DEComplexDisease: A tool for differential expression analysis and exploratory investigation to complex diseases using big data

Guofeng Meng

2017-11-06

Contents

0.1 Introduction

DEComplexDisease is designed to find the differential expressed genes (DEGs) for complex diseases, which are characterized by the heterogeneous expression profiles. Different from the established DEG analysis tools, it does not assume the patients of complex diseases to share the common DEGs. By applying a bi-clustering algorithm, DEComplexDisease finds the DEGs shared by many patients. In this way, DEComplexDisease describes the DEGs of complex disease in a novel syntax, e.g. a gene list composed of 200 genes are differentially expressed in 30% percent of studied complex disease patients.

Applying the DEComplexDisease analysis results, users are possible to find the patients affected by the same mechanism based on the shared signatures. This can be achieved by modelling the breakpoints of the bi-clustering analysis and enrichment analysis to the feature patients or genes, e.g. the patient carrying the same mutations. DEComplexDisease also supplies visualization tools for nearly all the output.

0.2 Package installation

```
source("https://bioconductor.org/biocLite.R")
biocLite("DEComplexDisease")
```

0.3 Clear the session

DEComplexDisease needs huge computing powers to do the whole analysis for the reasons that (1) it is designed for big data, which usually have hundreds of patients; (2) it implements a bi-clustering algorithm in many steps. Therefore, it is highly recommended to use multi-core parallel computing.

In `DEComplexDisease` package, we use `mclapply` tool of `parallel` package to do the multiple process computing. Users just need set "cores = n" in the `DEComplexDisease` functions to make use of multi-core hardware, where is "n" is the core number.

Before the analysis, it is better start a new session or run

```
rm(list=ls())
```

to clear the current session. Otherwise, the whole session will be copy to each process, which may consume Huge ROM memory. Once the memory is ran out, `DEComplexDisease` may report wrong results or other problems.

In this manual, all the examples assume users to have a 4-core computer by setting `cores`=4.

0.4 Prepare input data

`DEComplexDisease` takes two mandatory inputs: an expression matrix (exp) and a sample annotation vector (cl). The expression matrix can be the RNA-seq counts data or normalized microarray expression data. The samples annotation vector (cl) is a vector to indicate the disease states of samples. Mandatorily, `cl` has the same length and order with the columns of `exp`. `cl` only has two possible values: 1 and 0. 1 indicates the corresponding samples to be patients and 0 is control or normal samples. The optional input is the clinical annotation data, which can be use for visualization in Plot() function. The clinical annotation should have the `row.names` matched by the column names of `exp`.

In this manual, we use the RNA-seq counts data for 1217 breast cancer samples collected in TCGA project ("https://cancergenome.nih.gov/") as an example to illustrate the basic analysis steps of DEComplexDisease. The full dataset is available at https://sourceforge.net/projects/decd/:

```
>download.file("https://superb-sea2.dl.sourceforge.net/project/decd/decd_data.rda",
              "decd_data.rda");
>load("decd_data.rda")
>exp.brca[1:5,1:5]
         TCGA.D8.A27K.01A TCGA.BH.A0HU.01A TCGA.BH.A5IZ.01A TCGA.AR.A5QM.01A TCGA.BH.A1EV.01A
TSPAN6               6801             2139             3274             1268             1649
TNMD                    5               33                2               44                0
DPM1                 1772             2582             2895             1375             2073
SCYL3                4659             2115             1019             1490             2321
C1orf112             1203              884             1591              478              748
>head(cl.brca, 4)
TCGA.D8.A27K.01A TCGA.BH.A0HU.01A TCGA.BH.A5IZ.01A TCGA.AR.A5QM.01A
               1                1                1                1
```

0.5 The reference expression profiles

The performance of DEComplexDisease is highly dependent on the quality of reference samples. Therefore, it is recommended to evaluate the quality of expression data for control samples. Based on `cl`, the control samples are collected and evaluated with hierarchical clustering.

```
hc=hclust(dist(t(exp.brca[, cl==0])))
```

```
plot(hc)
rect.hclust(hc, k= 4, border = 'red')
```

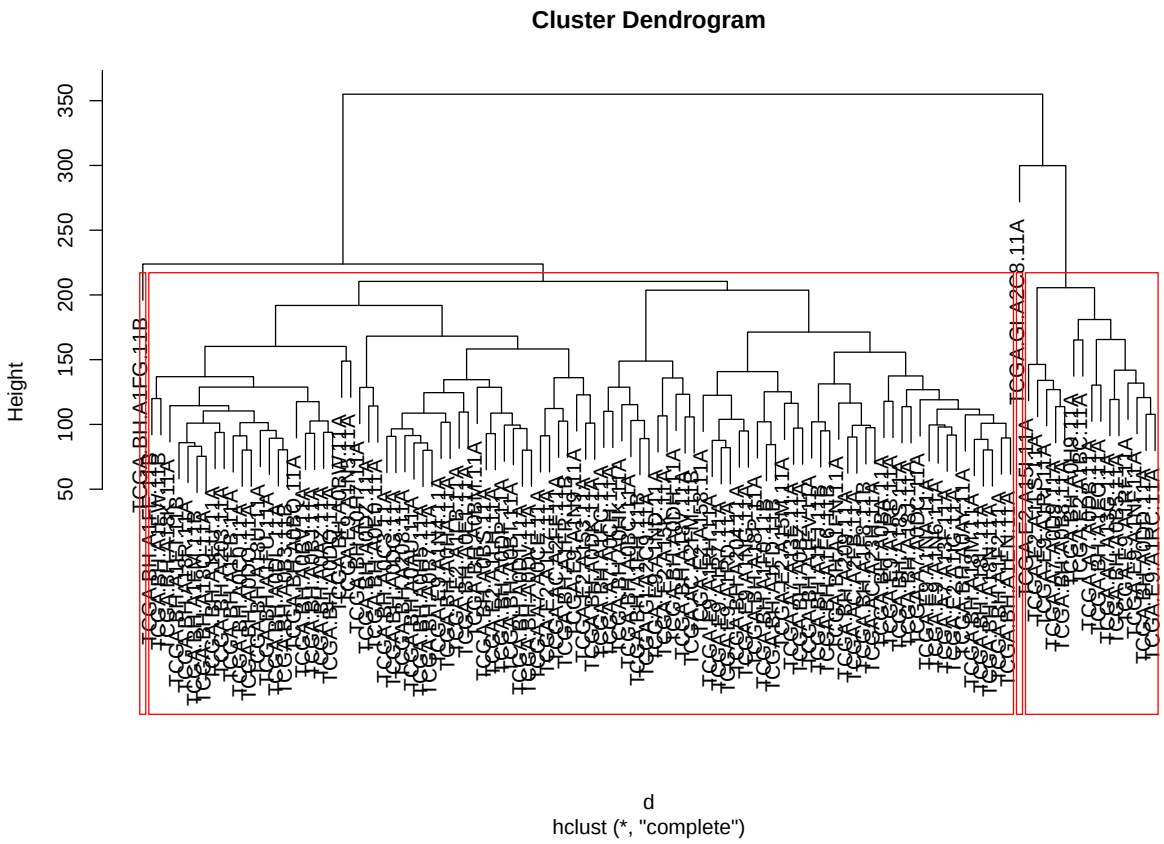


Figure 1: hierarchical clustering to normal samples

Based the hierarchical clustering results, the outliers are removed from reference building:

```
group=cutree(hc, k=4)
exp=exp.brca[, !colnames(exp.brca) %in% names(group[ group != 2])]
cl=cl.brca[ colnames(exp) ]
```

0.6 Transform expression matrix into binary DEG matrix

DEComplexDisease applies `bi.deg` function to estimate the distribution parameters that describes the gene expression profiles of control samples. The normal samples are collected based on `cl`. `bi.deg` has three methods to estimate the reference expression profiles of control samples: "edger", "deseq2" or "normalized". "edger" or "deseq2" is used for RNA-seq counts data by implementing the algorithms developed by edgeR or DESeq2. The dispersion (disp) and mean (mu) are estimated for each gene. The count number of $x_{i,j}$ is tested by

```
p=pnbinom(x, size=1/disp, mu=mu, lower.tail = F)
```

"normalized" is used for normalized RNA-seq or microarray data. The mean (mu) and standard deviation (sd) is estimated for each gene. The z-score and p-value is calculated by

```
  z=(x-mu)/sd
  p=pnorm(z,lower.tail=F)
```

Using the p-value cutoff defined by the users, `bi.deg` assigns 1 or -1 to indicate the up- or down-regulated genes, where 1 is the up-regulated genes and -1 is the down-regulated genes. The other genes are assigned with 0. In this example,

```
deg=bi.deg(exp, cl, method="edger", cutoff=0.05, cores=4)
```

`bi.deg` returns a `deg` object, which is a matrix of 1,0 and -1. Users can use Plot() to view the analysis results.

```
Plot(deg, ann=ann.er, show.genes=c("ESR1","FOXA1","GATA3","FOXC1"))
```

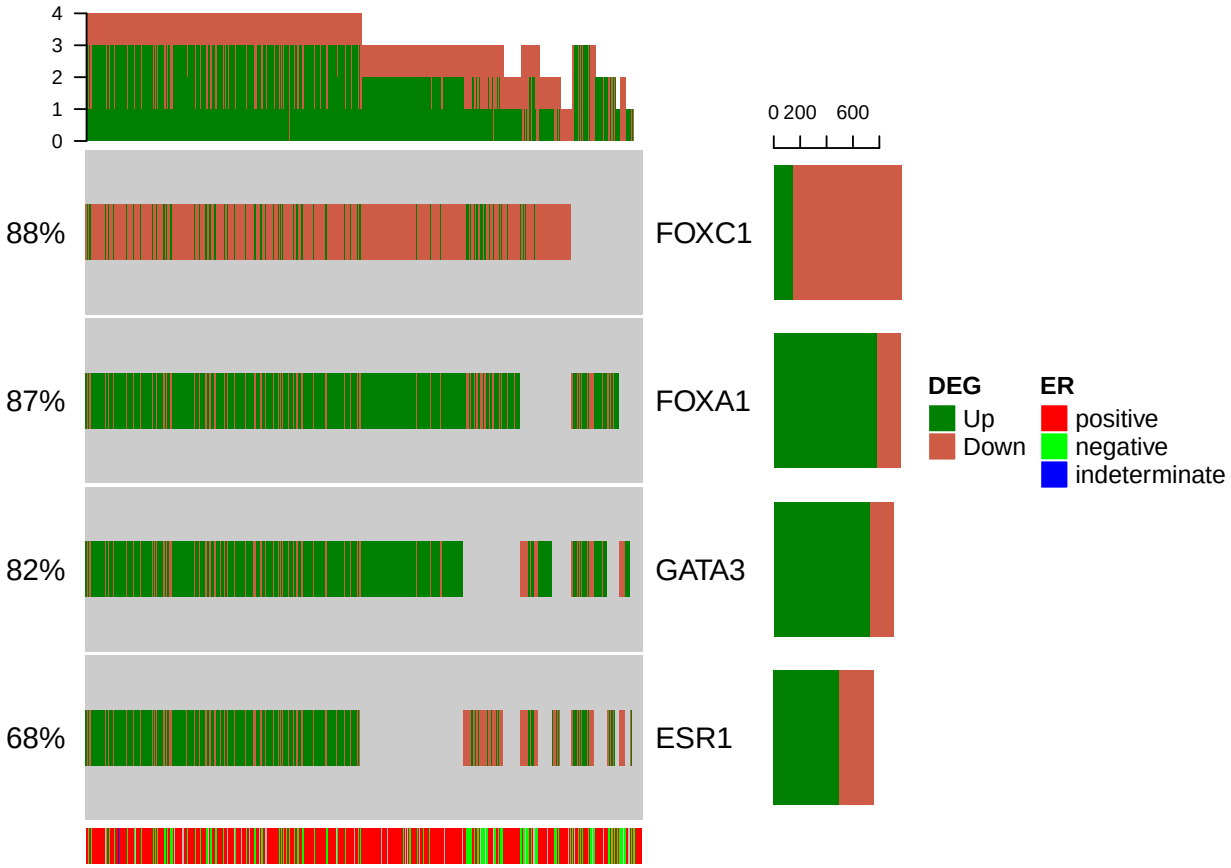


Figure 2: Plot for DEGs

Here, users can use `show.genes` to display the selected genes. Otherwise, the `max.n` genes with maximum enrichment will be showed. In this example, we also use a `ann` to show the annotation for ER status of the patients.

0.7 Cross-validated patient-specific DEGs

The DEGs from `bi.deg` have noises, e.g. the DEGs not associated with disease, when one sample is tested again references. The idea behind his analysis is that the disease associated DEGs are supposed to be observed in other patients. In another words, when there are enough samples, the patients can be used to cross-validation of high-confident DEGs. `deg.specific` implements a bi-clustering analysis algorithm to the binary DEG matrix to find the high-confident disease associated DEGs.

`deg.specific` is used to find the cross-validated DEGs. Users only need to choose the proper minimum gene and minimum patient number by setting `min.genes` and `min.patients`. Please note that `min.patients`

includes the seed patient itself.

```
res.deg=deg.specific(deg, min.genes=50, min.patients=5, cores=4)
```

If users only want to see the results some selected patients, use `test.patients` to reduce the computation time.

```
res.deg.test=deg.specific(deg, test.patients=brca1.mutated.patients, min.genes=50,
                          min.patients=8, cores=4)
```

`deg.specific` returns a `deg.specific` object, which is a list comprising of the cross-validated DEGs and the support neighbors. If `test.patients` is set, a `deg.specific.test` object is returned. Users can use Plot() to visualize the `deg.specific` and `deg.specific.test` object. In Figure 2, we shows the validated DEGs in all used breast cancer patients.

```
Plot(res.deg, ann=ann.er, show.genes=c("ESR1","FOXA1","GATA3","FOXC1"))
```

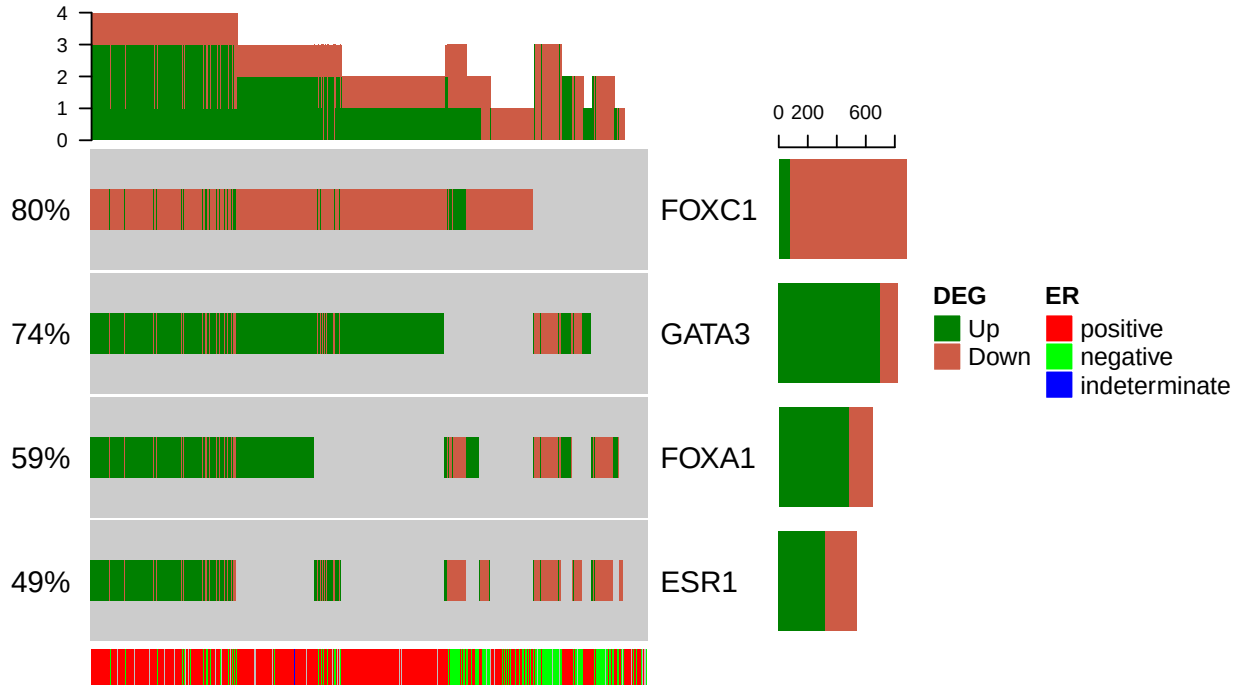


Figure 3: Plot for cross-validated DEGs

For `res.deg.test`, it can also be plotted :

```
Plot(res.deg.test, ann=ann.er, show.genes=c("ESR1","FOXA1","GATA3","FOXC1"))
```

0.8 Patient-seeded modules

In this work, the DEG modules refer to the DEG list shared by many patients. DEG modules are supposed to be the signatures of patients shared the similar causal mechanism. `seed.module` and `cluster.module` are designed to find such modules.

Like `deg.specific`, `seed.module` carries out a bi-clustering analysis to the output of `bi.deg` using each patient as a seed. The difference is that this function has more complex setting and steps to predict DEGs modules shared by many patients.

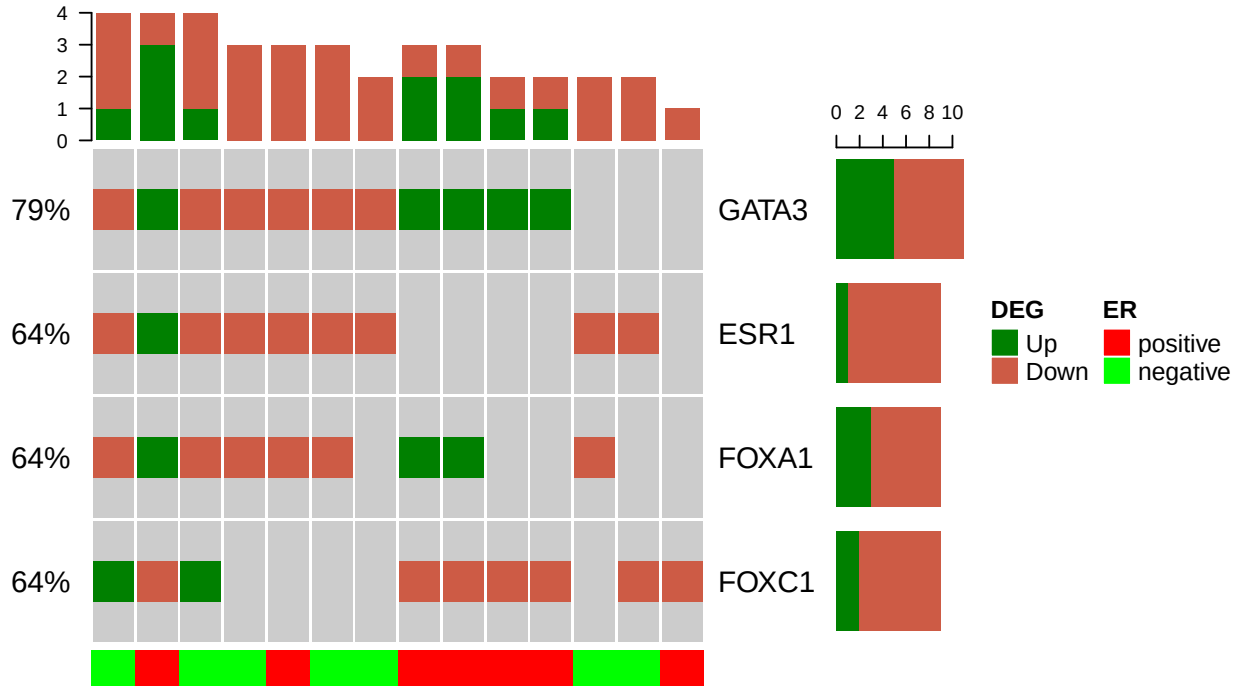


Figure 4: Plot for cross-validated marker genes

```
seed.mod1 = seed.module(deg, res.deg=res.specific, min.genes=100, min.patients=50,
                        overlap=0.85, cores=4)
```

or

```
seed.mod2 = seed.module(deg, test.patients=brca1.mutated.patients, min.genes=100,
                        min.patients=20, overlap=0.85,  cores=4)
```

No matter whatever the parameter setting is, `seed.module` will firstly try to find a modules shared by all the patients, where the final patient number may be less than the `min.patients` and gene number may be less than `min.genes`. If such module exists, it will named as `M0` and the module genes of `M0` will be removed from further discovery to module genes. That is to say, module genes of `M0` will not be included in other modules.

The bi-clustering analysis is started by a DEG seed, composing of the DEGs of a patient. If `res.deg` is set, only the patients with cross-validated DEGs will be used as seeds and the seed will initialized with the cross-validated DEGs. Otherwise, all the patients will be used and all the DEGs are used as seed. The DEGs of patients will be gradually removed to check if the left seed are observed in `min.patients` when keeping the similarity is not less than `overlap`. `seed.module` will record the track of gene-patient number in the bi-clustering analysis, which is stored in a `curve` key of output for each patient. If `test.patients` is set, only the patients listed in `test.patients` are used as seed for bi-clustering analysis.

`seed.module` will record the bi-clustering results at three scenarios: * `max.genes` records the patient and genes information when the seed is observed in `min.patient`. * `max.patients` stores the patient and gene information when `min.genes` are observed, which is also the terminated point of bi-clustering analysis. * `model` stores the gene/patient information when the gene-patients number `curve` has satisfied some modelling criteria defined by 'model.method', which may indicate the inclusion/exclusion of molecular mechanism.

In current version, 'model.method' has four possible values: "slope.clustering", "max.square", "min.slope" and "min.similarity", which indicate the different four different modelling methods: * `slope.clustering` has maximum slope changes, which may indicate the inclusion/exclusion of molecular mechanism; * `max.square` is the gene-patients number that has the maximum product; * `min.slope` has the minimum slope in gene-patient

number curve; * `min.similarity` is the point with minimum similarity scores

`seed.module` will return a 'seed.module' object. This is a list. It has one key with prefix of "decd": * "decd.input", the input information, including binary DEG matrix, test.patients and other parameter setting.

It may have one key of "M0": * "M0", a modules shared by all the patients. In many cases, M0 is NULL when M0 is not predicted.

Other keys are patient IDs, which are the modules predicted with DEG seed of the patient. Each one have several keys:

- "curve", the patient-gene number during bi-clustering analysis;
- "max.genes", the patient and genes when 'min.patients' is observed in bi-clustering analysis;
- "max.patients", the patient and genes when 'min.genes' is reached in bi-clustering analysis";
- "model", the patient and genes at the breakpoint of the `curve`;
- "genes.removed", the ordered genes that are removed from module during bi-clustering analysis;
- "patients.added", the ordered patients that are added to module during bi-clustering analysis

The 'seed.module' object can visualized with Plot().

```
Plot(seed.mod1, ann=er.ann, type="model", max.n=5)
```

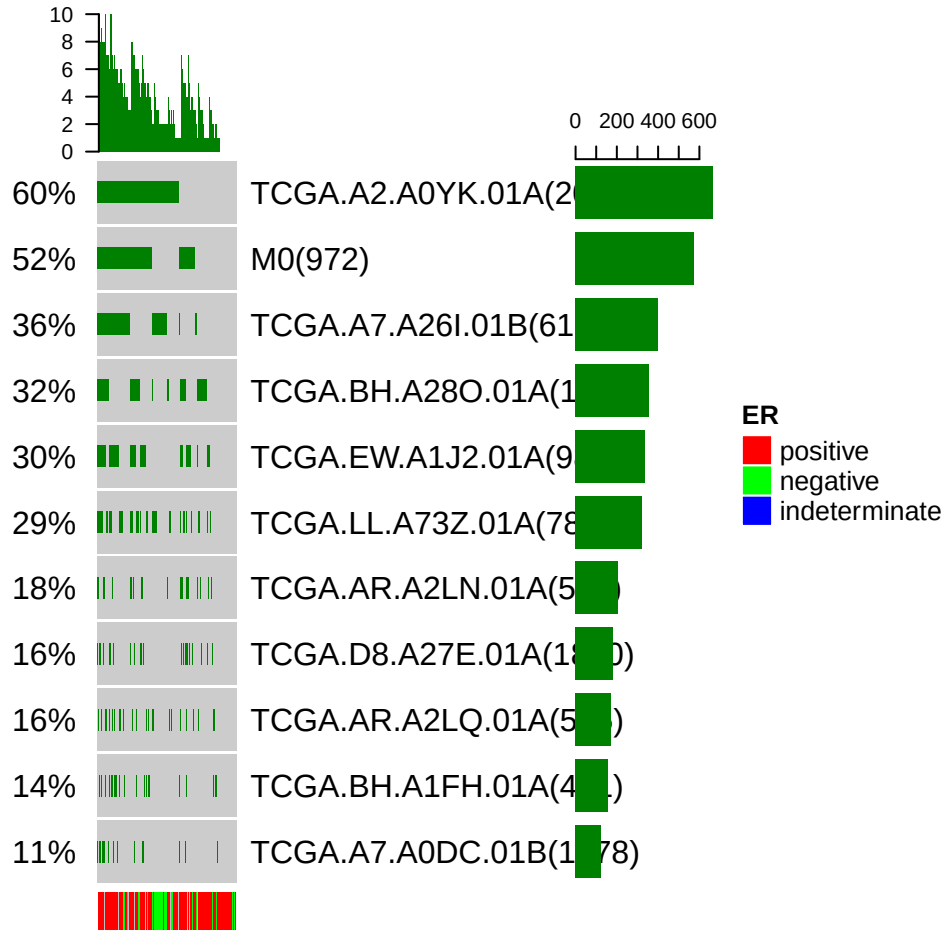


Figure 5: Plot for clustered module

0.9 Clustering patient-seeded modules

For the big data, there are usually many patient samples and DEComplexDisease may predict too many patient-seeded modules. `cluster.module` is used to cluster the patient-seeded modules by a modified k-means based on their patient and gene signature similarity. The patients within the same cluster are ranked based on their connecting degrees so that to find the representative patient(s). if `vote.seed` is set as false, the bi-clustering analysis results of the representative patient will be used as the final results of the module. Otherwise, a generic seed will be generated by a voting method and the final results is predicted by bi-clustering analysis using the new seed. All the modules will marked as "M1", "M2", "M3"...

```
cluster.mod1 <-  seed.module(seed.mod1, cores=4)
```

or

```
cluster.mod2 <- seed.module(seed.mod1, vote.seed=T, cores=4)
```

`cluster.module` returns a `cluster.module` object.

```
>sort(names(cluster.mod1), decreasing=T)
  [1] "M99"             "M98"             "M97"             "M96"             "M95"
  [6] "M94"             "M93"             "M92"             "M91"             "M90"
 [11] "M9"              "M89"             "M88"             "M87"             "M86"
 [16] "M85"             "M84"             "M83"             "M82"             "M81"
 [21] "M80"             "M8"              "M79"             "M78"             "M77"
 [26] "M76"             "M75"             "M74"             "M73"             "M72"
 [31] "M71"             "M70"             "M7"              "M69"             "M68"
 [36] "M67"             "M66"             "M65"             "M64"             "M63"
 [41] "M62"             "M61"             "M60"             "M6"              "M59"
 [46] "M58"             "M57"             "M56"             "M55"             "M54"
 [51] "M53"             "M52"             "M51"             "M50"             "M5"
 [56] "M49"             "M48"             "M47"             "M46"             "M45"
 [61] "M44"             "M43"             "M42"             "M41"             "M40"
 [66] "M4"              "M39"             "M38"             "M37"             "M36"
 [71] "M35"             "M34"             "M33"             "M32"             "M31"
 [76] "M30"             "M3"              "M29"             "M28"             "M27"
 [81] "M26"             "M25"             "M24"             "M23"             "M22"
 [86] "M21"             "M20"             "M2"              "M19"             "M18"
 [91] "M17"             "M16"             "M15"             "M14"             "M130"
 [96] "M13"             "M129"            "M128"            "M127"            "M126"
[101] "M125"            "M124"            "M123"            "M122"            "M121"
[106] "M120"            "M12"             "M119"            "M118"            "M117"
[111] "M116"            "M115"            "M114"            "M113"            "M112"
[116] "M111"            "M110"            "M11"             "M109"            "M108"
[121] "M107"            "M106"            "M105"            "M104"            "M103"
[126] "M102"            "M101"            "M100"            "M10"             "M1"
[131] "M0"              "decd.input"      "decd.clustering"))
>names(cluster.mod1[["decd.input"]])
[1] "genes"         "patients"      "overlap"       "deg"           "test.patients" "min.genes"
[7] "min.patients"  "vote.seed"     "model.method"
>names(cluster.mod1[["decd.clustering"]])
[1] "group"     "represent"
>names(cluster.mod1[["M1"]])
[1] "max.genes"      "max.patients"   "genes.removed"  "patients.added" "curve"
[6] "seed"           "model"
```

A `cluster.module` has one key with prefix of "decd":

- "decd.input", the input information, including binary DEG matrix, used.genes, used.patients and other parameter setting.

Other keys has a prefix of "M", which indicates modules. Each module have several keys:

- "curve", the patient-gene number during bi-clustering analysis;
- "max.genes", the patient and genes when `min.patients` is observed in bi-clustering analysis;
- "max.patients", the patient and genes when `min.genes` is reached in bi-clustering analysis";
- "model", the patient and genes at the breakpoint of the `curve`;
- "genes.removed", the ordered genes that are removed from module during bi-clustering analysis;
- "patients.added", the ordered patients that are added to module during bi-clustering analysis

The `deg.module` can be visualized by Plot().

```
Plot(cluster.module, ann=er.ann, type="model", max.n=5)
```

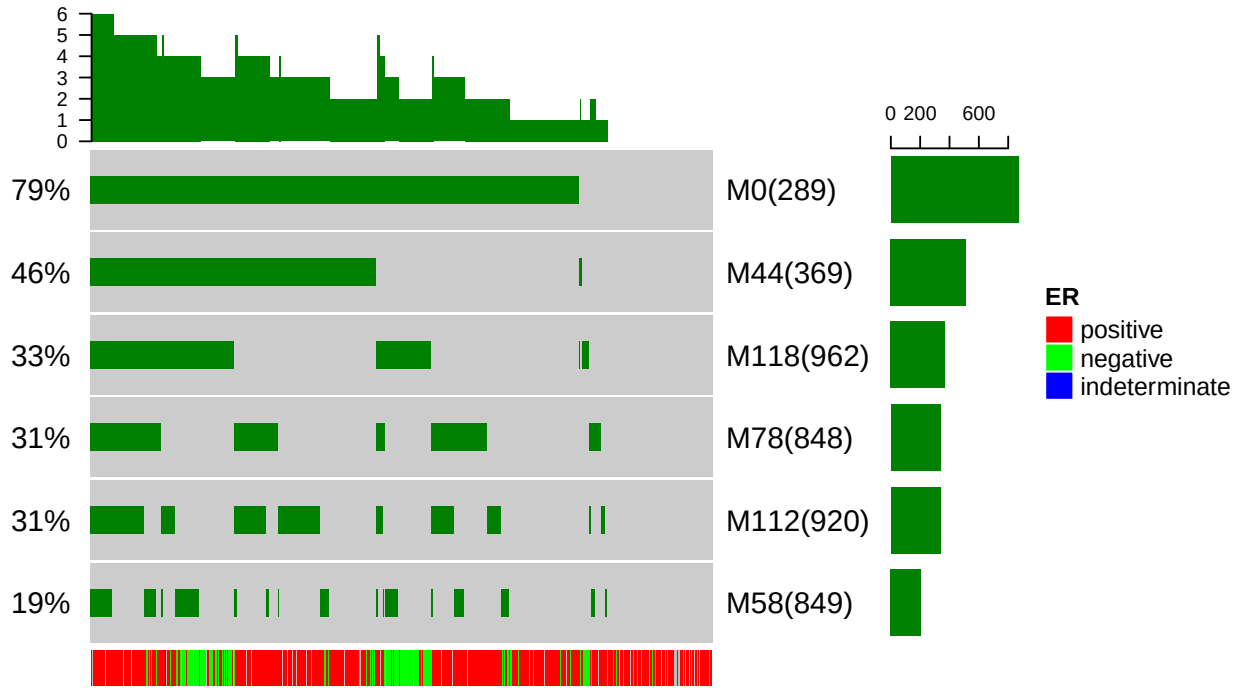
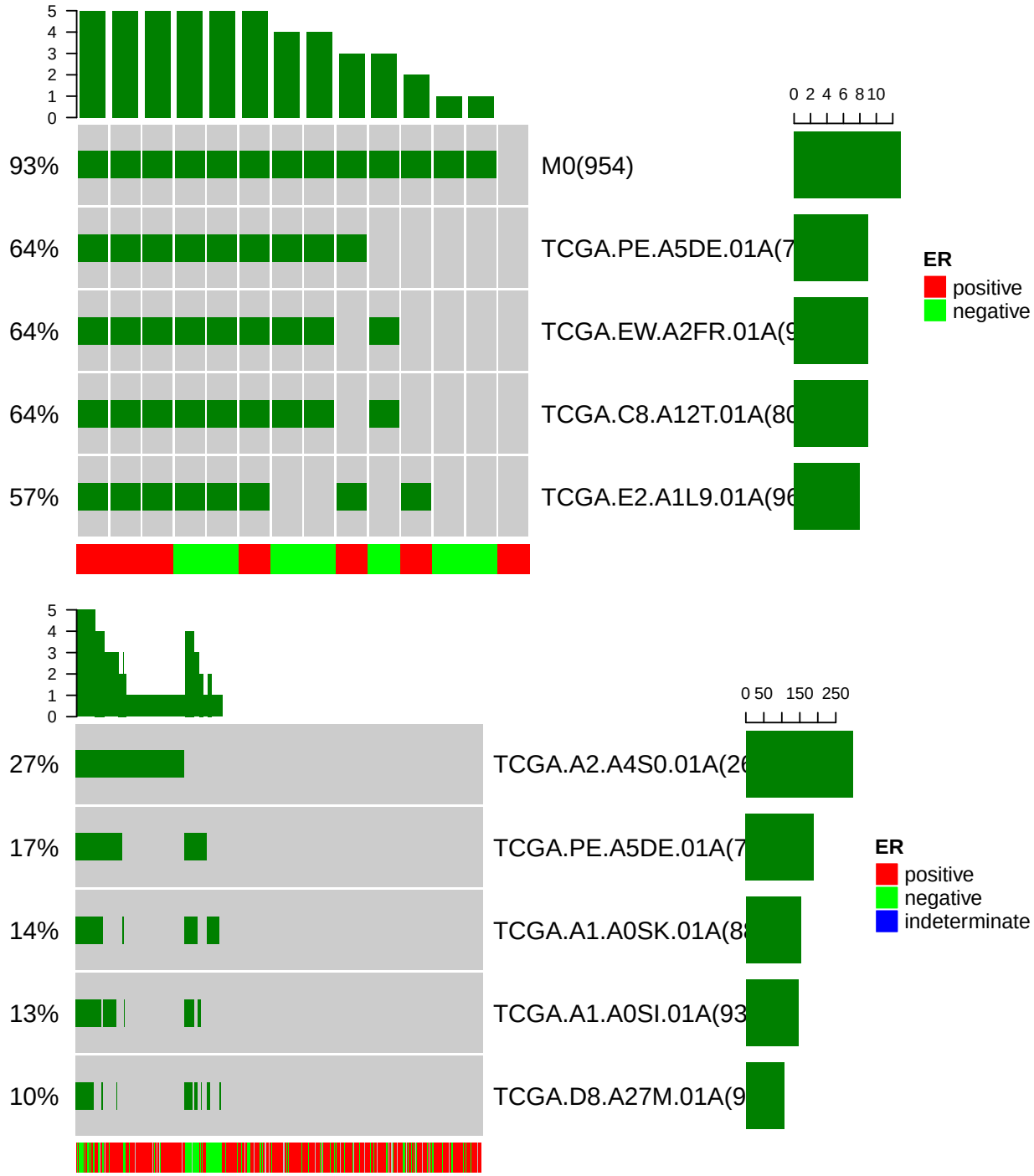


Figure 6: Plot for clustered module

In this examples, 5 modules are showed for their affiliation with the patients used in module discovery. It is possible to display these modules in independent samples by setting `deg` option. For examples,

```
brca1.module <- seed.module(deg[, brca1.mutated.patients], min.row=100, min.col=7,
                            cutoff=0.8, cores=4)
Plot(brca1.module, ann=ann.er, max.n=5, type="model")
Plot(brca1.module, ann=ann.er, deg=deg, max.n=5, type="model")
```



`deg` option can be use to display the association of the modules of disease A in disease B.

0.10 The patient and gene overlaps

The DEG modules may have partial overlaps for either genes or patients. Use "module.overlap" to check the gene and patient overlap among modules. This function is useful to check the relationship of modules and to choose the proper modules during the exploratory discovery steps.

```
module.overlap(cluster.mod1, max.n=5)
```

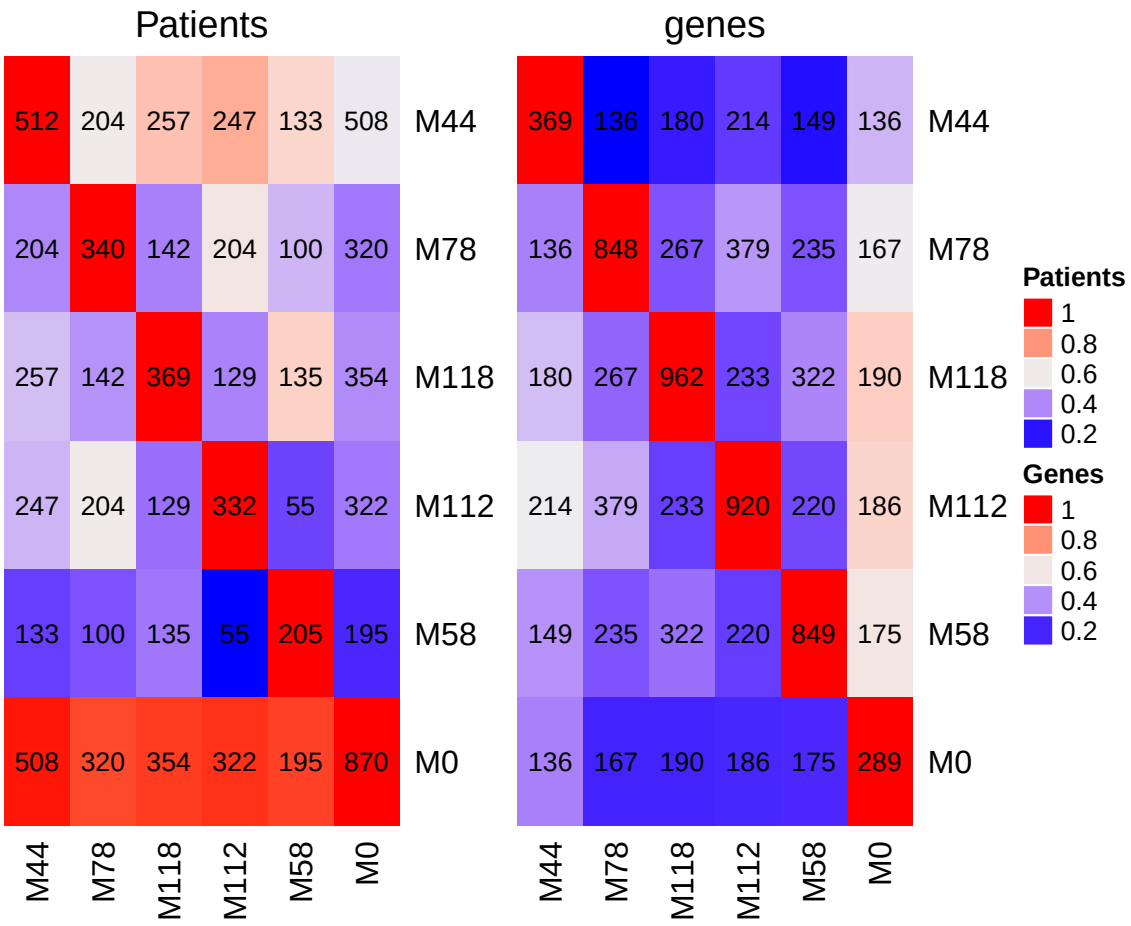


Figure 7: Module overlaps: genes and patients

0.11 Module comparison

`module.compare` is used to compare the modules from different studies, e.g. the different diseases or the different data for the same disease. In following examples, we calculated the modules for both ER+ and ER- breast cancer patients. The modules are compared with `module.compare`.

```
res.mod1 <- seed.module(deg[,er.pos],  min.genes=100, min.patients=50,cutoff=0.85, cores=4)
res.mod1 <- cluster.module(res.mod1)
#modules of ER+ samples
res.mod2 <- seed.module(deg[,er.neg],  min.genes=100, min.patients=50,cutoff=0.85, cores=4)
res.mod2 <- cluster.module(res.mod2)
#modules of ER- samples
```

```
module.compare(res.mod1, res.mod2, max.n1=10, max.n2=10)
```

Please note that, although this function is designed to do module comparison, it does not work to find the modules with distinct composition among conditions.

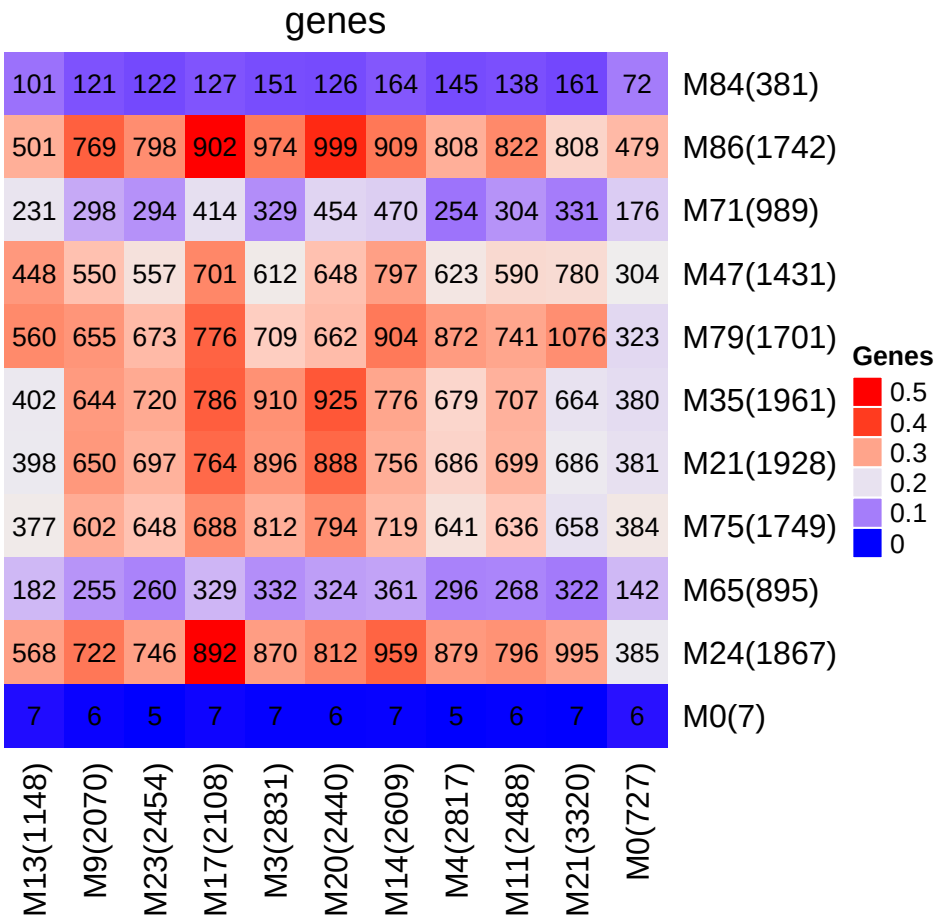


Figure 8: Module comparison

0.12 The curve for gene-patient number in module discovery

In the module discovery steps, a bi-clustering algorithm is applied to the binary DEG matrix. The DEGs of seed patients are gradually removed to a number of `min.genes`, which may result to a increased patient number from `min.patients`. In the output of `deg.module` tool, the track of gene-patient number change is recorded as `curve` for each module.

```
>names(cluster.mod1[["M1"]][["curve"]])
[1] "no.gene"    "no.patient" "score"
>head(cluster.mod1[["M1"]][["curve"]][["no.gene"]])
[1] 6616 6503 6475 6365 6264 6225
>head(cluster.mod1[["M1"]][["curve"]][["no.patient"]])
[1] 10 11 12 13 14 15
```

`module.curve` can show the patient-gene numbers in a simple way.

```
module.curve(cluster.mod1, "M1")
```

In this curve, `module.curve` highlights the points of "max.genes", "model","max.patients".

0.13 Modelling of patient-gene number in module discovery

With the default setting, `deg.module` may fail to find the best breakpoints for the `model` of some modules. It is possible for users to modify the `model` results for all or some modules. `module.modelling` provides such a tool. It provides two manners to modify the modelling results by either setting `keep.gene.num` or change the `model.method`.

Users can change the `model` results by manually setting the `keep.gene.num`, which is the the gene number where the `model` results is select. `keep.gene.num` can a integer value or a vector. If it is a integer number, all the modules will have the same `keep.gene.num` and this will change the modelling results for all the modules. If it is a vector, its elements should have module names as their names. Otherwise, only the first element will be used and all the modules will be set. When `keep.gene.num` is vector, it is not necessary to have the same length as modules. It is possible to only changes some of the modules. And the left modules will use the default setting.

In current version, `model.method` has four possible values: "slope.clustering", "max.square", "min.slope" and "min.similarity", which indicate the different four different modelling methods: * `slope.clustering` has maximum slope changes, which may indicate the inclusion/exclusion of molecular mechanisms. * `max.square` is the gene-patients number that has the maximum product; * `min.slope` is the point with minimum slope in gene-patient number curve; * `min.similarity` is based on the similarity scores and the point with minimum similarity scores is choosed.

```
x=c(100,300)
names(x)<-c("M1","M3")
new.cluster.mod1=module.modelling(cluster.mod1, keep.gene.num = x, method='slope.clustering',
                                  cores=4)
#here, only "M1" and "M3" are modified
new.cluster.mod1=module.modelling(cluster.mod1, keep.gene.num = 150)
# here, all the modules are modified
module.curve(new.cluster.mod1, "M1")
```

0.14 Find the modules associated with feature patients or genes

In complex diseases, many patients have the similar clinical trait or carry the same gene mutation. These feature patients are supposed to affected by the common mechanism. `model.screen` is used to find the modules that are potentially associated with the feature patients or genes.

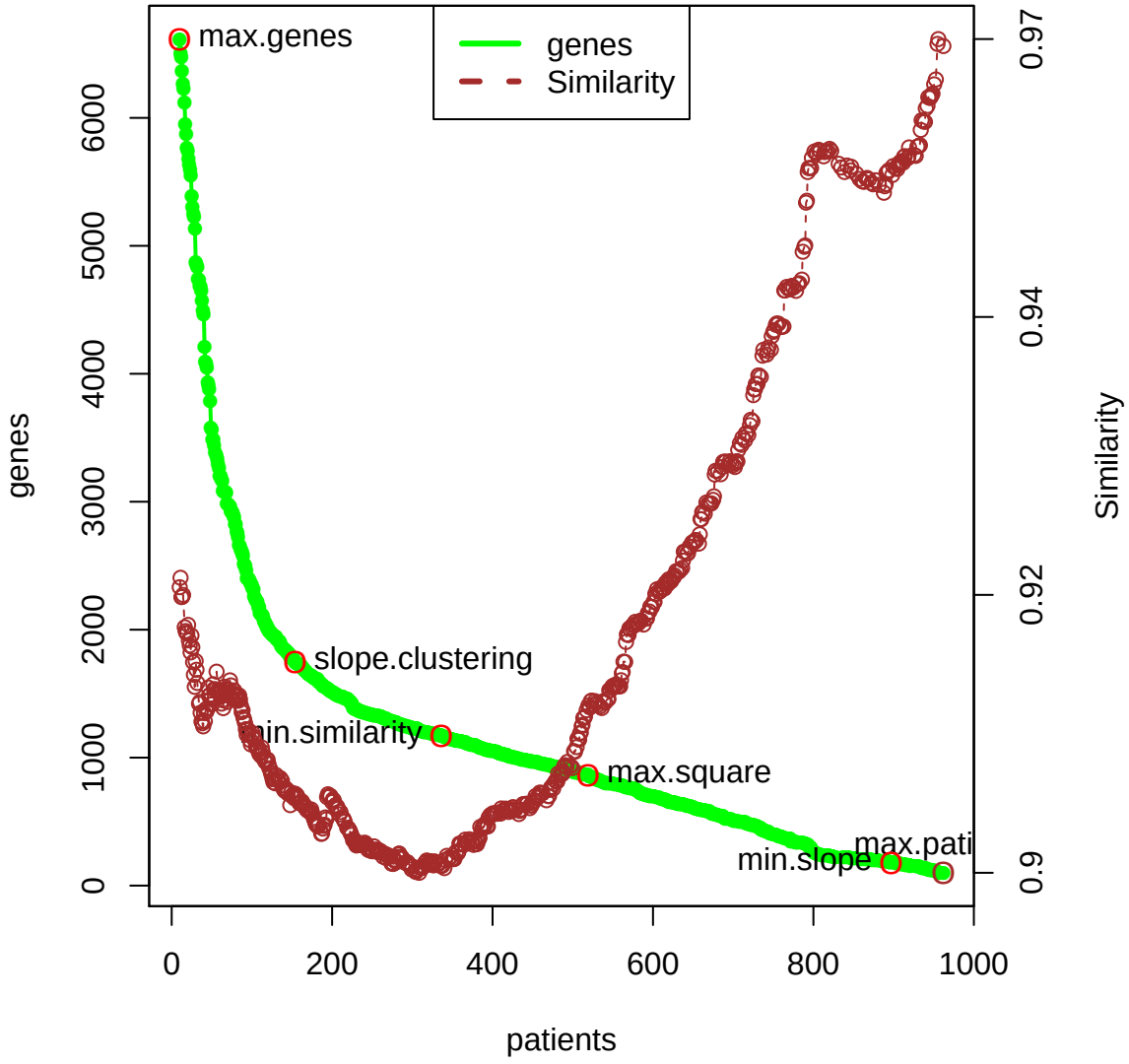


Figure 9: Patient-gene curve

```
module.screen(cluster.mod1, feature.patients=brca1.mutated.patients)
#search modules

module.screen(seed.mod1, feature.patients=brca1.mutated.patients,
              method="fisher.test")
```

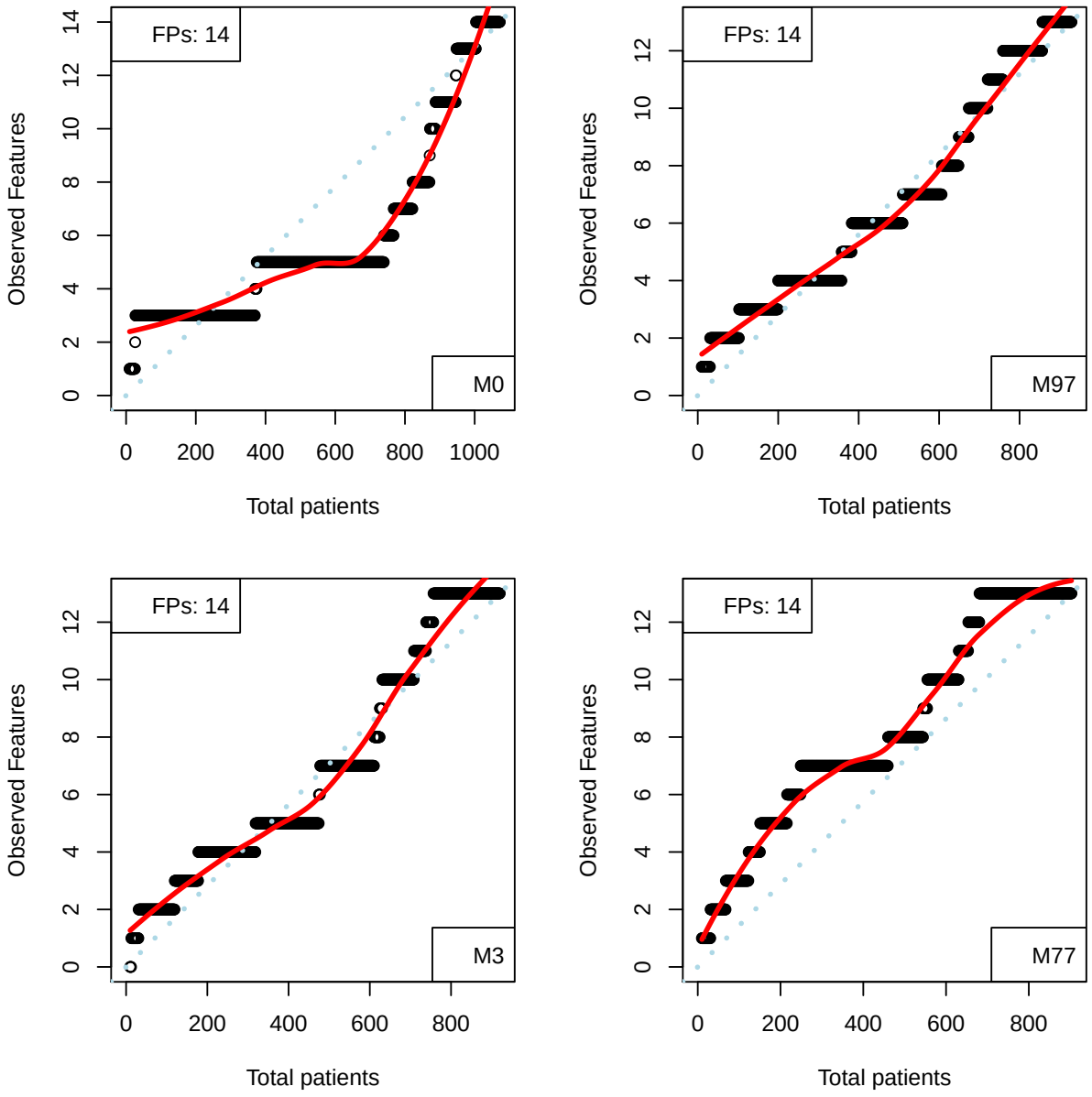


Figure 10: Module association with BRCA1

0.15 Functional association of predicted modules

To understand the disease relevance of modules that predicted by modelling to bi-clustering process, we propose a functional annotation based method for exploratory validation.

```
genes=cluster.mod1[["M1"]][["genes.removed"]]
# split the genes into overlapped windows
nas<-as.character(seq(1, length(genes)-400, by=50));

#functional annotation
library(clusterProfiler)
go.res=lapply(nas, function(x){
    ges=genes[x:(x+400)]
    eg = bitr(ges, fromType="SYMBOL", toType="ENTREZID", OrgDb="org.Hs.eg.db")
    ego <- enrichGO(gene          = unique(as.vector(eg$ENTREZID)),
                    OrgDb         = org.Hs.eg.db,
                    ont           = "BP",
                    pAdjustMethod = "BH",
                    pvalueCutoff  = 1,
                    qvalueCutoff  = 1,
                        readable      = TRUE)


    result=ego@result;
    rr=as.vector(result$GeneRatio);
    rr=as.numeric(gsub("\\/\\d+","",rr, perl=T))
    names(rr)<-as.vector(result$Description);
    out=data.frame(rr=rr, p=as.vector(result$pvalue))
    return(out)
})
names(go.res)<-as.character(nas)

# collect the GO terms
gos=vector()
for(x in nas){
    gos=append(gos, row.names(go.res[[x]]));
}
gos=unique(gos)

#collect the gene number
res1=matrix(ncol=length(nas), nrow=length(gos));
colnames(res1)<-nas;
row.names(res1)<-gos;
for(x in nas){
    res1[,x]=as.vector(go.res[[x]][gos,]$rr);
}

#collect the p-value
res2=matrix(ncol=length(nas), nrow=length(gos));
colnames(res2)<-nas;
row.names(res2)<-gos;
for(x in nas){
    res2[,x]=as.vector(go.res[[x]][gos,]$p);
}

res1=apply(res1, c(1,2), function(x) if(is.na(x)) 0 else x)
res2=apply(res2, c(1,2), function(x) if(is.na(x)) 1 else x)

#one example
```

```
test.go="response to estrogen";

#location of the removed genes
location=nas +200;

df=data.frame(location=res[test,], x=res1[test.go,]] p=res2[test.go,])

library(ggplot2)
qplot(x, cc, data=df, color=-log10(p), geom=c("point", "smooth"),
      xlab="Windows location",ylab="Genes with GO annotation")
+ scale_colour_gradientn(colours=rainbow(3)[c(3,2,1)])
+ annotate(geom="text", x=2000, y=17, label="GO:0043627:response to estrogen",
           color="green",size=4)
```

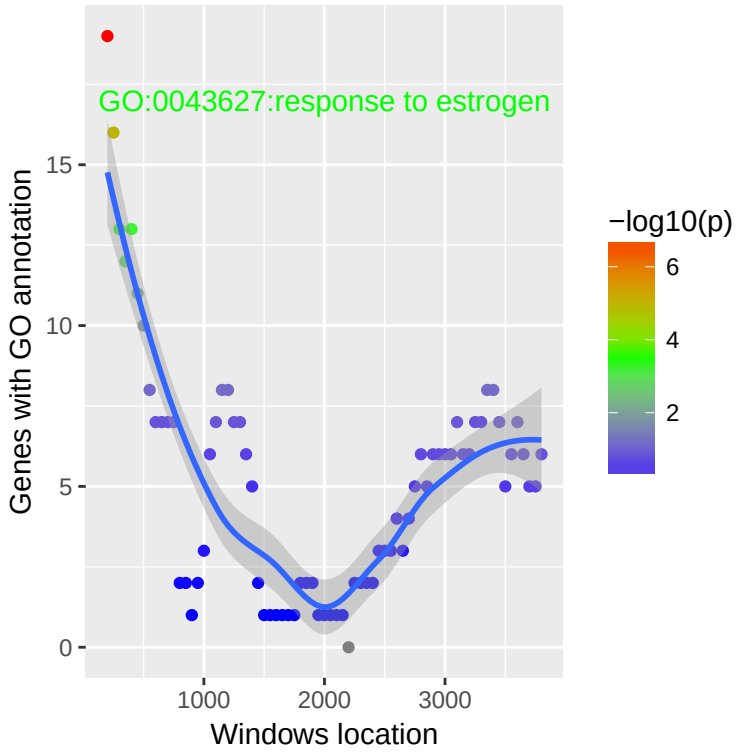


Figure 11: Module association with 'estrogen response'

0.16 Case application: Representive patient

TO be update soon