Supplementary Materials

overlappingCGM: Automatic detection and analysis of overlapping co-expressed gene modules

Quang-Huy Nguyen[1] & Duc-Hau Le[1,2*]

[1]Department of Computational Biomedicine, Vingroup Big Data Institute, Hanoi, Vietnam.
[2]College of Engineering and Computer Science, VinUniversity, Hanoi, Vietnam.

* To whom correspondence should be addressed. Tel: (84)912324564; Email: hauldhut@gmail.com

Table of Contents

1. Supplementary Methods

1.1 ICA and IPCA algorithms

Here we have decided to integrate ICA [1-3] and IPCA [4], assigned to the decomposition methods, into overlappingCGM based on evidence reporting that they are the most appropriate to detect co-expressed gene modules. Importantly, we have excluded principle component analysis (PCA) [5] method for the following reasons: (i) PCA assumes that gene expression follows a Gaussian distribution; however, many recent studies have demonstrated that microarray gene expression measurements follow instead a non-Gaussian distribution [6-9], (ii) The idea behind PCA is to decompose a big matrix into the product of several sub-matrices, and then retain the first few components which have maximum amount of variance. Mathematically, this helps to do dimension reduction, noise reduction, but the highest variance may be inappropriate to the biological reality [10, 11]. From that, we have chosen ICA and IPCA that may beat the limitations of PCA. Figure S1 shows the example output of the decomposition methods, including signatures and patterns with the number of components set to two.

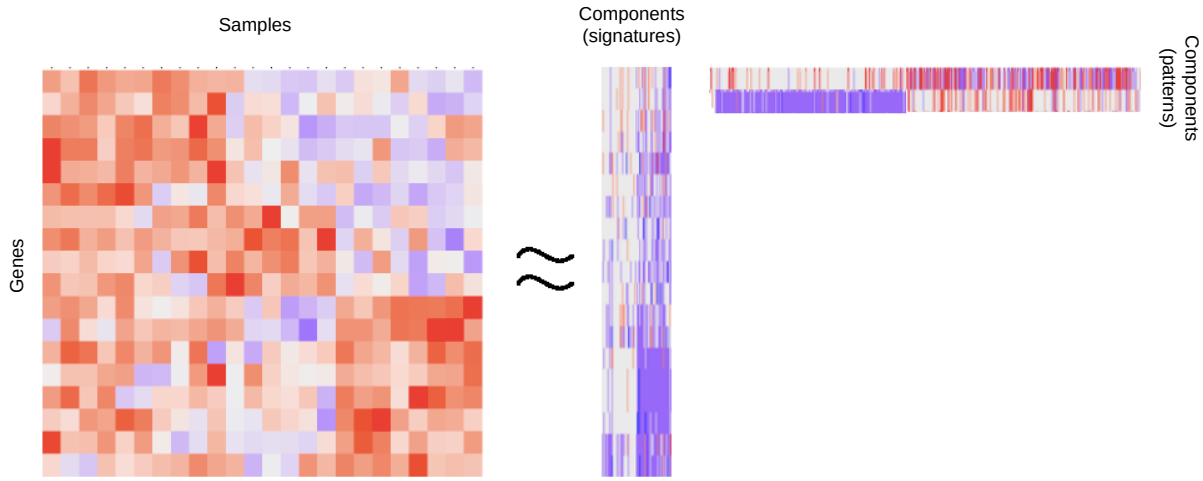

Figure S1. Heatmap shows general output of the ICA and IPCA methods with *k*=2 (*k*, number of principle components).

1.1.1 ICA algorithm

There is a number of "source signals" (or "original signals" or "statistically independent components"), but due to some external circumstances, only linear mixtures of the source signals are gained. The goal of ICA [1-3] is to recover independent source signals from a mixture of signals. ICA is simply written as a matrix decomposition:

$$X = SA$$

Where $X = x_{mn}$ is normalized data matrix, $S = s_{mk}$ is the impact of component k on gene expression level m (the first matrix product, vertical rectangle in Figure S1), and $A = a_{kn}$ is expression change of that component in individual n (the second matrix product, horizontal rectangle in Figure S1). These k components must be as statistically independent as possible.

In the first matrix product, ICA creates nearly sparse signatures, meaning that the majority of their impacts close to 0; nevertheless, each of them possess several genes for which its impacts are considerably different from 0 (genes being at both extremes of the distribution). Mathematically, each of the k columns of sub-matrix S, considered as "signatures" of the hidden biological processes, is a linear combination of the component on each gene expression (each row of S). Especially, different signatures may possess a overlapping set of those genes

In the second matrix product, rows of sub-matrix A represent expression change of the k components in individuals. In other words, each pattern reflects the change in expression level in individuals.

1.1.2 IPCA algorithm

The idea of IPCA [4] is a combination of ICA and PCA. Specifically, it first uses PCA in order to reduce the dimension of the data and generate the loading vectors, then applies the fastICA algorithm to those loading vectors, rendering independent principle components (IPC). this make IPCA more robust to noise.

1.2 daBEST algorithm

After running ICA or IPCA for extracting the components is usually to specify how many components are optimal. The goal is to identify the minimal set of components that can be used to describe the co-expressed gene modules. Importantly, there is not a gold standard for this work. In this study, we present the function *daBEST* in overlappingCGM in the hope of helping users to overcome the problem. The statistical method behind the function *daBEST* is based on random permutations adapted from [12] (Figure S2a). *daBEST* first permutes individuals independently for each gene expression in the real/original data (the certain number of permutations defined by users, say P times) and generates P corresponding random matrices. Then, it applies the singular value decomposition (SVD) to the original matrix and every resulting random matrices, rendering a vector of singular values for each matrix. Suppose we have k principle components $C = (1, \dots, k)$. We turn those vectors to vectors of percentage of variance explained (%VE) as follows:

$$\%VE_p = \frac{\sigma_p}{\sum_{i=1}^{k} \sigma_i}$$

where $\%VE_p$ is the p^{th} %VE and σ_p is the p^{th} singular value in that component, $p \in C$. Next, average (over P random matrices) the first %VE, the second %VE, …, the kth %VE. At last, the (first) intersection between a curve obtained from %VE of the real matrix and a curve obtained from the average %VE of all the random matrices plotted in the screeplot is the optimal component number, considering that beyond this number, components mostly reflected noise. (Figure S2b). If the intersection point is a decimal number on the horizontal axis, *daBEST* rounds up that number to the next higher one.

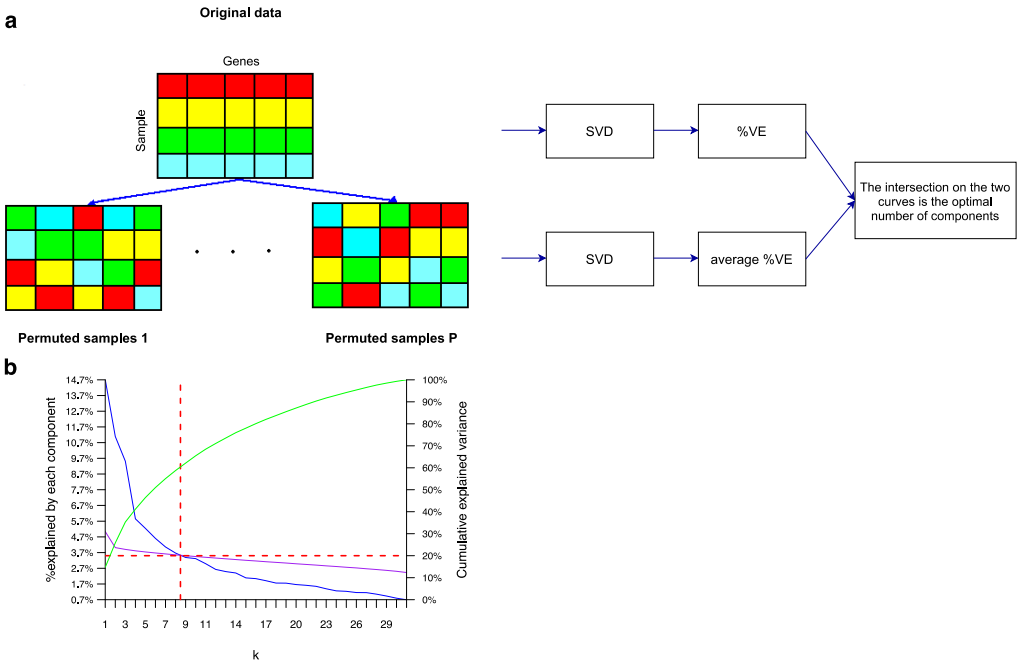

Figure S2. (a) pipeline of the permutation procedure of *daBEST* where all the samples (rows) are permuted independently for each gene (each column). It then applies the singular value decomposition (SVD) to both the original data and every random matrices, and infers the optimal number of components based on the intersection between the two curves created by the percentage of variance explained (%VE) of the real data and average %VE of all the *P* random matrices. (b) The example screeplot illustrates %VE (left Y-axis) and the culmulative VE (right Y-axis, calculated based on %VE) by the principal components *k* (X-axis) of the SVD. The blue solid curve shows %VE and the green curve shows the cumulative VE of the real data matrix across principle components. The purple solid curve presents average %VE of the *P* random matrices across principle components. The optimal number of components is determined at the intersection between the blue curve (observed variance) and the purple curve (variance expected under random). The optimal number is approximately 9 (red vertical line).

It is crucial to consider how big *P* needs to be as Horn *et al* [12] have also mentioned this issue. Moreover, simulation studies have reported that $P < 99$ have low power, whereas $P \geq 499$ is recommended [13-15]. Additionally, $P = 1000$ has been proved to produce good results [15, 16]. From that, 1000 is left as default in *daBEST*. Users should be aware that an increase in numbers of permutations will lead to a significant increase in computation time with possibly only a small gain on the approximation.

Also, to suggest which method (ICA or IPCA) should be selected, *daBEST* runs idenpendently the two methods for the input data matrix and check whether all principle components generated from each method have all their kurtosis values < 3 or not. If a certain method issues all principle components whose kurtosis < 3, it should be ignored.

2. Supplementary Results

overlappingCGM is an R package can be freely available on Github repository (…). To show a straightforward use of using overlappingCGM, we re-use -omic data used in our prior study [17], downloaded from our github repository (…) or the cBioPortal for Cancer Genomics (http://www.cbioportal.org) [18, 19], including gene expression (EXP; n = 1,904), in a cohort of breast cancer patients. Besides, we also apply our tool to mouse metabolic syndrome containing liver EXP from female mice (n =135) [20]. The raw data and R codes for pre-processing processes can be seen in (…).

2.1 human breast cancer

In our previous study [17], we have discovered 31 driver genes associated with breast cancer. Here we first preprocess the gene expression matrix (`exp`) following the requirement of overlappingCGM, including its rows are the 1904 patients and its columns are those 31 genes. Note that normalization does not need to be done in advance because overlappingCGM can do this task on its own. Also, we must preprocess the clinical data (`clinicalEXP`), including the selection of clinical features of interest, and turning its data structure into required one: its rows are the 1904 samples and its columns are clinical features of those patients. In present study, we re-select three clinical features: the number of lymph (`lymph`), Nottingham prognostic index (`npi`), and tumor stages (`stage`) selected in the previous work.

Load necessary libraries

```
# Load necessary libraries
library("dynamicTreeCut") # Module identification
library("flashClust")      #Fast   implementation   of   hierarchical
clustering
library("Hmisc")  #perform variables clustering
library("WGCNA")   #WGCNA tool
library(purrr); library(dplyr);  #data processing and manipulation
library("cluster")    # compute agglomerative coefficient
library(overlappingCGM)

#load raw data
exp = read.table('data_mRNA_median_Zscores.txt', sep = '\t',
check.names = FALSE, header = TRUE, row.names = NULL)

clinical = read.table('data_clinical_patient.txt', sep = '\t',
check.names = FALSE, header = TRUE, row.names = 1,fill=TRUE)

#31 identified driver genes
driver=c("MAP2K4", "ARID1A", "PIK3CA", "TBX3", "MAP3K1", "TP53",
"AKT1", "GATA3", "CDH1", "RB1", "CDKN1B", "NCOR1", "CDKN2A",
"ERBB2", "KRAS", "BRCA2", "BAP1", "PTEN", "CBFB", "KMT2C", "RUNX1",
"NF1", "PIK3R1", "ERBB3", "FOXO3", "SMAD4", "GPS2", "AGTR2",
"ZFP36L1", "MEN1","SF3B1")
length(driver) #31 driver genes

#only keep the 31 driver genes in exp
exp=exp %>%
  dplyr::filter(.$Hugo_Symbol %in% driver) %>%
  tibble::column_to_rownames('Hugo_Symbol') %>%
  dplyr::select(-Entrez_Gene_Id)

#check dimension
dim(exp) # 31 1904
dim(clinical) # 2509   21

#match patients sharing between exp versus clinical
#exp are the matrix whose rows are samples and columns are genes
exp = exp[,intersect(colnames(exp), rownames(clinical))] %>% t()
```

```
clinicalEXP = clinical[intersect(rownames(exp), rownames(clinical)),
]

#preprocess clinicalEXP
clinicalEXP = clinicalEXP %>%
  tibble::rownames_to_column('sample') %>%
  dplyr::select(c(sample, LYMPH_NODES_EXAMINED_POSITIVE, NPI,
stage)) %>%
  tibble::column_to_rownames('sample')

colnames(clinicalEXP)[1:3] =  c("lymph", "npi", "stage")
str(clinicalEXP)
# 'data.frame': 1904 obs. of  5 variables:
# $ lymph : int  1 5 8 1 0 1 0 2 0 6 ...
# $ npi   : num  4.04 6.03 6.03 5.04 3.05 ...
# $ stage : int  2 2 3 2 2 2 1 2 2 4 ...
```

We apply WGCNA to the data with all tuning parameters are left as default, except for the minimum module size at 10 due to having the 31 genes only.

```
#WGCNA
wgcna = blockwiseModules(exp, power = 6,
    TOMType = "unsigned", minModuleSize = 10,
    reassignThreshold = 0, mergeCutHeight = 0.25,
    numericLabels = TRUE, pamRespectsDendro = FALSE,
    verbose = 3)
```

For overlappingCGM, we first use `daBEST` to determine the optimal number of components and which method should be selected. Then, `daBEST` suggests choosing nine components and the ICA method. We then input the expression data `exp` and clinical data `clinicalEXP` to overlappingCGM with those suggested selections. Here we feed `ICA-Zscore` to the `method` argument as an example since this post-processing step do not create the grey module.

```
# overlappingCGM
daBEST(data = exp)
# >> overlappingCGM suggests choosing the optimal number of
components is: 9
# >> overlappingCGM also suggests using ICA for your case.

cgm=overlappingCGM(data = exp, clinical = clinicalEXP, ncomp = 9,
                       method = 'ICA-Zscore')
```

For iWGCNA, we use R code released on our Github (https://github.com/hauldhut/drivergene).

```
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
enableWGCNAThreads() ### Allowing parallel execution

#we choose the soft-thresholding of 6 based on our prior work
softPower = 6;
adjacency = adjacency(exp, power = softPower,
                      type = "signed");

# Turn adjacency into topological overlap
```

```
TOM = TOMsimilarity(adjacency, TOMType = "signed");
dissTOM = 1-TOM

# Hierichical clustering
#Seek the optimal agglomeration method
#methods to assess
m <- c( "average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")
# function to compute agglomerative coefficient
set.seed(25896)
ac <- function(x) {
  agnes(exp, method = x)$ac
}

map_dbl(m, ac)  # Agglomerative coefficient of each agglomeration
method
# average    single  complete      ward
# 0.4136737 0.3510251 0.4815638 0.5563910

#assign gene names from adjacency to dissTOM
rownames(dissTOM) = rownames(adjacency)
colnames(dissTOM) = colnames(adjacency)
# Call the hierarchical clustering function
geneTree = hclust(as.dist(dissTOM), method = "ward.D2");

#We set the minimum module size at 10:
minModuleSize = 10;

# Module identification using dynamic tree cut:
dynamicMods = cutreeDynamic(dendro = geneTree, distM = dissTOM,
                            minClusterSize = minModuleSize);
table(dynamicMods)
# dynamicMods
# 1  2
# 16 15

# Convert numeric lables into colors
moduleColors = labels2colors(dynamicMods)

#Define numbers of genes and samples
nGenes = ncol(exp);
nSamples = nrow(exp);
# Recalculate MEs with color labels
MEs0 = moduleEigengenes(exp, moduleColors)$eigengenes
MEs = orderMEs(MEs0)
```

At last, we compare the performance of overlappingCGM with WGCNA and iWGCNA as follows:

```
#overlappingCGM versus WGCNA
cor(cgm$patterns, wgcna$MEs)

#overlappingCGM versus iWGCNA
cor(cgm$patterns, MEs)
```

The comparative results are shown in TableS1 in Supplementary Table.

2.2 mouse metabolic syndrome

Firstly, we load the raw data including EXP and its clinical data (`exp` and `cli`)

```
#load raw file
exp = read.table("LiverFemale3600.csv", header = T, check.names = F,
sep=",")
cli = read.table("ClinicalTraits.csv", header = T, check.names = F,
sep=",", row.names = 1)
```

We remove missing genes (i.e., coded as 0) and duplicated genes due to insufficient information to retain them.

```
#remove missing gene names and duplicated genes in exp
exp = exp[which(exp$gene_symbol != "0"),]

dup = duplicated(exp$gene_symbol)
exp = exp[which(dup == FALSE),]

#turn exp1 into satisfactory format of DrGA
#DrGA requires data whose rows are samples and columns are genes.
exp = exp %>%
  dplyr::select(-c(substanceBXH, LocusLinkID, ProteomeID,
cytogeneticLoc,
            CHROMOSOME, StartPosition, EndPosition)) %>%
  tibble::remove_rownames() %>%
  tibble::column_to_rownames('gene_symbol') %>%
  drop_na() %>% t()
```

Since overlappingCGM is sensitive to outlier samples, we find potential ones by using hierarchical clustering method. Finally, we exclude a mouse named F2_221 (Figure S3).

```
#detect outliers
sampleTree = hclust(dist(exp), method = "average")
par(cex = 0.6);par(mar = c(0,4,2,0))
plot(sampleTree, main = "Sample clustering to detect outliers",
sub="", xlab="",
     cex.lab = 1.5,cex.axis = 1.5, cex.main = 2)

# Plot a line to show the cut
abline(h = 12.2, col = "red");
# Determine cluster under the line
clust = cutreeStatic(sampleTree, cutHeight = 12.2, minSize = 10)
table(clust)
# clust 1 contains the samples we want to keep.
keepSamples = (clust==1)
exp = exp[keepSamples, ]
```

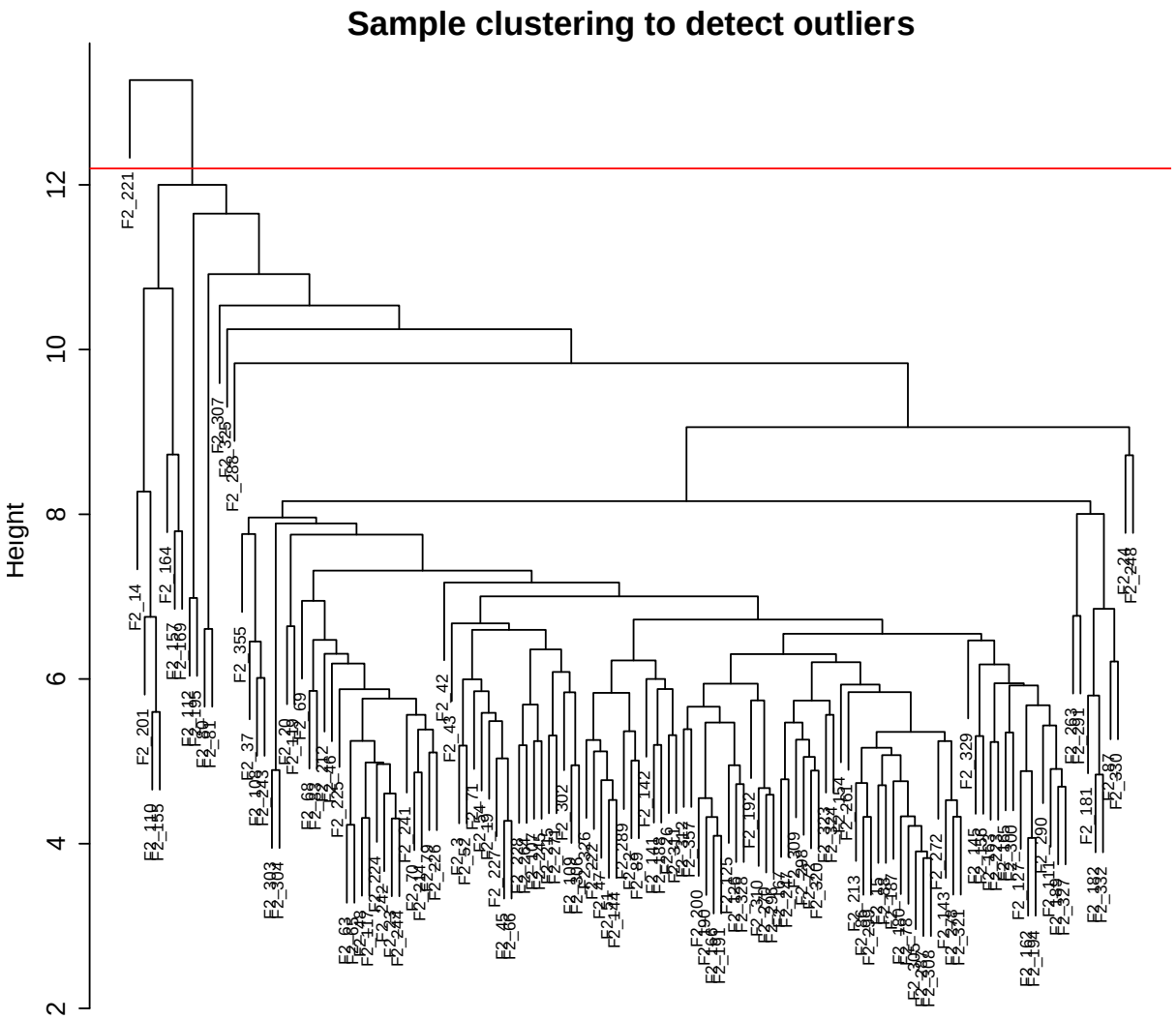

Figure S3. Detect and remove outliers.

We keep mice that share between `exp` and `cli` at their rows and in exactly the same order. We also only keep eight nescessary clinical features among 20 physiological features; i.e., body weight `weight_g`, body length `length_cm`, abdominal fat `ab_fat`, total fat `total_fat`, ulcerative colitis `UC`, free fatty acids `FFA`, glycemic index `Glucose`, two LDL and VLDL cholesterol levels `LDL_plus_VLDL` (Figure S4).

```
#match mouses that share between cli versus exp
cli = cli[cli$Mice %in% rownames(exp),]
cli = cli %>%
  remove_rownames() %>%
  tibble::column_to_rownames('Mice') %>%
  dplyr::select(-c(Number, sex, Mouse_ID, Strain, DOB, parents,
Western_Diet,
            Sac_Date, comments, Note))

#how the clinical traits relate to the sample dendrogram.
# Re-cluster samples
sampleTree2 = hclust(dist(exp), method = "average")
```

```
# Convert traits to a color representation: white means low, red
means high, grey means missing entry
traitColors = numbers2colors(cli, signed = FALSE);
# Plot the sample dendrogram and the colors underneath.
plotDendroAndColors(sampleTree2, traitColors,groupLabels =
names(cli), main = "Sample dendrogram and trait heatmap")
#white means a low value, red a high value, and grey a missing
entry.
```

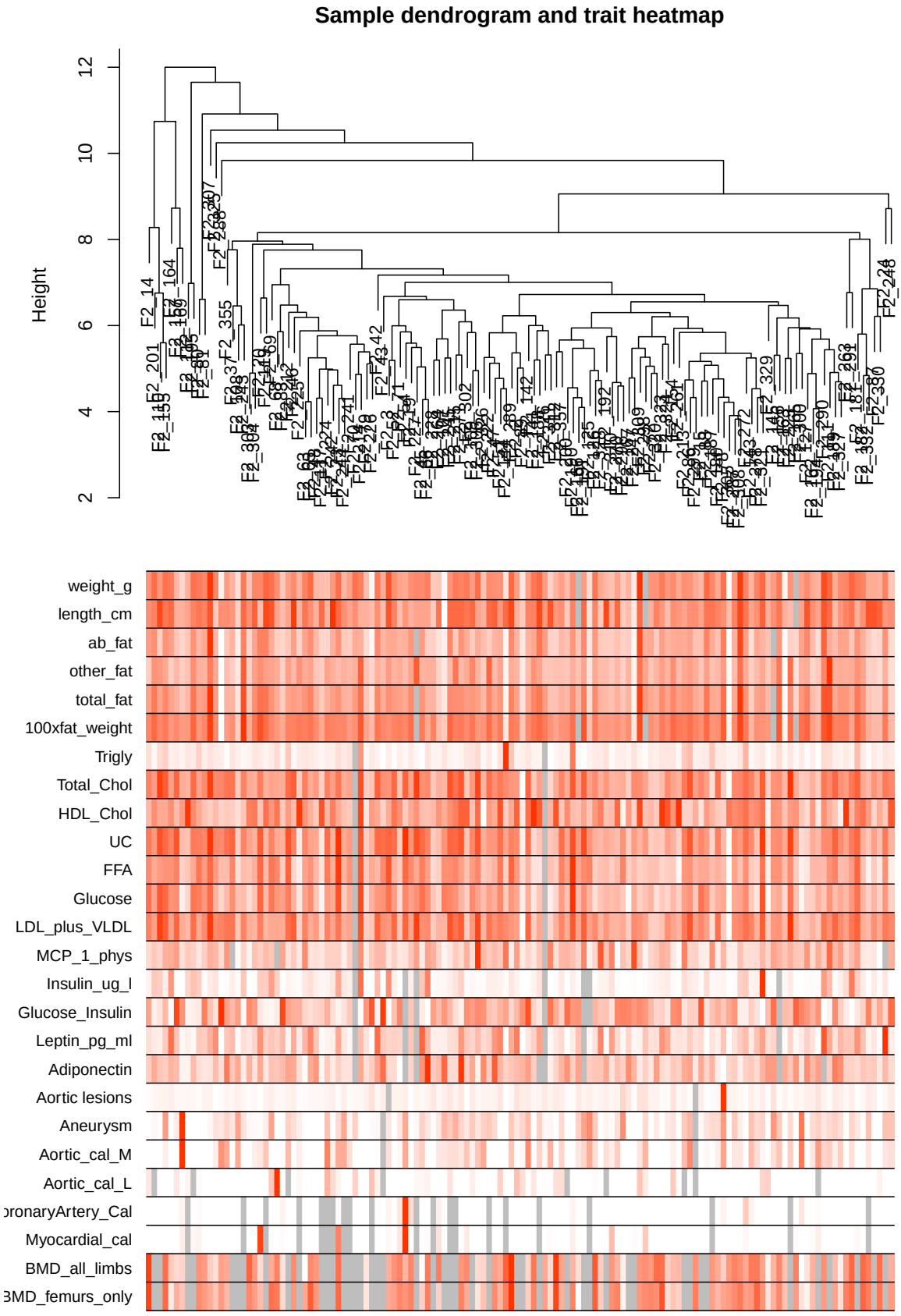

Figure S4. How the clinical features relate to the sample dendrogram. White means a low value, red a high value, and grey a missing entry

```
#Only keep several clinical features with red color
cli = cli %>%
  dplyr::select(weight_g, length_cm, ab_fat,
                total_fat, UC, FFA, Glucose,
                LDL_plus_VLDL)

#make sure that mice that share between exp and cli are included at
their rows and in exactly the same order
all(rownames(exp) == rownames(cli))
#[1] FALSE
exp = exp[rownames(cli), ]

#check dimension
dim(exp)
#[1] 134 2281
dim(cli)
#[1] 134  8
```

After the preprocessing step, we again apply WGCNA to the data with defaut parameters.

```
wgcna = blockwiseModules(exp, power = 6,
        TOMType = "unsigned", minModuleSize = 30,
        reassignThreshold = 0, mergeCutHeight = 0.25,
        numericLabels = TRUE, pamRespectsDendro = FALSE,
        verbose = 3)
```

Similarly, the followings are R codes for applying daBEST and overlappingCGM to the data. In this turn, daBEST indicates that 18 components are the best but cannot indicate which method should be selected. As suggested by [21], ICA-based decomposition method shows the best performance in identifying gene module plus we try reviewing if which post-processing step will not generate the grey module and realize that ICA-Zscore is the best fit for the data.

```
#overlappingCGM
daBEST(data = exp)
# >> overlappingCGM suggests choosing the optimal number of
components is: 18
# >> Both ICA and IPCA-FDR are appropriate for your case.
Please use another more stringent approach to make the best
decision.

cgm=overlappingCGM(data = exp, clinical = cli, ncomp = 18,
                           method = 'ICA-Zscore')
```

And for iWGCNA

```
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
enableWGCNAThreads() ### Allowing parallel execution

#we realize that the soft-thresholding of 5 is best fit for the data
softPower = 5;
adjacency = adjacency(exp, power = softPower,
                      type = "signed");

# Turn adjacency into topological overlap
```

```
TOM = TOMsimilarity(adjacency, TOMType = "signed");
dissTOM = 1-TOM

# Hierichical clustering
#Seek the optimal agglomeration method
#methods to assess
m <- c( "average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")
# function to compute agglomerative coefficient
set.seed(25896)
ac <- function(x) {
  agnes(exp, method = x)$ac
}

map_dbl(m, ac) # Agglomerative coefficient of each agglomeration
method
# average    single  complete      ward
# 0.4136737 0.3510251 0.4815638 0.5563910

#assign gene names from adjacency to dissTOM
rownames(dissTOM) = rownames(adjacency)
colnames(dissTOM) = colnames(adjacency)
# Call the hierarchical clustering function
geneTree = hclust(as.dist(dissTOM), method = "ward.D2");

#We set the minimum module size at 10:
minModuleSize = 10;

# Module identification using dynamic tree cut:
dynamicMods = cutreeDynamic(dendro = geneTree, distM = dissTOM,
                            minClusterSize = minModuleSize);
table(dynamicMods)
# dynamicMods
# 1   2   3   4   5   6   7   8   9  10  11  12
# 341 311 272 230 216 211 188 168 112  84  80  68

# Convert numeric lables into colors
moduleColors = labels2colors(dynamicMods)

#Define numbers of genes and samples
nGenes = ncol(exp);
nSamples = nrow(exp);
# Recalculate MEs with color labels
MEs0 = moduleEigengenes(exp, moduleColors)$eigengenes
MEs = orderMEs(MEs0)
```

At last, we compare the performance of overlappingCGM with WGCNA and iWGCNA as follows:

```
#overlappingCGM versus WGCNA
cor(cgm$patterns, wgcna$MEs)

#overlappingCGM versus iWGCNA
cor(cgm$patterns, MEs)
```

The comparative results are shown in TableS3 in Supplementary Table.

3. Understanding the tool and its results

3.1 daBEST function

```
daBEST(data = NULL, P=1000, standardize = T, verbose = T)
```

where the argument `data` is gene expression matrix whose rows are samples and columns are genes. `P` is the number of permutations which is set to 1000 as default like explained above. If your expression matrix is not standardized, please feed `T/TRUE` to the argument `standardize` to let overlappingCGM do this.

When you correctly input your data and the parameters to daBEST, it will automatically output in the R console result as follows (Figure S5).

```
>> overlappingCGM suggests choosing the optimal number of components is: 9
>> overlappingCGM also suggests using ICA for your case.
Done in 1.637788550059 mins.
```

Figure S5. daBEST runs with the input of the 31 driver genes in human breast cancer and `P = 1000`. The results are printed in the R console result.

3.2 overlappingCGM

```
cgm = overlappingCGM(data = NULL, clinical = NULL, ncomp = NULL,
standardize = T, method = c("ICA-FDR", "ICA-Zscore", "IPCA-FDR"))
```

where the argument `data` is gene expression matrix whose rows are samples and columns are genes. `clinical` is clinical data including clinical features of choice. If your expression matrix is not standardized, please feed `T/TRUE` to the argument `standardize` to let overlappingCGM do this. The argument `ncomp` is user-decided number of components that can be refer to the suggestion of daBEST. The argument `method` shows options are in a format as "method - post-processing", in which the methods may be ICA or IPCA and the post-processing procedures may be FDR or Z-score.

When you correctly input your data and the parameters to overlappingCGM, it will automatically output in the R console result as follows (Figure S6 and Figure S7).

```
> cgm=overlappingCGM(data = exp, clinical = clinicalEXP, ncomp = 9,
+                    method = 'ICA-Zscore')
>> overlappingCGM detected 6 components whose kurtosis <= 3
>> overlappingCGM excluded them in order to serve to find out modules of genes, leaving 3 components for
 further analyses
>> As shown in Z-score table related to expression levels of each gene within each module, please choose
 a probability threshold to let overlappingCGM be able to assign genes to modules specifically.
>> *NOTE that your selection will affect the minimum number of genes within a certain module.
>> The probability threshold you want to choose is?0.5
>> The exact number of genes assigned to each module is:
    turquoise blue brown
num        25   20    22

 - Starting to detect hub-genes within each discovered module...
 >>>> Top 10 hub genes identified in the turquoise module are: KMT2C BAP1 PTEN NF1 RUNX1  ZFP36L1 CDKN1B
 BRCA2 MAP3K1 PIK3CA

 >>>> Top 10 hub genes identified in the blue module are: CDH1 PIK3R1 GATA3 CDKN2A TBX3  SMAD4 KRAS RB1
 MEN1 RUNX1

 >>>> Top 10 hub genes identified in the brown module are: KRAS GPS2 SF3B1 AGTR2 RB1  NCOR1 SMAD4 ERBB3
 FOXO3 NF1
```

Figure S6. overlappingCGM runs with the input of the 31 driver genes in human breast cancer. The results are printed in the R console result. Firstly, overlappingCGM seeks and removes signatures whose kurtosis value < 3. Then, it asks users to choose the threshold to distribute genes to modules. Due to the small number of genes, we choose the threshold of 0.5 sigma on either side from the zero mean. Finally, overlappingCGM prints all the results in the R console result.

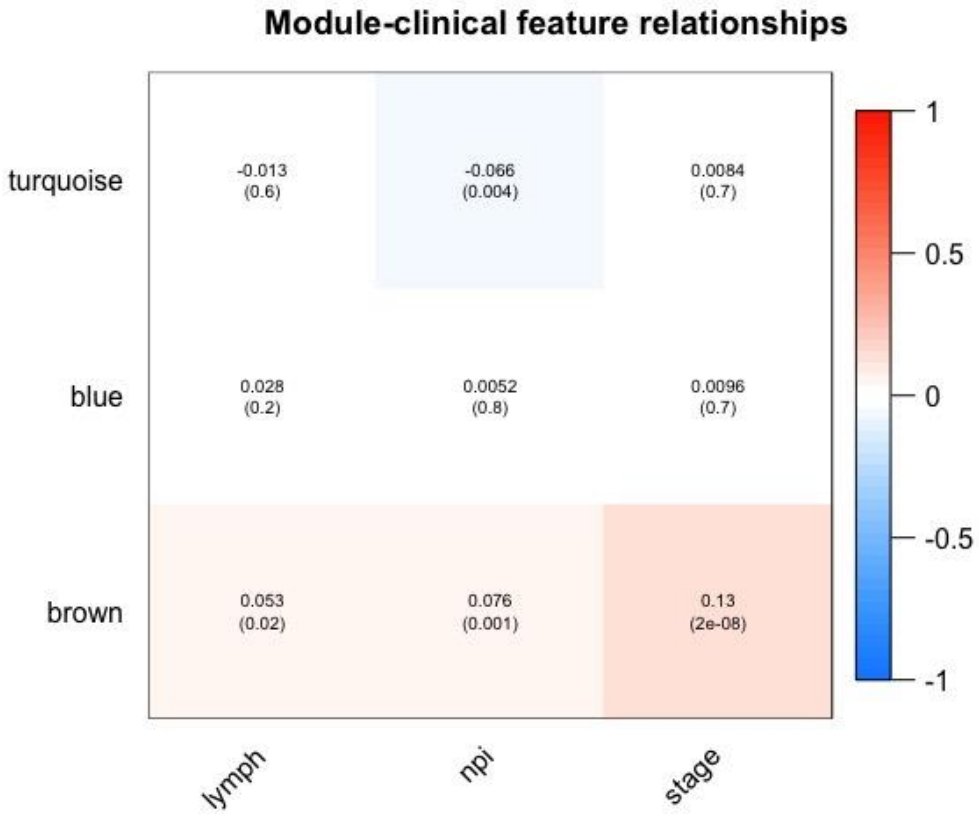

Figure S7. Associations between the three patterns obtained by overlappingCGM and the clinical features of interest calculated by the Pearson's correlation. Abbreviation: the number of lymph node, lymph; Nottingham prognostic index, npi; tumor stages, stage.

Note that the users can specifically see which genes are distributed to which color-coded module by the command (Figure S8)

```
cgm$signatures
```

	turquoise	blue	brown
NCOR1	1	0	1
ZFP36L1	1	1	0
SMAD4	1	1	1
CDKN1B	1	0	1
CDH1	0	1	1
PIK3R1	1	1	0
BRCA2	1	1	0
KMT2C	1	1	0
KRAS	0	1	1
CBFB	1	1	1
MAP2K4	1	0	1
BAP1	1	1	0
ERBB2	1	0	1
TP53	1	0	1

Figure S8. Results of which genes (rows) are distributed to which color-coded module (columns). 1 = module-assigned, 0 is otherwise.

4. Supplementary References

1. Comon, P., *Independent component analysis, a new concept?* Signal processing, 1994. **36**(3): p. 287-314.
2. Hyvärinen, A. and E. Oja, *Independent component analysis: algorithms and applications.* Neural networks, 2000. **13**(4-5): p. 411-430.
3. Liebermeister, W., *Linear modes of gene expression determined by independent component analysis.* Bioinformatics, 2002. **18**(1): p. 51-60.
4. Yao, F., J. Coquery, and K.-A. Lê Cao, *Independent Principal Component Analysis for biologically meaningful dimension reduction of large biological data sets.* BMC Bioinformatics, 2012. **13**(1): p. 24.
5. Jolliffe, I.T. and J. Cadima, *Principal component analysis: a review and recent developments.* Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2016. **374**(2065): p. 20150202.
6. Lee, S.-I. and S. Batzoglou, *Application of independent component analysis to microarrays.* Genome biology, 2003. **4**(11): p. 1-21.
7. Purdom, E. and S.P. Holmes, *Error distribution for gene expression data.* Statistical applications in genetics and molecular biology, 2005. **4**(1).
8. Huang, D.-S. and C.-H. Zheng, *Independent component analysis-based penalized discriminant method for tumor classification using gene expression data.* Bioinformatics, 2006. **22**(15): p. 1855-1862.
9. Engreitz, J.M., et al., *Independent component analysis: mining microarray data for fundamental human gene expression modules.* Journal of biomedical informatics, 2010. **43**(6): p. 932-944.
10. Scholz, M., et al., *Metabolite fingerprinting: detecting biological features by independent component analysis.* Bioinformatics, 2004. **20**(15): p. 2447-2454.
11. Yeung, K.Y. and W.L. Ruzzo, *Principal component analysis for clustering gene expression data.* Bioinformatics, 2001. **17**(9): p. 763-774.
12. Horn, J.L., *A rationale and test for the number of factors in factor analysis.* Psychometrika, 1965. **30**(2): p. 179-185.
13. Buja, A. and N. Eyuboglu, *Remarks on parallel analysis. Multivariate Siniša Subotic 222 Glorfeld, LW (1995). An improvement on Horn's parallel analysis methodology for selecting the correct number of factors to retain.* Educational and Psychological, 1992.
14. Abdi, H. and L.J. Williams, *Principal component analysis.* Wiley interdisciplinary reviews: computational statistics, 2010. **2**(4): p. 433-459.
15. Mariëlle, L., *Nonparametric Inference in Nonlinear Principal Components Analysis: Exploration and Beyond*. 2007, **Leiden University**.
16. Landgrebe, J., W. Wurst, and G. Welzl, *Permutation-validated principal components analysis of microarray data.* Genome Biology, 2002. **3**(4): p. 1-11.
17. Nguyen, Q.-H. and D.-H. Le, *Improving existing analysis pipeline to identify and analyze cancer driver genes using multi-omics data.* Scientific Reports, 2020. **10**(1): p. 20521.
18. Cerami, E., et al., *The cBio Cancer Genomics Portal: An Open Platform for Exploring Multidimensional Cancer Genomics Data.* Cancer Discovery, 2012. **2**(5): p. 401-404.
19. Gao, J., et al., *Integrative analysis of complex cancer genomics and clinical profiles using the cBioPortal.* Sci Signal, 2013. **6**(269): p. pl1.
20. Ghazalpour, A., et al., *Integrating Genetic and Network Analysis to Characterize Genes Related to Mouse Weight.* PLOS Genetics, 2006. **2**(8): p. e130.

21. Saelens, W., R. Cannoodt, and Y. Saeys, *A comprehensive evaluation of module detection methods for gene expression data.* Nature Communications, 2018. **9**(1): p. 1090.