Pigengene: Computing and using eigengenes

Habil Zare

Modified: 26 April, 2016. Compiled: September 28, 2021

Contents

1 Introduction

Gene expression profiling technologies such as microarray or RNA Seq provide valuable datasets, however, inferring biological information from these data remains cumbersome. *Pigengene* address two challenges:

1. **Curse of dimensionality:** The number of features in an expression profile is usually very high. For instance, there are about 20,000 genes in human. In contrast, the number of samples (patients) is often very limited in practice, and may not exceed a few hundreds. Yeung et al. have shown that standard data reduction methods such as principal component analysis (PCA) are not appropriate to directly apply on gene expression data [1]. Instead, *Pigengene* addresses this challenge by applying PCA on gene modules.

2. **Normalization:** Data produced using different technologies, or in different labs, are not easily comparable. *Pigengene* identifies *eigengenes*, informative biological signatures that are robust with respect to the profiling platform. For instance, it can identify the signatures (compute the eigengenes) on microarray data, and infer them on biologically-related RNA Seq data. The resulting signatures are directly comparable even if the set of samples (patients) are independent and disjoint in the two analyzed datasets.

2 How to run *Pigengene* ?

2.1 Installation

Pigengene is an *R* package that can be downloaded and installed from *Bioconductor* by the followig commands in R:

```
if (!requireNamespace("BiocManager", quietly=TRUE)) install.packages("BiocManager")
BiocManager::install("Pigengene")
```

Alternatively, if the source code is already available, the package can be installed by the following command in Linux:

```
R CMD INSTALL Pigengene_x.y.z.tar.gz
```

where x.y.z. determines the version. The second approach requires all the dependencies be installed manually, therefore, the first approach is preferred.

2.2 A quick overview

Pigengene identifies gene modules (clusters), computes an eigengene for each module, and uses these biological signatures as features for classification. The main function is `one.step.pigengene` which requires a gene expression profile and the corresponding conditions (types). Individual functions are also provided to facilitate running the pipeline in a customized way. The inferred biological signatures (eigengenes) are useful for supervised or unsupervised analyses.

2.3 What is an eigengene?

In most functions of this package, eigenegenes are computed or used as robust biological signatures. Briefly, each eigengene $\mathcal{E}$ is a weighted average of the expression of all genes in a given set of n genes (also known as a gene module or a cluster of genes).

$$\mathcal{E} = \alpha_1 g_1 + \alpha_2 g_2 + \cdots + \alpha_n g_n, \quad (1)$$

where α_i represents the weight corresponding to gene g_i. The weights are adjusted in a way that the explained variance is maximized. This guarantees that the loss in the biological information in minimized.

2.4 A toy example

For a quick start, the application of *Pigengene* pipeline on some leukemia dataset is demonstrated below [2]. The first step is to load the package and data in *R*:

```
library(Pigengene)
```

```
## Loading required package:  graph
```

```
## Loading required package:  BiocGenerics
```

```
## Loading required package:  parallel
```

```
##
## Attaching package:  'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

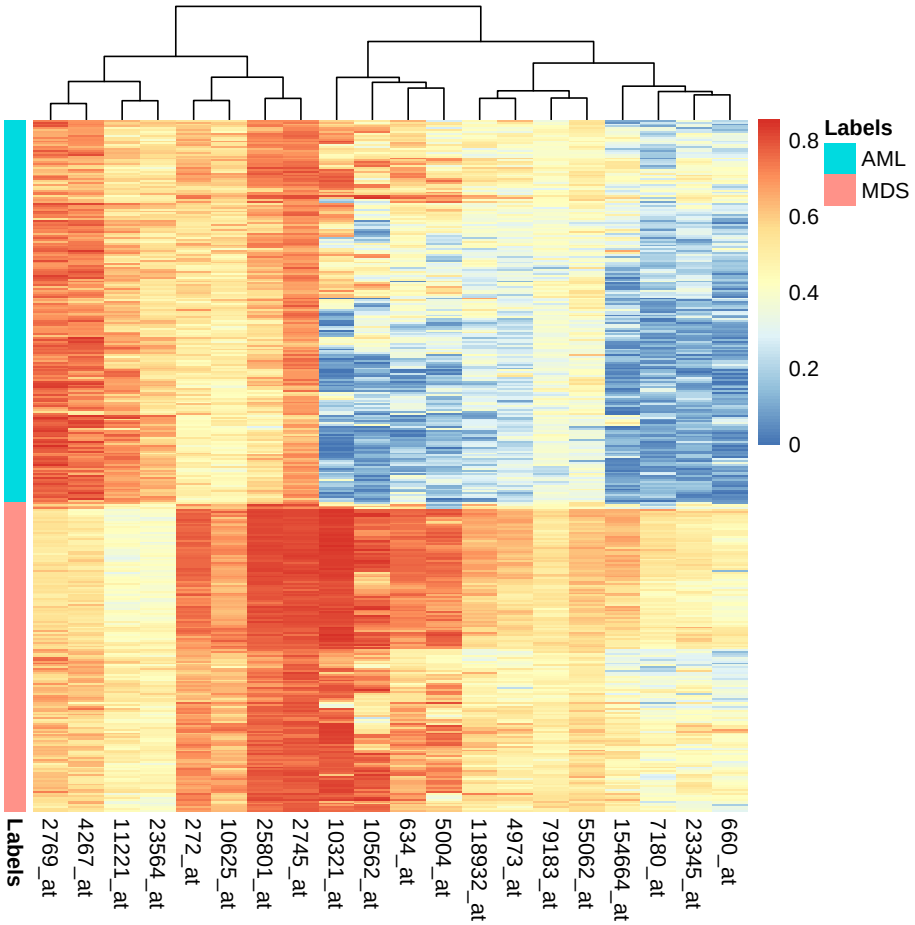
```
## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, basename, cbind, colnames, dirname, do.call,
##     duplicated, eval, evalq, get, grep, grepl, intersect,
##     is.unsorted, lapply, mapply, match, mget, order, paste, pmax,
```

```
##     pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply,
##     setdiff, sort, table, tapply, union, unique, unsplit, which.max,
##     which.min

## Loading required package:  BiocStyle

##

data(aml)
data(mds)
d1 <- rbind(aml,mds)
Labels <- c(rep("AML",nrow(aml)),rep("MDS",nrow(mds)))
names(Labels) <- rownames(d1)
Disease <- as.data.frame(Labels)
h1 <- pheatmap.type(d1[,1:20],annRow=Disease,show_rownames=FALSE)
```



Please note that the provided data in the package is sub-sampled for a quicker demonstration. For real applications, the expression of thousands of genes should be provided in order to the co-expression network analysis to be appro-

priate. It is common to first perform differential expression analysis, sort all the genes based on p-values, and use the top-third as the input [3]. Analyzing such input with *Pigengene* can take a few hours and may require 5-10 GB of memory. The following command runs Pigengene pipeline on the toy data:

```
p1 <- one.step.pigengene(Data=d1,saveDir='pigengene', bnNum=0, verbose=1,
      seed=1, Labels=Labels, toCompact=FALSE, doHeat=FALSE)
```

```
## Pigengene started analizing 366 samples using 1000 genes...
```

```
## Warning:  executing %dopar% sequentially:  no parallel backend registered
```

```
## Pigengenes...
```

```
## Pigengene plots in:
```

```
## /private/tmp/RtmpntQPoS/Rbuildbb70412531c8/Pigengene/vignettes/pigengene/plots
```

```
## Making decision trees...
```

```
## minPerLeaf:  8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37
```

```
## costs:
```

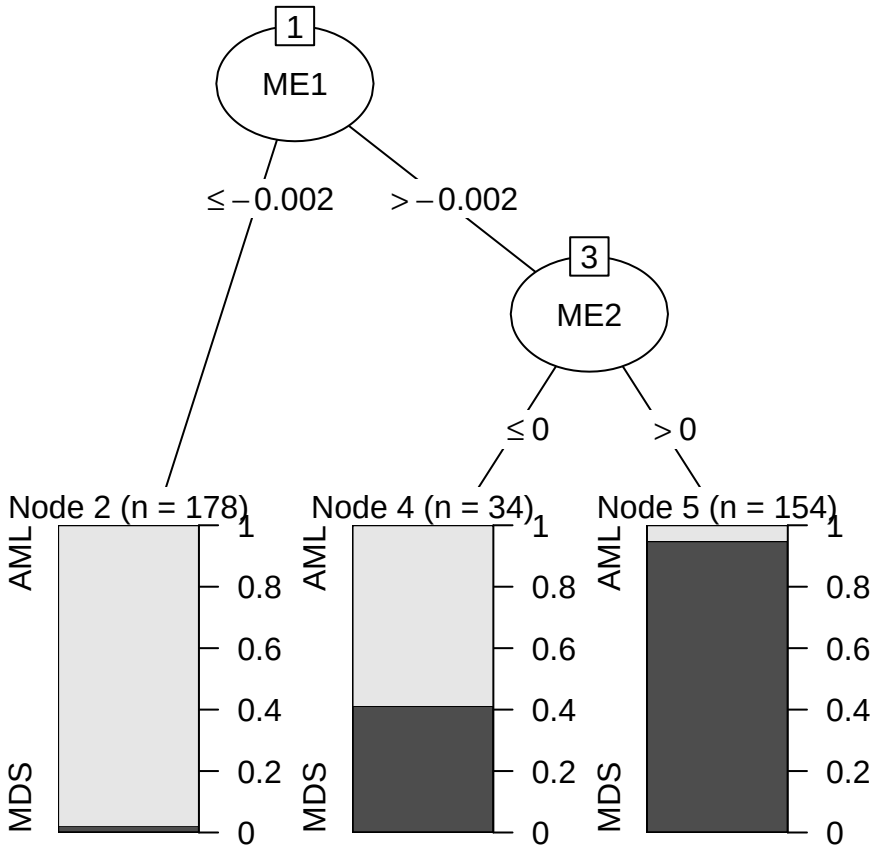
```
##     AML MDS
## AML   0   1
## MDS   1   0
```

```
## toCompact:  FALSE
```

Results and figures are saved in `pigengene` folder under the current directory. For more advanced applications, the user is encouraged to analyze the data step-by-step and customize the individual functions such as `compute.pigenegene` and `make.decision.tree`.

In addition to the provided decision trees, the user can also take alternative approaches to perform classification, clustering, survival analysis, etc. *using eigengenes as robust biological signatures (informative features)*. Eigengenes and other useful objects can be retrieved from the output. For instance, `c5treeRes` is a list containing the results of fitting decision trees to the eigengenes. As shown above, a couple of trees were fitted, one per a value for `minPerLeaf`. The following command plots the tree corresponding to 34, i.e., it was fitted requiring the minimum number of samples per every leaf to be at least 34.

```
plot(p1$c5treeRes$c5Trees[["34"]])
```

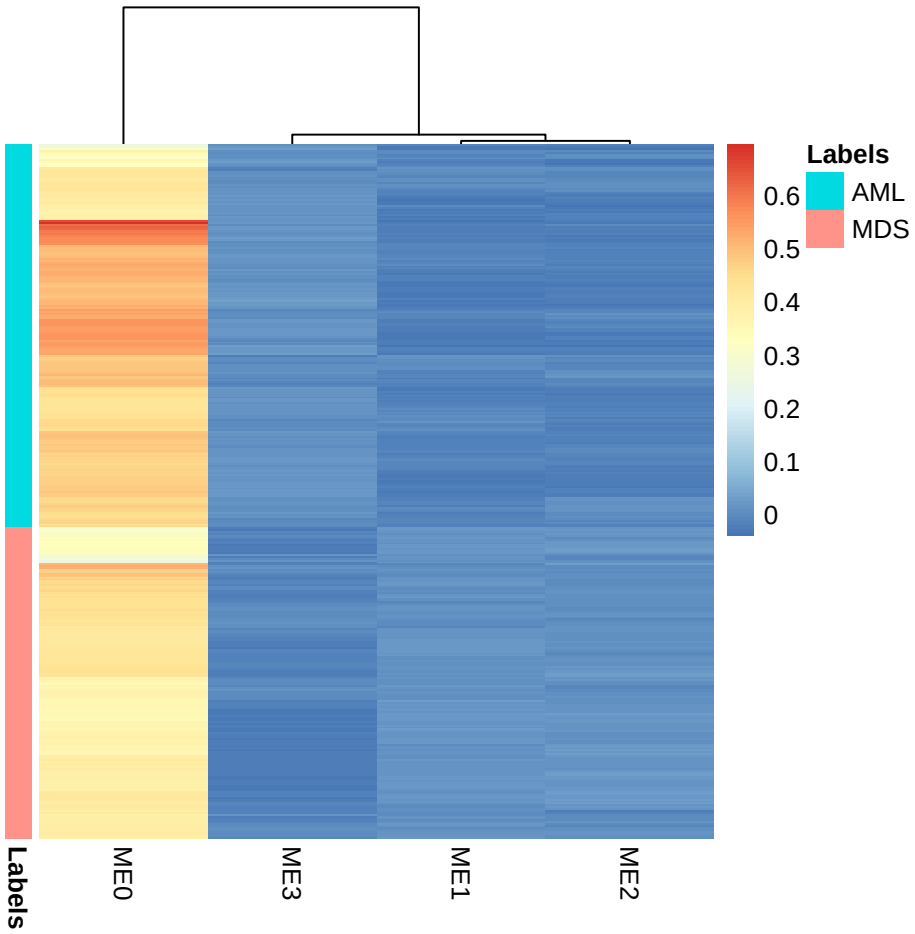


The tree corresponding to other values are saved in `pigengene` folder. Of note, is the `pigenegene` object that contains the matrix of inferred eigenegenes. Each row corresponds to a sample, and each column represents an eigengene.

```
dim(p1$pigengene$eigengenes)
```

```
## [1] 366   4
```

```
p1 <- pheatmap.type(p1$pigengene$eigengenes,annRow=Disease,show_rownames=FALSE)
```



2.5 Running the *Pigengene* pipeline step by step

If you are curious about the specific steps in the *Pigengene* pipeline, or you need to run some steps with different settings, you can follow the steps below. The results will be similar to the output of the `one.step.pigengene` function. The first step is quality control to make sure that the matrices have row and column names, and do not include too many `NA` values:

```
## QC:
checked <- check.pigengene.input(Data=d1, Labels=Labels)
DataI <- checked$Data
LabelsI <- checked$Labels
```

We oversample the data such that the number of cases in each condition is almost balanced.

```
wData <- balance(Data=DataI, Labels=LabelsI, verbose=1)$balanced
```

```
## Balancing...

## Oversampling to:   1818 of type AML ,  1804 of type MDS ,
```

Weighted gene co-expression network analysis (WGCNA) [4] does not assume any cutoff (i.e., hard threshold) on the correlation values. Instead, it raises the correlation values to a power so that the correlation values that are close to zero diminish. This hyperparameter is called β, and can be estimated using as follows:

```
betaI <- calculate.beta(RsquaredCut=0.8, Data=wData, verbose=1)$power

## Calculating beta...

## beta:  9

saveDir <- "steps" ## Results will be saved in this folder.
dir.create(saveDir)
```

Once we have an estimate for β, WGCNA can be done in one step using the following function to identify gene modules (i.e., clusters):

```
## WGCNA
wgRes <- wgcna.one.step(Data=wData, seed=1, power=betaI,
      saveDir=saveDir, verbose=1)

## Identifying the modules (WGCNA)...

## power= 9

## 4 modules were identified with the following sizes:

## modules
##   0   1   2   3
##   1 441 333 225

## save(net, file='steps/net.RData')

## save(wgOneStep, file='steps/wgOneStep.RData')
```

The output of `wgcna.one.step` is a list of objects including `modules`, which is a numeric vector named with genes. Genes that map to the same numeric value are considered to be in the same module. We use this information to compute an eigengene for every module.

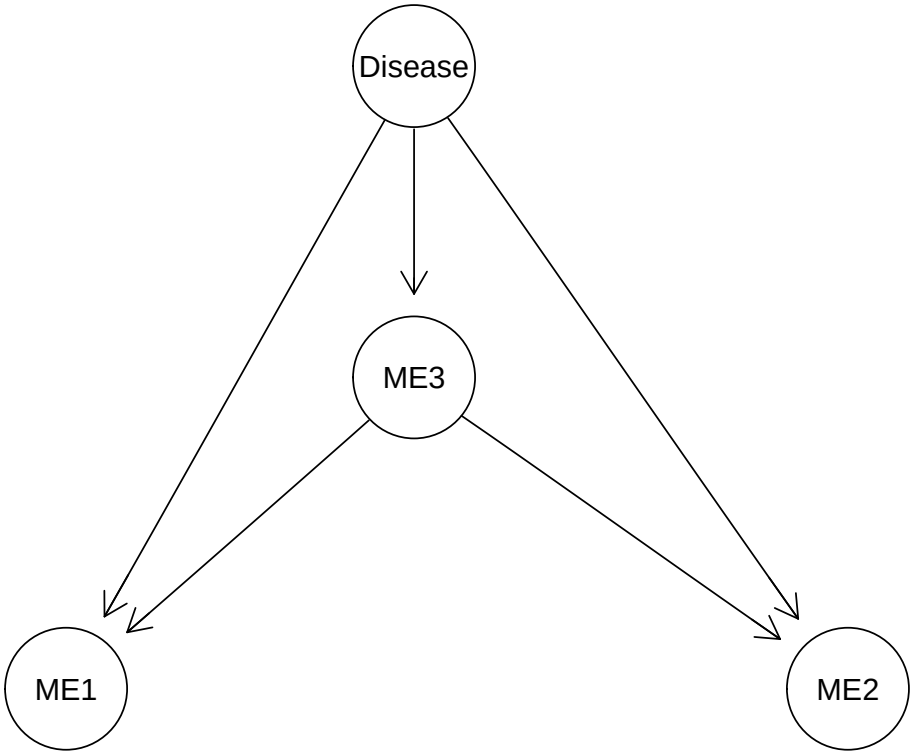
```
## Eigengenes:
pigengene <- compute.pigengene(Data=DataI, Labels=LabelsI,
                saveFile=file.path(saveDir, 'pigengene.RData'),
```

```
                  modules=wgRes$modules, verbose=1)
## Pigengenes...
## Pigengene plots in:
## steps/plots
class(pigengene)
## [1] "pigengene"
print(dim(pigengene$eigengenes)) ##This is the eigengenes matrix.
## [1] 366   4
print(pigengene$eigengenes[1:3,1:4])
##                     ME1          ME2          ME3       ME0
## GSM376049 -0.006958185 -0.002890136 -0.006020223 0.4557040
## GSM376050 -0.003067442 -0.017728108  0.017231152 0.4298970
## GSM376051 -0.004466249  0.008799329  0.003679651 0.4251949
```

The number of columns in the eigengene matrix is equal to the number of modules, and the number of rows is equal to the number of samples.

Eigengenes can be used as robust biological signatures (i.e., features in machine learning terms) for classification, clustering, exploratory analysis, etc. For example, we can use them as random variables to fit a Bayesian network to data [5].

```
## Learning the BN structure:
library(bnlearn)
learnt <- learn.bn(pigengene=pigengene, bnPath=file.path(saveDir, "bn"),
                bnNum=10, ## In real applications, at least 100-1000.
                seed=1, verbose=1)
## learn.bn() with bnNum= 10 started at:
## 2021-09-28 14:05:42
## Warning in discretize(as.data.frame(d), method = "interval", breaks
= length(unique(Disease))):  at least one variable should be continuous
## learn.bn() took:
## 11.24964 secs
```




```
BN <- learnt$consensus1$BN
```

The `learn.bn` function has many arguments. See the corresponding documentation and publication [5] for technical details. Because usually thousands of individual networks are needed, it is wise to train many models in parallel on a cluster, which can be done with appropriate settings for the `learn.bn` input arguments. We can replot the consensus Bayesian network, which is already saved on disk, using the `draw.bn` function.

```
drawn <- draw.bn(BN)
## By construction, the Disease node can have no parents.
```

The `learn.bn` function tries to find the best structure for the optimum Bayesian network. The conditional probability tables are yet to be inferred from the data.

```
## Fit the parameters of the Bayesian network:
fit <- bn.fit(x=BN, data=learnt$consensus1$Data, method="bayes", iss=10)
##where learnt$consensus1$Data is the discretized data matrix.

## The conditional probability table for one of the children of the Disease node:
selectedFeatures <- children("Disease", x=BN)
print(fit[[selectedFeatures[1]]])
```

```
##
##   Parameters of node ME1 (multinomial distribution)
##
## Conditional probability table:
##
## , , Disease = AML
##
##                          ME3
## ME1                       [-0.029534,-0.00131458] (-0.00131458,0.0151399]
##   [-0.0361341,-0.0184773]             0.136904762             0.149032992
##   (-0.0184773,0.00862247]             0.619047619             0.824800910
##   (0.00862247,0.0308412]              0.244047619             0.026166098
##                          ME3
## ME1                       (0.0151399,0.0336717]
##   [-0.0361341,-0.0184773]           0.645833333
##   (-0.0184773,0.00862247]           0.348039216
##   (0.00862247,0.0308412]            0.006127451
##
## , , Disease = MDS
##
##                          ME3
## ME1                       [-0.029534,-0.00131458] (-0.00131458,0.0151399]
##   [-0.0361341,-0.0184773]             0.004219409             0.017006803
##   (-0.0184773,0.00862247]             0.163713080             0.598639456
##   (0.00862247,0.0308412]              0.832067511             0.384353741
##                          ME3
## ME1                       (0.0151399,0.0336717]
##   [-0.0361341,-0.0184773]           0.119047619
##   (-0.0184773,0.00862247]           0.547619048
##   (0.00862247,0.0308412]            0.333333333
```

The fitted Bayesian network can be used for predicting the labels (i.e., values of the Disease node).

```
l2 <- predict(object=fit, node="Disease", data=learnt$consensus1$Data, method="bayes-lw")
table(LabelsI, l2)
```

```
##        l2
## LabelsI AML MDS
##     AML 196   6
##     MDS  29 135
```

Eigengenes can also be used in predictive models simpler than a Bayesian network. For example, a decision tree can be fitted using the `make.decision.tree` function [6].

```
## Decision trees:
treePath <- file.path(saveDir, 'C5Trees')
dir.create(path=treePath)
treeRes <- make.decision.tree(pigengene=pigengene, Data=DataI,
               selectedFeatures=selectedFeatures, saveDir=treePath,
               verbose=1, toCompact=FALSE)

## Making decision trees...

## minPerLeaf:  8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37

## costs:

##     AML MDS
## AML   0   1
## MDS   1   0

## toCompact:  FALSE

## Warning in normalizePath(dir):  path[1]="/private/tmp/RtmpntQPoS/Rbuildbb70412531c8/Pige
No such file or directory

## Warning in normalizePath(dir):  path[1]="/private/tmp/RtmpntQPoS/Rbuildbb70412531c8/Pige
No such file or directory

## Warning in normalizePath(dir):  path[1]="/private/tmp/RtmpntQPoS/Rbuildbb70412531c8/Pige
No such file or directory
```

If `selectedFeatures="All"`, the `make.decision.tree` function automatically selects a "minimal" subset of eigengenes in order to prevent overfitting.

2.6 Citation

The methodology and an interesting application of *Pigengene* on studying hematological malignancies is presented in the following reference [6].

```
citation("Pigengene")
```

To cite package 'Pigengene' in publications use:

Amir Foroushani et al.(2016) Large-scale gene network analysis reveals the significance of extracellular matrix pathway and homeobox genes in acute myeloid leukemia: an introduction to the Pigengene package and its applications, Foroushani et al., BMC Medical Genomics. URL: https://bmcmedgenomics.biomedcentral.com/articles/10.1186/s1292
017-0253-6.

A BibTeX entry for LaTeX users is

@Article, author = Amir Foroushani and et al., title = Large-scale gene network analysis reveals the significance of extracellular matrix pathway and homeobox genes in acute myeloid leukemia: an introduction to the Pigengene package and its applications, journal = BMC Medical Genomics, year = 2017, volume = 10, number = 1, pages = 16, month = 3,

3 Session Information

The output of `sessionInfo` on the system that compiled this document is as follows:

```
toLatex(sessionInfo())
```

- R version 4.1.1 (2021-08-10), `x86_64-apple-darwin17.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Running under: `macOS Mojave 10.14.6`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.0.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, utils
- Other packages: BiocGenerics 0.38.0, BiocStyle 2.20.2, Pigengene 1.18.10, bnlearn 4.7, graph 1.70.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.54.1, Biobase 2.52.0, BiocManager 1.30.16, Biostrings 2.60.2, C50 0.1.5, Cubist 0.3.0, DBI 1.1.1, Formula 1.2-4, GO.db 3.13.0, GenomeInfoDb 1.28.4, GenomeInfoDbData 1.2.6, Hmisc 4.5-0, IRanges 2.26.0, KEGGREST 1.32.0, MASS 7.3-54, Matrix 1.3-4, R6 2.5.1, RColorBrewer 1.1-2, RCurl 1.98-1.5, RSQLite 2.2.8,

Rcpp 1.0.7, Rgraphviz 2.36.0, S4Vectors 0.30.1, WGCNA 1.70-3, XVector 0.32.0, assertthat 0.2.1, backports 1.2.1, base64enc 0.1-3, bit 4.0.4, bit64 4.0.5, bitops 1.0-7, blob 1.2.2, cachem 1.0.6, checkmate 2.0.0, cluster 2.1.2, codetools 0.2-18, colorspace 2.0-2, compiler 4.1.1, crayon 1.4.1, data.table 1.14.2, digest 0.6.28, doParallel 1.0.16, dplyr 1.0.7, dynamicTreeCut 1.63-1, ellipsis 0.3.2, evaluate 0.14, fansi 0.5.0, farver 2.1.0, fastcluster 1.2.3, fastmap 1.1.0, foreach 1.5.1, foreign 0.8-81, gdata 2.18.0, generics 0.1.0, ggplot2 3.3.5, glue 1.4.2, grid 4.1.1, gridExtra 2.3, gtable 0.3.0, gtools 3.9.2, highr 0.9, htmlTable 2.2.1, htmltools 0.5.2, htmlwidgets 1.5.4, httr 1.4.2, impute 1.66.0, inum 1.0-4, iterators 1.0.13, jpeg 0.1-9, knitr 1.34, lattice 0.20-45, latticeExtra 0.6-29, libcoin 1.0-9, lifecycle 1.0.1, magrittr 2.0.1, matrixStats 0.61.0, memoise 2.0.0, munsell 0.5.0, mvtnorm 1.1-2, nnet 7.3-16, partykit 1.2-15, pheatmap 1.0.12, pillar 1.6.3, pkgconfig 2.0.3, plyr 1.8.6, png 0.1-7, preprocessCore 1.54.0, purrr 0.3.4, reshape2 1.4.4, rlang 0.4.11, rmarkdown 2.11, rpart 4.1-15, rstudioapi 0.13, scales 1.1.1, splines 4.1.1, stats4 4.1.1, stringi 1.7.4, stringr 1.4.0, survival 3.2-13, tibble 3.1.4, tidyselect 1.1.1, tools 4.1.1, utf8 1.2.2, vctrs 0.3.8, xfun 0.26, yaml 2.2.1, zlibbioc 1.38.0

References

[1] Ka Yee Yeung and Walter L. Ruzzo. Principal component analysis for clustering gene expression data. *Bioinformatics*, 17(9):763–774, 2001.

[2] Ken I Mills, Alexander Kohlmann, P Mickey Williams, Lothar Wieczorek, Wei-min Liu, Rachel Li, Wen Wei, David T Bowen, Helmut Loeffler, Jesus M Hernandez, et al. Microarray-based classifiers and prognosis models identify subgroups with distinct clinical outcomes and high risk of aml transformation of myelodysplastic syndrome. *Blood*, 114(5):1063–1072, 2009.

[3] Bin Zhang, Chris Gaiteri, Liviu-Gabriel Bodea, Zhi Wang, Joshua McElwee, Alexei A Podtelezhnikov, Chunsheng Zhang, Tao Xie, Linh Tran, Radu Dobrin, et al. Integrated systems approach identifies genetic nodes and networks in late-onset alzheimer's disease. *Cell*, 153(3):707–720, 2013.

[4] Peter Langfelder and Steve Horvath. Wgcna: an r package for weighted correlation network analysis. *BMC bioinformatics*, 9(1):559, 2008.

[5] Rupesh Agrahari, Amir Foroushani, T Roderick Docking, Linda Chang, Gerben Duns, Monika Hudoba, Aly Karsan, and Habil Zare. Applications of bayesian network models in predicting types of hematological malignancies. *Scientific reports*, 8(1):6951, 2018.

[6] A Foroushani, R Agrahari, R Docking, A Karsan, and H Zare. Large-scale gene network analysis reveals the significance of extracellular matrix pathway and homeobox genes in acute myeloid leukemia. *In preparation*.