the number of single as well as double knockouts, the number of reporter (downstream) genes, the false positive and false negative rate and lastly the number of replicates. With all those parameter, a corresponding dataset is simulated.

Furthermore, the user can choose whether to perform exhaustive search or use Greedy Hill Climbing (recommended for network > 5 genes), whereas for the latter a number of iterations has to be set as well.

epiNEM.PlotResults() In order to plot the winning pathway structure, *epiNEM.PlotResults* can be called. Its input is an object of *epiNEM*, containing information on the model structure, the introduced logics and its positions, the likelihood and the attachment of E-Genes.

2 Validation studies

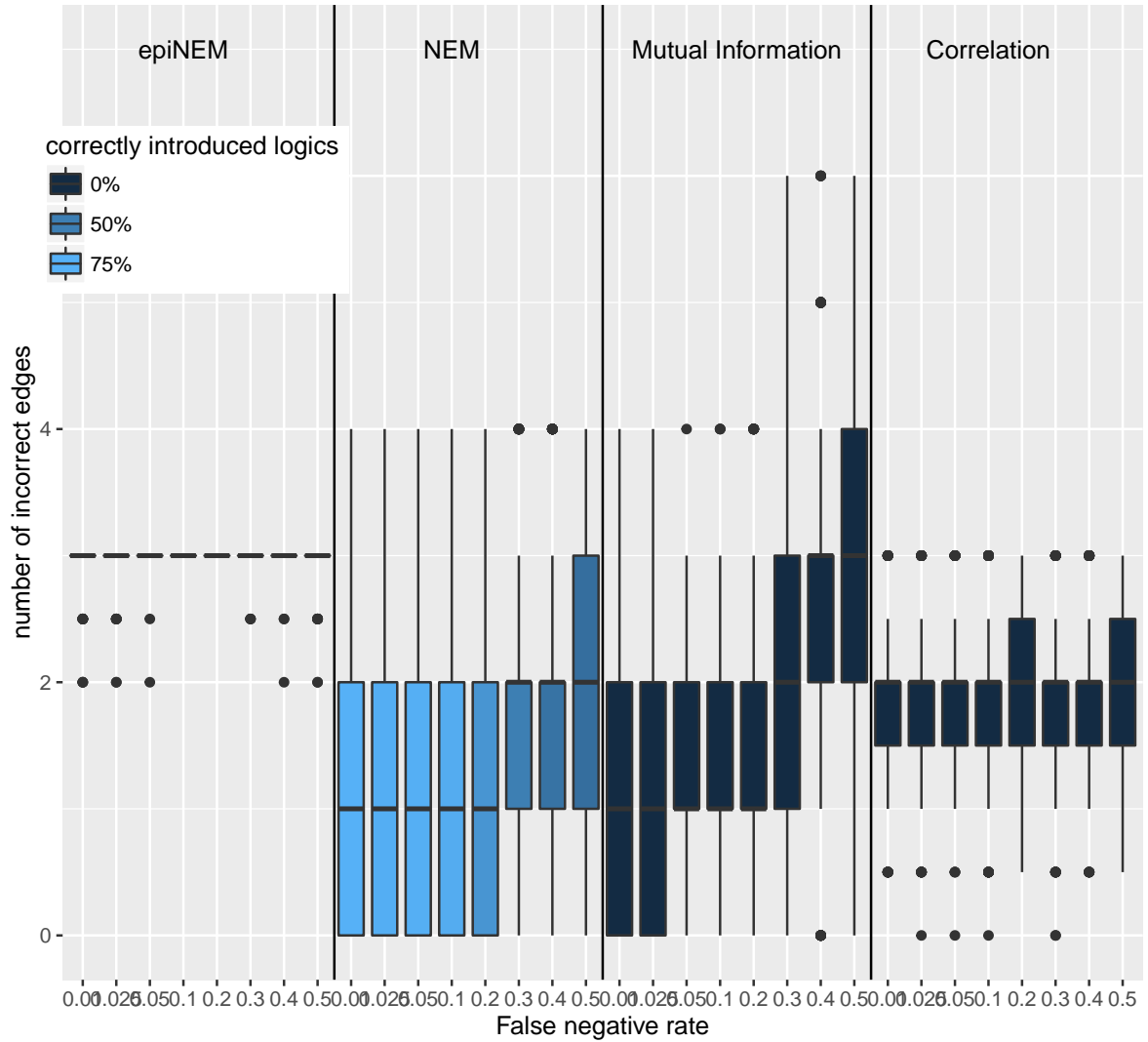
2.1 Comparing against other methods

In this section, we compare epiNEMs against the classic Nested Effects Models, as well as against network reconstruction by using Correlation and Mutual Information. We performed 500 simulations of 3 node networks for different false negative rates.

```
simul500 <- epiNEM.Simulations(random, 500)
```

Those networks were then learned by the above mentioned methods. For each method, we plot the number of wrong edges for each false negative rate.

Reconstruction of 500 simulated 3-node networks for varying false negative rates



3 Case Study

3.1 Data overview

The data provided in this package is based on the study of van Wageningen [reference]. The dataset consists of in total 150 viable protein kinase and phosphatase perturbations, as well as 15 viable double knockouts. DNA microarray expression profiles were generated under a single growth condition. For each mutant, two independent cultures have been measured and p-values according to relative expression change as well as mutant/wildtype ratios recorded. The raw data input consists of a matrix of p-values indicating the relative change of gene expression for each perturbation. The columns indicate gene deletions, the rows represent all downstream genes that were investigated for exhibiting effects.

3.2 Running the main function

First we load the R packages and load the example data file from http://www.holstegelab.nl/publications/sv/signaling_redundancy/. The data file contains a gene expression matrix of all kinase phosphatase mutants. For the analysis, a small number of genes can be selected, which we specify in a 'genelist'.

```
## load data file from local copy or from URL
address <-
  "http://www.holstegelab.nl/publications/sv/signaling_redundancy/downloads/DataS1.txt"
data <- data.table(read.csv(url(address), sep="\t", header=TRUE))
genelist <- toupper(c('hsl1', 'cla4', 'gin4', 'swel', 'hsl1.cla4'))

## read in genes from user and prepare for further analysis
read_in_genes <- function(genes){
  return(unlist(lapply(genes, function(x) {paste(x, '.del.vs..wt.1', sep='')})))
}
single <- read_in_genes(genelist)
```

Also, we only consider changes in expression which result in a p-value < 0.05 . All those are set to 1, the other ones to 0.

```
## binarise data: p-value < 0.05 and foldchange > 1.7 -> significant, else not
## set user specific p-value?!
#binarise <- function(x) {
#  ifelse((x < 0.05 & x > 0), 1, 0)
#}

## numericalize factors
as.numeric.factor <- function(x) {as.numeric(levels(x))[x]}
```

Last, we put all constraints together, meaning, we select and binarize the data.

```
data %<>% dplyr::select(-c(systematicName, geneSymbol)) %>%
  filter(-1) %>%
  mutate_each(funs(droplevels)) %>%
  mutate_each(funs(as.numeric.factor)) %>%
  data.frame()

# consider foldchange and gene expression
new <- matrix(0, ncol=dim(data)[2]/2, nrow=dim(data)[1])
colnames(new) <- colnames(data)[1:(dim(data)[2]/2)]
n=1
for (i in seq(1, dim(data)[2], by=2)){
  colnames(new)[n] <- colnames(data)[i+1]
  for (k in 1:dim(data)[1]){
```

```

    if ((abs(data[k,i]) >= 0.7) & (data[k,i+1] < 0.05 & data[k,i+1] > 0))
      new[k,n] <- 1
    else new[k,n] <- 0
  }
  n = n+1
}

# filter out 0-rows and set new colnames
new3 <- data.table(new[,which(colSums(new)!=0)])
new5 <- data.table(new3[which(rowSums(new3)!=0),])
new4 <- new5 %>% (function(x){setnames(x, names(x), toupper(names(x))) %>%
  gsub('.DEL.VS..WT.1', '', .) %>%
  gsub('.VS..WT', '', .) %>%
  gsub('.DEL.', '.', .)}; return(x)})

mutants      <- names(new4)
experiments  <- mutants[!grepl("\\.", mutants)]
dKO          <- mutants[which(!mutants %in% experiments)]
dKO          <- dKO[!grepl('\\..*\\. ', dKO)]
sKO <- unlist(strsplit(dKO, ".", fixed=TRUE))

sKO_add <- sKO[which(!sKO %in% names(new4))]

for (i in 1:length(sKO_add)){
  new4[, sKO_add[i]] <- c(rep(0,dim(new4)[1]))
}

PWadd <- genelists[which(!genelists %in% names(new4))]
for( i in 1:length(PWadd)){
  new4[, PWadd[i]] <- c(rep(0, dim(new4)[1]))
}

pathwayData <- new4 %>%
  select_(.dots = genelists)

```

Rownames correspond to sample identifiers. Columns indicate the mutants. A 1 corresponds to a change in expression for an identifier at a knockdown, a 0 represents no effect.

An already processed version of the complete dataset is provided in this package and accessible by calling *epiNEMdata*.

```

lognem <- epiNEM(filename=pathwayData, method = "greedy", nIterations=3)

## [1] "Progress (of 3 iterations):"
##
## 1....
## 2...
## 3.....

```

epiNEM introduces a logic gate for every node, which has exactly two parents. This logic gate represents the relationship of the parents, the interaction. If there is no interaction, as assumed in the classic NEMs, an OR-gate is introduced. Functional overlap of two genes is represented by an AND-gate, meaning both parents must be knocked out at the same time, in order for the signal to be interrupted. The logics NOTA and NOTB stand for a so called masking effect and the XOR-gate can be interpreted as some kind of mutual prevention of the parents from acting on the child gene.

The search space of epiNEM is now extended by those logic gates, and each model evaluated. Output of the algorithm is all information on the highest-scoring pathway structure.

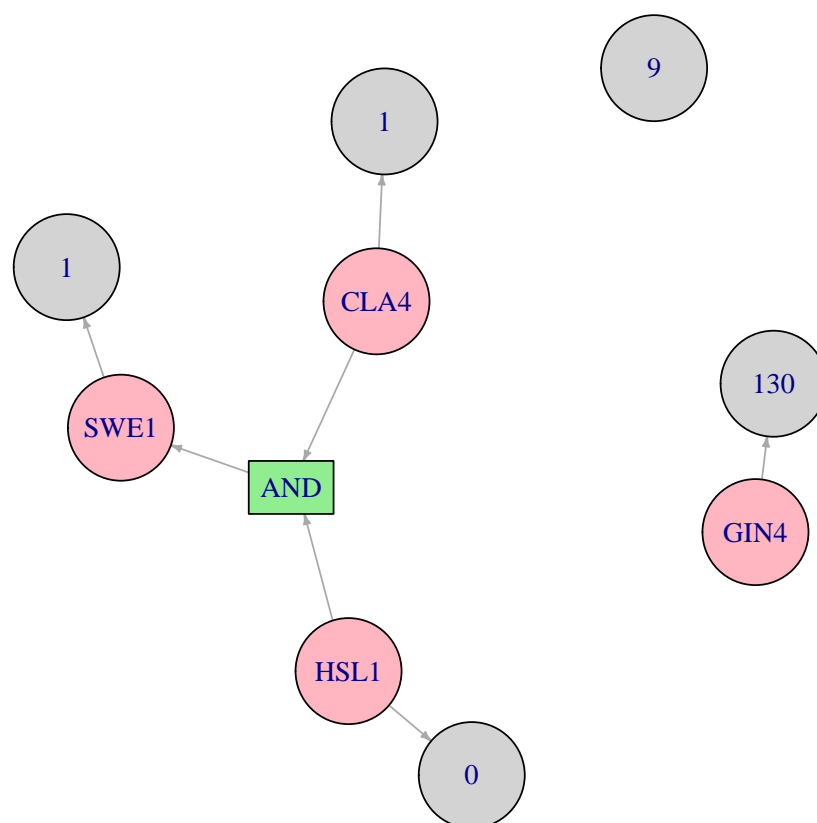
```
#lognem <- unlist(lognem, recursive = FALSE)
lognem

## $origModel
##      CLA4 GIN4 HSL1 SWE1
## CLA4    0    0    0    1
## GIN4    0    0    0    0
## HSL1    0    0    0    1
## SWE1    0    0    0    0
##
## $model
##      CLA4 GIN4 HSL1 SWE1
## HSL1      1    0    0    0
## CLA4      0    1    0    0
## GIN4      0    0    1    0
## SWE1      0    0    0    1
## HSL1.CLA4  1    0    1    1
##
## $logics
## Var1
## "AND"
##
## $column
##      c
## [1,] 4
##
## $score
## [1] -460.3305
##
## $EGeneset
##      noE
## CLA4    1
## GIN4  130
## HSL1    0
## SWE1    1
## null    9
```

3.3 Plotting the result

This winning pathway structure can further be visualized by the function `PlotResults()`, which is also available with the package `epiNEM`. The grey circles, which are attached to the genes, denote the number of E-genes attached to them. The unconnected circle includes all those, which could not uniquely be attached to a single S-gene.

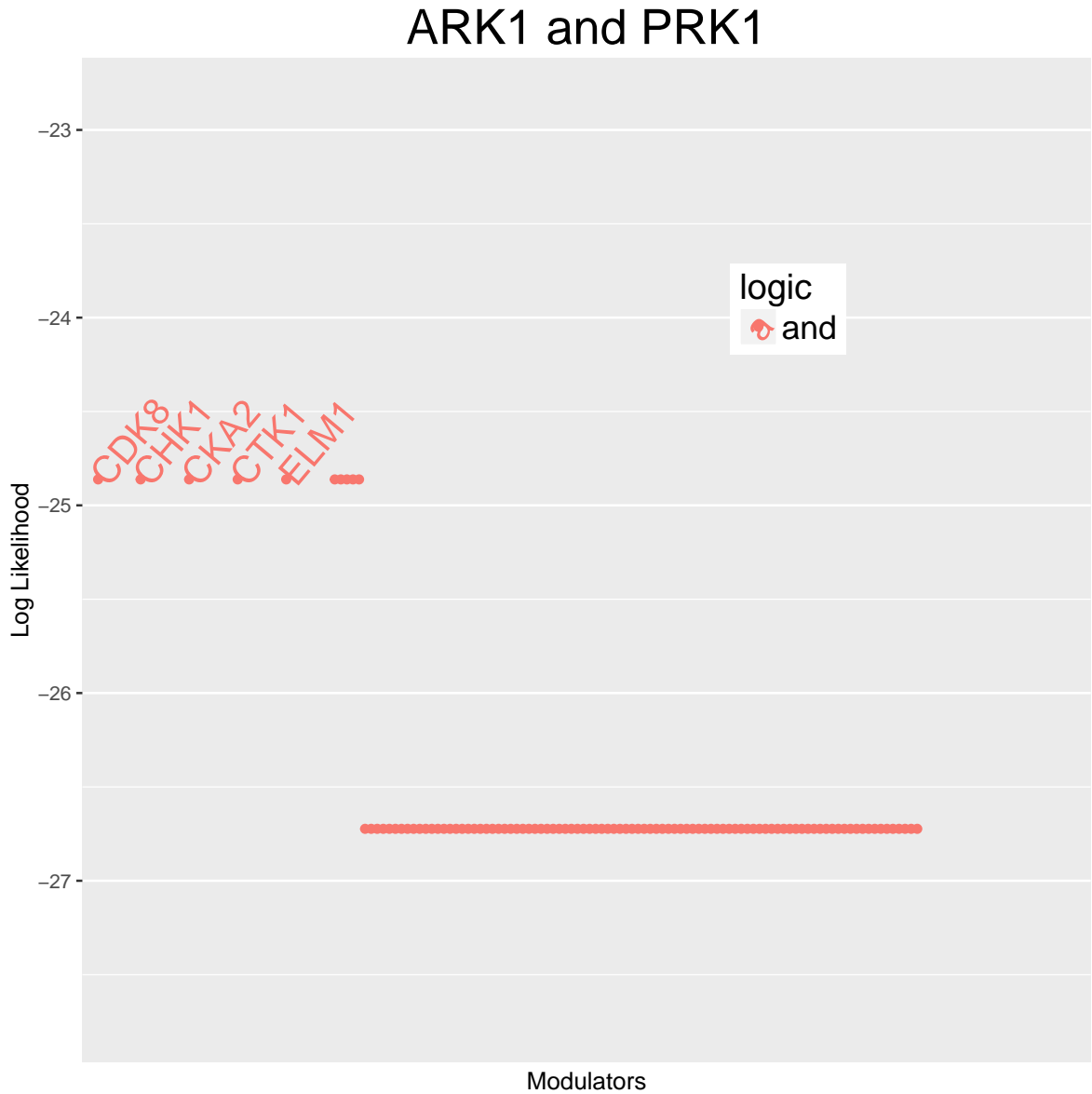
```
epiNEM::epiNEM.PlotResults(lognem)
```



3.4 Triplet screening

For analysing the mixed epistasis found by van Wageningen et. al., we constructed networks of only three nodes for every double knockout. We combined each pair with any remaining gene

and evaluate each of those small networks. Depending on the introduced logic, we were able to confirm most experimental results.



All other results can be found in SI.

4 Session Info

```
sessionInfo()

## R version 3.2.1 (2015-06-18)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.4 (Yosemite)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] parallel stats4 grid stats graphics grDevices utils
## [8] datasets methods base
##
## other attached packages:
## [1] epiNEM_0.1 gtools_3.5.0 dplyr_0.4.3
## [4] tabead_0.0.0.9000 org.Sc.sgd.db_3.2.3 RSQLite_1.0.0
## [7] DBI_0.3.1 AnnotationDbi_1.32.3 IRanges_2.4.6
## [10] S4Vectors_0.8.7 Biobase_2.30.0 BiocGenerics_0.16.1
## [13] e1071_1.6-7 igraph_1.0.1 BoolNet_2.1.1
## [16] data.table_1.9.6 ggplot2_2.1.0 tidyr_0.4.0
## [19] magrittr_1.5 knitr_1.12
##
## loaded via a namespace (and not attached):
## [1] Rcpp_0.12.3 formatR_1.2.1 plyr_1.8.3 highr_0.5.1
## [5] class_7.3-14 tools_3.2.1 digest_0.6.9 evaluate_0.8
## [9] memoise_0.2.1 gtable_0.2.0 stringr_1.0.0 roxygen2_5.0.1
## [13] devtools_1.9.1 R6_2.1.1 XML_3.98-1.4 codetools_0.2-14
## [17] scales_0.4.0 assertthat_0.1 colorspace_1.2-6 labeling_0.3
## [21] stringi_1.0-1 lazyeval_0.1.10 munsell_0.4.3 chron_2.3-47
```