MineICA: Independent component analysis of transcriptomic data

Anne Biton, Andrei Zinovyev, Emmanuel Barillot, François Radvanyi.

May 4, 2016

Abstract

MineICA supplies a framework for the storage and the study of a decomposition resulting from the application of independent component analysis (ICA) to transcriptomic data. It allows to integrate additional data associated with the samples (other molecular data, as well as clinical and pathological data) and data associated with the genes. It defines a new class *IcaSet* extending the class *eSet* of the package *Biobase*, which allows to store the inputs (genomic dataset and sample information) and outputs (mixing and source matrix) of ICA. *MineICA* helps the biological interpretation of the components by studying their association with variables (e.g sample annotations) and biological processes, and enables the comparison of components from different datasets using correlation-based graph. In practice, by creating interactive summarization of the results and comprehensive plots, *MineICA* makes much easier the interpretation of the numerous data resulting from the application of ICA to transcriptomic data.

1 Introduction

Unlike ICA, clustering methods and PCA are routinely applied to perform unsupervised analysis of genomic high-throughput data. Several studies highlighted the outperformance of ICA over PCA and clustering-based methods in obtaining a more realistic decomposition of the expression data into consistent patterns of coexpressed and coregulated genes **????** associated with the studied phenotypes, like histological grade or estrogen receptor status in breast cancer **?**. Unlike PCA, ICA does not impose an orthogonality constraints between the independent components (ICs). The less frequent use of ICA analysis in bioinformatics studies may be explained by the non-trivial interpretation of its outputs. The aim of *MineICA* is to make the most of the ICA by making available methods to make easier the interpretation of its results.

Several ICA algorithms exist and generally rely on random initializations and compute non-unique solutions. The analysis of the reproducibility of the components across datasets is thus a crucial point in the analysis by for example enabling the selection of components that do not arise from a local minima. *MineICA* implements the study of the component reproducibility among different data sets through correlation-based graphs.

ICA provides a decomposition of the expression matrix $X = AS$, where A, the mixing matrix, contains the activity of the components on the samples (e.g tumor samples) and S, the source matrix, provides the contribution of each feature (e.g genes) to the components. The source matrix S is thus used to biologically interpret the components by studying their contributing genes, and the matrix A is used to associate the component with sample features by studying the distribution of the samples on the components according to their characteristics (e.g clinical or molecular variables).

2 Software features

MineICA offers the following functionalities:

Storage of the ICA results *MineICA* implements the class *IcaSet* whose aim is to contain and describe an ICA decomposition of high-throughput data.

Storage of analysis parameters *MineICA* implements the class *MineICAParams* which aims at containing parameters required for the analysis of the ICA results.

Association with variables *MineICA* proposes functions to test whether qualitative and quantitative variables (e.g sample annotations) are differently distributed on the components or differently distributed among clusters defined on the components.

Annotation of the features The package also provides functions to easily describe feature (e.g gene) annotations using *biomaRt*. The resulting annotation being displayed in HTML files.

Association with gene sets *MineICA* provides functions to run enrichment analysis of the contributing genes using package *GOstats*.

Visualization *MineICA* provides functions to visualize heatmaps of the contributing features, distribution of the variables on the components, or correlation graph between different ICA.

3 Case study

Using microarray-based gene expression data of 200 breast cancer tumors stored in the package `breastCancerMAINZ ?`, this vignette shows how *MineICA* can be used to study an ICA-based decomposition.

3.1 Loading the library and the data

We first load some dependent libraries :

```
> library(Biobase)
> library(plyr)
> library(ggplot2)
> library(foreach)
> library(xtable)
> library(biomaRt)
> library(GOstats)
> library(cluster)
> library(marray)
> library(mclust)
> library(RColorBrewer)
> library(igraph)
> library(Rgraphviz)
> library(graph)
> library(colorspace)
```

```
> library(annotate)
> library(scales)
> library(gtools)
```

We then load the *MineICA* package by typing or pasting the following codes in R command line:

```
> library(MineICA)
```

3.2 Creation of an *IcaSet* object

Class *IcaSet* extends *eSet* class of package *Biobase*. The *eSet* class won't be described here, please refer to the documentation for details about the attributes of the class `http://www.bioconductor.org/packages/2.12/bioc/vignettes/Biobase/inst/doc/Qviews.pdf`. Reading the documentation of `expressionSet` class, another subclass of *eSet*, may also be very useful `http://www.bioconductor.org/packages/release/bioc/vignettes/Biobase/inst/doc/ExpressionSetIntroduction.pdf`.

Beside including slots of *eSet* class, *IcaSet* class includes additionnal slots in order to contain the ICA outputs A and S (slots `A, S, SByGene`) and information regarding the components (slots `compNames, indComp`, ...). You can get an overview of the structure and available methods by reading the help page:

```
> help(IcaSet)
```

IcaSet class proposes two levels of storage for the data, the "feature" and "gene" levels. The slots `S` (source matrix) and `dat` (original data) refer to the feature level, while the slots `SByGene` (source matrix indexed by genes) and `datByGene` (data indexed by genes) refer to the gene level. It allows to store at the same time the results of ICA applied to the original data indexed by features, these features speaking generally not for themselves (eg, probe set IDs), and the data indexed by annotations of these features into a more comprehensive ID (e.g gene ids).
By default, in *MineICA*, the second level of annotation is called the "gene" level but it may in fact correspond to any other annotation (e.g, isoforms, exons, ...).

In addition to the demands of the *eSet* class for object validity, validity method for *IcaSet* enforces that the sample, feature, and gene names of the slot elements are identical. For example, the row names of the `phenoData` (the sample annotations) and mixing matrix `A` must be similar to the column names of `dat` and `datByGene`. Similarly, row names of `S` and `SByGene` must be similar to row names of `dat` and `datByGene`. The number of components must also be consistent between `A` and `S`.

3.2.1 load an example of expression data

We load the `eSet mainz` included in the data package `breastCancerMAINZ`.

```
> ## load Mainz expression data and sample annotations.
> library(breastCancerMAINZ)
> data(mainz)
> show(mainz)
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 22283 features, 200 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: MAINZ_BC6001 MAINZ_BC6002 ... MAINZ_BC6232 (200 total)
  varLabels: samplename dataset ... e.os (21 total)
  varMetadata: labelDescription
featureData
  featureNames: 1007_s_at 1053_at ... AFFX-TrpnX-M_at (22283 total)
  fvarLabels: probe Gene.title ... GO.Component.1 (22 total)
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
  pubMedIds: 18593943
Annotation: hgu133a
```

```
> ## we restrict the data to the 10,000 probe sets with the highest IQR
> mainz <- selectFeatures_IQR(mainz,10000)
```

3.2.2 Run ICA

We now run the *JADE* algorithm to compute an ICA decomposition of the Mainz data.

```
> library(JADE)
> ## Features are mean-centered before ICA computation
> exprs(mainz) <- t(apply(exprs(mainz),1,scale,scale=FALSE))
> colnames(exprs(mainz)) <- sampleNames(mainz)
> ## run ICA-JADE
> resJade <- runICA(X=exprs(mainz), nbComp=5, method = "JADE", maxit=10000)
```

Another ICA algorithm, `fastICA`, is implemented in `R` and may be run with function `runICA`. FastICA relies on random initializations and the estimated components may vary between iterations. A way to alleviate this problem is to run `fastICA` several times, cluster the estimates, and use as the final estimates the centrotypes of the clusters. This strategy is proposed in the matlab package *icasso* **?**. The function `clusterFastICARuns` implements this strategy and can be used to estimate the components:

```
> library(fastICA)
> ## Random initializations are used for each iteration of FastICA
> ## Estimates are clustered using hierarchical clustering with average linkage
> res <- clusterFastICARuns(X=exprs(mainz), nbComp=5, alg.type="deflation", nbIt=10,
+                           funClus="hclust", method="average")
```

The returned estimates are ranked according to their Iq indices which measure the compactness of the clusters and are defined as the differences between the intra-cluster similarity and the extra-cluster similiarity **?**.

3.2.3 Create a *MineICAParams* object, function `buildMineICAParams`

Before building an IcaSet instance, we need to create a *MineICAParams* instance that will contain a few parameters used during the analysis of the ICA decomposition.
You need to specify the directory where you would like to put the outputs of the analysis (slot `resPath`), the threshold applied to the projection values used to select the contributing elements (slot `selCutoff`), and the threshold you would like to use for statistical significance (slot `pvalCutoff`):

```
> ## build params
> params <- buildMineICAParams(resPath="mainz/", selCutoff=3, pvalCutoff=0.05)
```

If the original data and the ICA outputs A and S were stored in files, the file names would have been included in slots `annotfile, Sfile, Afile`, and `datfile`.

3.2.4 Create an *IcaSet* instance, function `buildIcaSet`

Mainz data and the corresponding ICA results have now to be stored in an *IcaSet* object. This task is made easier thanks to the function `buildIcaSet`.

Before building the *IcaSet* object, several information (corresponding to *IcaSet* slots) regarding the feature and gene ids remain to be defined.

annotation: This slot contains the name of an annotation package for the data, if available.
The Mainz data are from HG-U133A microarrays and are indexed by Affymetrix probe set ids. The corresponding annotation package is `hgu133a.db` and must be loaded:

```
> ## load annotation package
> library(hgu133a.db)
```

attribute `typeID`: The slot `typeID` of an *IcaSet* object includes the types of ids to be used for the annotation of the features, and the description of the feature and/or gene ids.
`typeID` encompasses three elements to be defined:

`geneID_annotation`**:** defines the object supported by the annotation package (if provided) needed to annotate the features into genes. To see the list of the available objects in the given package:

```
> ls("package:hgu133a.db")
```

```
 [1] "hgu133a"               "hgu133a.db"            "hgu133aACCNUM"
 [4] "hgu133aALIAS2PROBE"    "hgu133aCHR"            "hgu133aCHRLENGTHS"
 [7] "hgu133aCHRLOC"         "hgu133aCHRLOCEND"      "hgu133aENSEMBL"
[10] "hgu133aENSEMBL2PROBE"  "hgu133aENTREZID"       "hgu133aENZYME"
[13] "hgu133aENZYME2PROBE"   "hgu133aGENENAME"       "hgu133aGO"
[16] "hgu133aGO2ALLPROBES"   "hgu133aGO2PROBE"       "hgu133aMAP"
[19] "hgu133aMAPCOUNTS"      "hgu133aOMIM"           "hgu133aORGANISM"
```

```
[22] "hgu133aORGPKG"        "hgu133aPATH"          "hgu133aPATH2PROBE"
[25] "hgu133aPFAM"          "hgu133aPMID"          "hgu133aPMID2PROBE"
[28] "hgu133aPROSITE"       "hgu133aREFSEQ"        "hgu133aSYMBOL"
[31] "hgu133aUNIGENE"       "hgu133aUNIPROT"       "hgu133a_dbInfo"
[34] "hgu133a_dbconn"       "hgu133a_dbfile"       "hgu133a_dbschema"
```

Here we will use `"SYMBOL"` for Gene Symbols. If no annotation package is provided, this element is not useful and *biomaRt* is used to perform the annotation if required.

The two following elements are the IDs used to query *biomaRt*. A database of interest first needs to be specified. Here we use Ensembl for human.

```
> mart <- useMart(biomart="ensembl", dataset="hsapiens_gene_ensembl")
```

`geneID_biomart:` specifies the type of gene id, and is used for the description of the genes and features if no annotation package is provided. It must be one of the IDs available in the filters of the `mart` object:

```
> listFilters(mart)[120:125,]
```

```
                        name
120         with_affy_hc_g110
121        with_affy_hg_focus
122        with_affy_u133_x3p
123        with_affy_hg_u133a
124      with_affy_hg_u133a_2
125 with_affy_hg_u133_plus_2
                                                 description
120         with Affymetrix Microarray hc g110 probeset ID(s)
121        with Affymetrix Microarray hg Focus probeset ID(s)
122        with Affymetrix Microarray u133 x3p probeset ID(s)
123        with Affymetrix Microarray hg u133a probeset ID(s)
124      with Affymetrix Microarray hg u133a 2 probeset ID(s)
125 with Affymetrix Microarray hg u133 plus 2 probeset ID(s)
```

Here we will use `geneID_biomart='hgnc_symbol'` for Gene Symbols.

`featureID_biomart:` specifies the type of feature ID, must be one of the attributes available in `mart`:

```
> listAttributes(mart)[grep(x=listAttributes(mart)[,1],pattern="affy")[1:5],]
```

```
                  name                  description         page
100        affy_hc_g110        Affy HC G110 probeset feature_page
101       affy_hg_focus       Affy HG FOCUS probeset feature_page
102 affy_hg_u133_plus_2 Affy HG U133-PLUS-2 probeset feature_page
103     affy_hg_u133a_2     Affy HG U133A_2 probeset feature_page
104       affy_hg_u133a       Affy HG U133A probeset feature_page
```

HG-U133A probe sets correspond to `affy_hg_u133a`.

The function `buildIcaSet` encompasses the step of feature annotation. During the annotation step (either performed using the annotation package or *biomaRt*) if several features are available for a same gene, the median value across those features is attributed to the gene.

Data can also be provided at the final annotation level (e.g `dat` and `S` are already indexed by gene ids), in that case please use `alreadyAnnot=TRUE` in the function `buildIcaSet` so that no annotation will be performed.

We can now build the object `icaSetMainz` with help of function `buildIcaSet`:

```
> ## Define typeID, Mainz data originate from affymetrix HG-U133a  microarray
> ## and are indexed by probe sets.
> ## The probe sets are annotated into Gene Symbols
> typeIDmainz <-  c(geneID_annotation="SYMBOL", geneID_biomart="hgnc_symbol",
+                     featureID_biomart="affy_hg_u133a")
> ## define the reference samples if any, here no normal sample is available
> refSamplesMainz <- character(0)
> resBuild <- buildIcaSet(params=params, A=data.frame(resJade$A), S=data.frame(resJade$S),
+                         dat=exprs(mainz), pData=pData(mainz), refSamples=refSamplesMainz,
+                         annotation="hgu133a.db", typeID= typeIDmainz,
+                         chipManu = "affymetrix", mart=mart)
> icaSetMainz <- resBuild$icaSet
> params <- resBuild$params
```

3.2.5 *IcaSet* basics

An instance of *IcaSet* has been built, we now explore some of the basic operations.

When printed, a brief summary of the contents of the object, based on the one available in class *eSet*, is displayed:

```
> icaSetMainz
```

```
Number of components: 5
Component labels: 1 2 3 4 5
IcaSet (storageMode: lockedEnvironment)
assayData: 10000 features, 200 samples
  element names: dat
protocolData: none
```

```
phenoData
  sampleNames: MAINZ_BC6001 MAINZ_BC6002 ... MAINZ_BC6232 (200 total)
  varLabels: samplename dataset ... e.os (21 total)
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation: hgu133a.db
```

Accessing A number of accessor functions are available to extract data from an *IcaSet* instance. We will describe the most common accessor functions, most of them being inherited from class *eSet*.

You can access the phenotype data using `pData`:

```
> annot <- pData(icaSetMainz)
```

The columns of the phenotype data, are called the variables. The variable labels can be retrieved using the function `varLabels` and one variable can be accessed using $:

```
> varLabels(icaSetMainz)[1:5]
```

```
[1] "samplename" "dataset"    "series"     "id"         "filename"
```

```
> icaSetMainz$grade[1:5]
```

```
[1] 2 3 3 2 2
```

The feature names and their annotations (called "genes" by default), such as the sample names can respectively be retrieved using the functions `featureNames`, `geneNames`, and `sampleNames`:

```
> featureNames(icaSetMainz)[1:5] # probe set ids
```

```
[1] "1255_g_at" "1320_at"   "1405_i_at" "1431_at"   "1438_at"
```

```
> geneNames(icaSetMainz)[1:5] #gene symbols
```

```
[1] "GUCA1A" "PTPN21" "CCL5"   "CYP2E1" "EPHB3"
```

```
> sampleNames(icaSetMainz)[1:5]
```

```
[1] "MAINZ_BC6001" "MAINZ_BC6002" "MAINZ_BC6003" "MAINZ_BC6004" "MAINZ_BC6005"
```

The data ICA was applied to and its annotation into genes (if available) can be accessed using the functions `dat` and `datByGene`

```
> head(dat(icaSetMainz)) #probe set level
> head(datByGene(icaSetMainz)) #gene level
```

An ICA decomposition consists of two matrices, the mixing matrix A containing the sample contributions and the source matrix S containing the feature projections. They can be retrieved from an *IcaSet* object using the methods of the same name: `A` and `S`. The method `S` returns matrix S at the feature level (e.g, containing projection of the features), while `SByGene` returns the projection values at the gene level:

```
> A(icaSetMainz)
> S(icaSetMainz)
> SByGene(icaSetMainz)
```

The number of components computed by ICA, but also their labels and indices can be extracted with `nbComp, compNames`, and `indComp`:

```
> nbComp(icaSetMainz)
> compNames(icaSetMainz)
> indComp(icaSetMainz)
```

For graphical purpose, an *IcaSet* object includes a slot `witGenes` containing either one gene id per component (or whatever is refered as the level "gene"), or one feature id per component if the *IcaSet* object has only one level of annotation. Each "witness" is a contributor of the component and is used to denote the direction of the expression according to the direction of the component. These witnesses can be automaticall selected with the function `selectWitnessGenes` and are automatically defined when an *IcaSet* object is created through `buildIcaSet`. By default, for a given component, a witness gene corresponds to an individual having an absolute scaled projection value larger than `selCutoff` in at most one IC. Its sign of contribution should be the same than the majority of the selected contributing genes.

```
> witGenes(icaSetMainz)[1:5]
```

```
      1         2         3         4         5
"IGHM"   "GABRP"   "CDCA8"  "PICALM" "COL11A1"
```

```
> ## We can for example modify the second contributing gene
> witGenes(icaSetMainz)[2] <- "KRT16"
```

Accessing slots in different data formats Slots `A, S,` and `SByGene` are `data.frame` objects. A common need is to extract these data.frame in the form of a list where row names are preserved. It can be done using the functions `Alist, Slist`, and `SlistByGene`.

Setting The names of the accession functions described above, can also be used for setting the slots of an *IcaSet* object by adding operator `<-` and the new value. For example:

```
> compNames(icaSetMainz) <- paste("IC",1:nbComp(icaSetMainz),sep="")
```

Subsetting Subsetting an IcaSet is very similar to subsetting the expression matrix that is contained within the IcaSet to a subset of features/genes (first argument) and samples (second argument), except for a third argument that allows to subsets the components.

```
> ## select tumor samples of grade 3
> keepSamples <- sampleNames(icaSetMainz)[icaSetMainz$grade=="3"]
> ## Subset icaSetMainz to the grade-3 samples
> icaSetMainz[,keepSamples]
> ## Subset icaSetMainz to the grade-3 samples and the first five components
> icaSetMainz[,keepSamples,1:5]
> ## Subset icaSetMainz to the first 10 features
> icaSetMainz[featureNames(icaSetMainz)[1:10],keepSamples]
```

Useful basic functions Other functions which allow to extract data from an *IcaSet* object are available.

Select the contributing features or genes: When applying ICA decomposition to genomic data, for example here gene expression data, the distribution of the gene projections on the ICs is expected to be super-Gaussian: a large portion of genes follows a (super-)Gaussian centered at zero and a small portion belongs to an outgrowth located on the right and/or on the left of the distribution. In order to select the elements belonging to this outgrowth, we used the conventional way based on a threshold. The thresholds can typically be 3 or 4 standard deviations from the mean. We refer to the resulting selected genes as the "contributing genes".

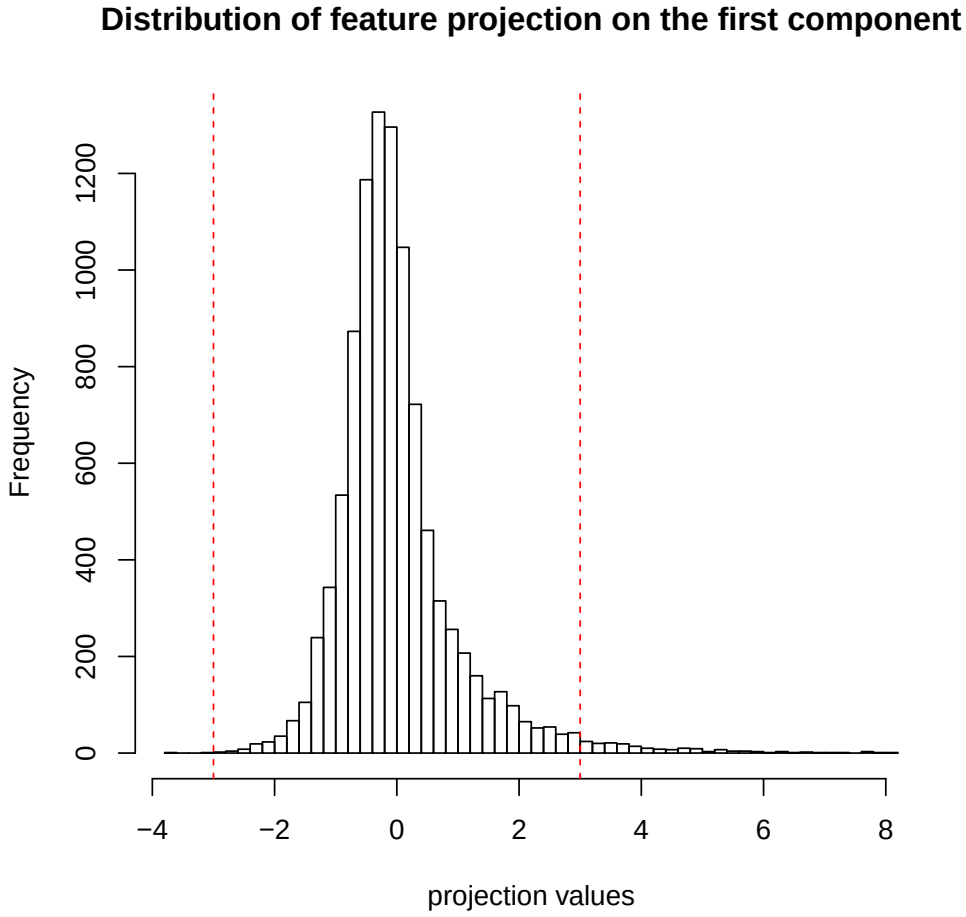
Here is the histogram of the projection values for the first component.

The function `selectContrib` allows to select the contributing elements from a list of projection values.

```
> ## Extract the contributing genes
> contrib <- selectContrib(icaSetMainz, cutoff=3, level="genes")
> ## Show the first contributing genes of the first and third components
> sort(abs(contrib[[1]]),decreasing=TRUE)[1:10]
```

```
    IGHM     NKG7 TNFRSF17      IGK     GZMK     IGKC    CXCL9   CXCL13
6.659809 6.463074 6.363992 5.851299 5.533274 5.505683 5.475967 5.429822
  JCHAIN IGLV1-44
5.223099 5.201415
```

```
> sort(abs(contrib[[3]]),decreasing=TRUE)[1:10]
```


Distribution of feature projection on the first component
Frequency
0
200
400
600
800
1000
1200
−4
−2
0
2
4
6
8
projection values

```
    MMP1    CDCA8      OGN      ELN  SCGB1D2    MFAP4    CDC45    ACKR1
5.717376 5.477812 5.371826 5.225109 5.192641 5.162009 5.140146 5.041739
   KRT14    CENPN
4.938156 4.613233
```

```
> ## One can also want to apply different cutoffs depending on the components
> ## for example using the first 4 components:
> contrib <- selectContrib(icaSetMainz[,,1:4], cutoff=c(4,4,4,3), level="genes")
```

Extract data of a specific component: The function `getComp` allows to extract the projection values and sample contribution of a specific component:

```
> ## extract sample contributions and gene projections of the second component
> comp2 <- getComp(icaSetMainz, level="genes", ind=2)
> ## access the sample contributions
> comp2$contrib[1:5]
```

```
MAINZ_BC6001 MAINZ_BC6002 MAINZ_BC6003 MAINZ_BC6004 MAINZ_BC6005
 -0.14515901   0.32744737  -0.09054997   0.10447404   0.23829698
```

```
> ## access the gene projections
> comp2$proj[1:5]
```

```
     GUCA1A      PTPN21        CCL5      CYP2E1       EPHB3
 0.25970790 -0.30932799  0.60651225  0.06477177 -2.51301758
```

3.3 Run global analysis

The function `runAn` enables to study an *IcaSet* object by calling all the functions dedicated to the analysis of an ICA decomposition in the package *MineICA*. The outputs are written in the path `resPath(params)`, each sub-directory containing the outputs of a specific analysis.

We apply the function `runAn` to the object `icaSetMainz`:

```
> ## select the annotations of interest
> varLabels(icaSetMainz)
```

```
 [1] "samplename"    "dataset"       "series"        "id"
 [5] "filename"      "size"          "age"           "er"
 [9] "grade"         "pgr"           "her2"          "brca.mutation"
[13] "e.dmfs"        "t.dmfs"        "node"          "t.rfs"
[17] "e.rfs"         "treatment"     "tissue"        "t.os"
[21] "e.os"
```

```
> # restrict the phenotype data to the variables of interest
> keepVar <- c("age","er","grade")
> # specify the variables that should be treated as character
> icaSetMainz$er <- c("0"="ER-","1"="ER+")[as.character(icaSetMainz$er)]
> icaSetMainz$grade <- as.character(icaSetMainz$grade)


> ## Run the analysis of the ICA decomposition
> # only enrichment in KEGG gene sets are tested
> runAn(params=params, icaSet=icaSetMainz, writeGenesByComp = TRUE,
+       keepVar=keepVar, dbGOstats = "KEGG")
```

The resulting plots and data are located in the main results path, which here is the "mainz/" current directory:

```
> resPath(params)
```

```
[1] "mainz/"
```

The sub-directories automatically created by the function `runAn` are the following:

ProjByComp/: contains the annotations of the features or genes, one file per component;

varAnalysisOnA/: contains two directories: 'qual/' and 'quant/' which respectively contain the results of the association between components and qualitative and/or quantitative variables;

Heatmaps/: contains the heatmaps (one pdf file per component) of the contributing genes by component;

varOnSampleHist/: contains the histograms of the sample contributions superimposed with the histograms of the groups of samples defined by the variables of interest (e.g tumor grade).

3.4 Run analysis by calling individual functions

The functions implicitely called by `runAn` can be run individually. In this section, we will provide examples of each of these functions.

3.4.1 Write description of contributing genes or features, function `writeProjByComp`

Each component is a direction in the space where axis are the samples and points are genes whose locations are defined by their expression profiles across samples. In matrix S, each component is thus defined by a vector of gene projection values. When applying ICA to gene expression data, each component is typically triggered by a group of genes co-expressed on a subset of samples. These genes responsible for the existence of the component will typically have high projections, we call them the *contributing* genes.

The first way to study a component is to look at its contributing genes. The function `writeProjByComp` allows to describe genes with a projection value higher than a given threshold on each component.

As in PCA, the components computed by ICA are defined up to their sign. On a given component, genes with opposite projection signs are elements whose expressions are anti-correlated on

the samples distributed at both ends of the component. The function `writeProjByComp` therefore orders genes by absolute projection values.

This function creates a HTML file per component containing the description of the contributing features or genes, and a file containing the projection values of each feature or gene across all components.

The needed information are queried through *biomaRt*. By default, the descriptors used to annotate the gene ids are their Gene Symbols, Ensembl IDs, biological description and genomic locations. If you would like to add descriptors, please fill argument `typeRetrieved`. Here we will content ourselves with the defaults ones. You can change the threshold used to select the genes to be described using the argument `selCutoffWrite`.

Here we are interested in the description of the projection values at the gene level (`level=■genes■`).

```
> resW <- writeProjByComp(icaSet=icaSetMainz, params=params, mart=mart,
+                         level='genes', selCutoffWrite=2.5)
> ## the description of the contributing genes of each component is contained
> ## in res$listAnnotComp which contains the gene id, its projection value, the number and
> ## the indices of the components on which it exceeds the threshold, and its description.
> head(resW$listAnnotComp[[1]])
```

```
    hgnc_symbol scaled_proj nbOcc_forThreshold:3 comp_forThreshold:3
154        IGHM        6.66                    1                    1
155        IGHM        6.66                    1                    1
228        NKG7       6.463                    1                    1
279    TNFRSF17       6.364                    1                    1
156         IGK       5.851                    1                    1
102        GZMK       5.533                    1                    1
                                                                                  description
154                      immunoglobulin heavy constant mu [Source:HGNC Symbol;Acc:HGNC:5541]
155                      immunoglobulin heavy constant mu [Source:HGNC Symbol;Acc:HGNC:5541]
228                  natural killer cell granule protein 7 [Source:HGNC Symbol;Acc:HGNC:7830]
279 tumor necrosis factor receptor superfamily member 17 [Source:HGNC Symbol;Acc:HGNC:11913]
156                                                                                      <NA>
102                                                granzyme K [Source:HGNC Symbol;Acc:HGNC:4711]
       chromosome_name start_position end_position   band strand
154 CHR_HSCHR14_3_CTG1      105854991    106236343 q32.33     -1
155                 14      105854220    105856218 q32.33     -1
228                 19       51371606     51372715 q13.41     -1
279                 16       11965107     11968068 p13.13      1
156               <NA>             NA           NA   <NA>     NA
102                  5       55024253     55034570  q11.2      1
    ensembl_gene_id
154 ENSG00000282657
155 ENSG00000211899
228 ENSG00000105374
279 ENSG00000048462
```

```
156            <NA>
102 ENSG00000113088
```

```
> ## The number of components a gene contributes to is available
> ## in res$nbOccInComp
> head(resW$nbOccInComp)
```

```
           gene nbOcc components sd_expr      1       2       3       4       5
S100A7   S100A7     2        2,4 3.86750 0.2359  -4.897   0.961  -3.699  -1.394
SCGB1D2 SCGB1D2     1          3 3.43860 -1.283   2.066  -5.184  0.4333 -0.4833
CPB1       CPB1     1          1 2.77680 -3.644   2.674 -0.8729  -1.339  0.9012
GRIA2     GRIA2     2        1,2 2.76040 -2.948   3.552 -0.2685  -0.166 -0.8901
MMP1       MMP1     2        3,5 2.64700  2.705 -0.9315    5.79 -0.3799   3.053
CYP4F8   CYP4F8     2        4,5 2.60910 -1.109  0.4198  -1.945    2.95   3.158
```

```
> ## The output HTML files are located in the path:
> genesPath(params)
```

```
[1] "mainz/ProjByComp/"
```

3.4.2 Plot heatmaps of the contributing elements, function `plot_heatmapsOnSel`

A way to visualize the pattern captured by a component is to draw the heatmap of its contributing features/genes. The function `plot_heatmapsOnSel` enables to plot the heatmaps of the contributing genes for each component. On those heatmap, features and samples are either ranked by their contribution value to the component, or clustered with hierarchical clustering.

Here we choose to study the data at the gene level (`level="genes"`), and a threshold of 3 is used for the selection of the contributing genes.

```
> ## selection of the variables we want to display on the heatmap
> keepVar <- c("er","grade")
> ## For the second component, select contributing genes using a threshold of 3
> ## on the absolute projection values,
> ## heatmap with dendrogram
> resH <- plot_heatmapsOnSel(icaSet = icaSetMainz, selCutoff = 3, level = "genes",
+                            keepVar = keepVar,
+                            doSamplesDendro = TRUE, doGenesDendro = TRUE, keepComp = 2,
+                            heatmapCol = maPalette(low = "blue", high = "red", mid = "yellow",
+                            file = "heatmapWithDendro", annot2col=annot2col(params))
> ## heatmap where genes and samples are ordered by contribution values
> resH <- plot_heatmapsOnSel(icaSet = icaSetMainz, selCutoff = 3, level = "genes",
+                            keepVar = keepVar,
+                            doSamplesDendro = FALSE, doGenesDendro = FALSE, keepComp = 2,
```

```
+                             heatmapCol = maPalette(low = "blue", high = "red", mid = "yellow",
+                             file = "heatmapWithoutDendro", annot2col=annot2col(params))
>
```

The heatmap where samples are ranked by sample contributions shows a group of tumors distributed at the left/negative end of the IC that strongly under-express and over-express sone of the contributing genes of the component, and whose pattern of expression is strongly anticorrelated with the tumors distributed at the opposite end of the component (Figure **??**). According to the second row of the top panel displaying the tumor annotations, these tumors are preferentially ER negative.

3.4.3 Gene enrichment analysis, function `runEnrich`

To obtain a biological interpretation of the component, it can be useful to study the association of its contributing genes with gene sets grouping genes involved in a same biological processes or sharing a same factor of regulation. In order to identify the gene sets which are enriched in the list of selected (contributing) genes, the function **`runEnrich`** uses `R GOstats` package **?** which makes use of a hypergeometric distribution to test the over-representation of a gene set in a given list of genes.

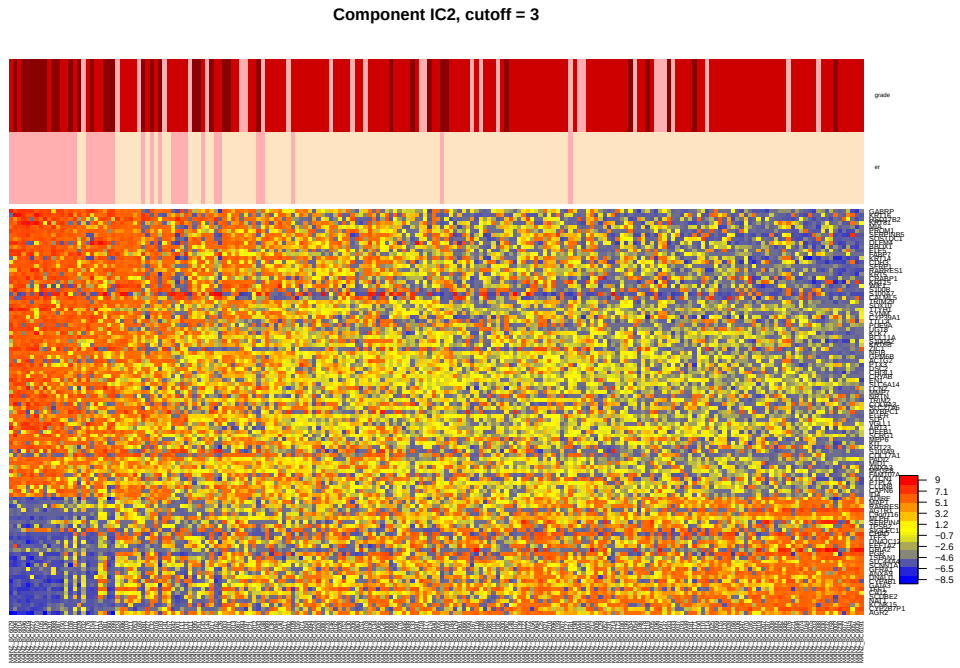
```
> ## run enrichment analysis on the first three components of icaSetMainz,
> ## using gene sets from the ontology 'Biological Process' (BP) of Gene Ontology (GO)
> resEnrich <- runEnrich(params=params,icaSet=icaSetMainz[,,1:3],
+                        dbs=c("GO"), ontos="BP")
```

The output **`resEnrich`** is a list whose each element contains results obtained on each database for every component tested. For each component, three enrichment results are available, depending on how contributing genes are selected: on the absolute projection values ("both"), on the positive projection ("pos"), and on the negative projection ("neg").

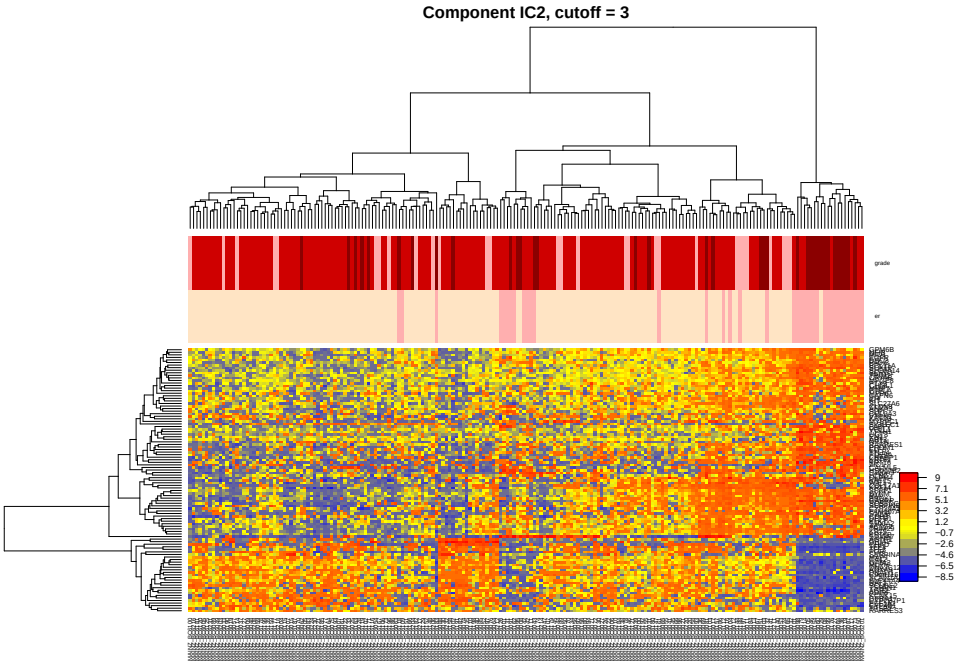
We can see that the first component is associated with immune reaction, the second component with epiderm development, and the third component with cell cycle:

```
> ## Access results obtained for GO/BP for the first three components
> # First component, when gene selection was based on the negative projection values
> head(resEnrich$GO$BP[[1]]$left)
```

```
      GOBPID       Pvalue OddsRatio ExpCount Count Size
1 GO:0006955 4.133986e-16  16.11393 2.180819    21  185
2 GO:0002694 3.535657e-14  10.23999 3.246918    23  199
3 GO:0050867 1.059836e-11  10.34766 2.382161    18  146
4 GO:0002429 2.104749e-11  13.19753 1.566352    15   96
5 GO:0050863 2.964082e-11  14.54225 1.337114    14   85
6 GO:0051251 3.414818e-11  14.36465 1.350438    14   85
                                                    Term
1                                          immune response
2                              regulation of leukocyte activation
```



(a)



(b)

Figure 1: Heatmap of component 2. The expression matrix is restricted to the contributing genes with an absolute scaled projection exceeding 3, and each gene expression profile is centered. In heatmap (a), genes and samples are ranked by their contribution to the IC.

```
3                              positive regulation of cell activation
4 immune response-activating cell surface receptor signaling pathway
5                                     regulation of T cell activation
6                         positive regulation of lymphocyte activation

1          CD7,MS4A1,CD27,CTSW,GZMA,HLA-DOB,IGHD,IGHM,IGJ,IL2RG,CXCL10,LTB,LY9,CXCL9,CCL18,CXCL
2 AIF1,CD2,CD3D,CD3G,CD247,CD27,CD37,CD38,HLA-DQB1,LCK,PTPRC,CCL5,CCL19,XCL1,EBI3,LILRB1,PTPN22
3                              AIF1,CD2,CD3D,CD3G,CD247,CD27,CD38,HLA-DQB1,LCK,PTPRC,CCL5,CCL19
4                                        CD3D,CD3G,CD247,CD38,HLA-DQB1,IGHG1,IGKC,IGLC1,LCK,PRF
5                                               AIF1,CD3D,CD3G,CD247,HLA-DQB1,PTPRC,CCL5,XCL1
6                                               AIF1,CD3D,CD3G,CD247,CD38,HLA-DQB1,PTPRC,CCL5
```

```
> # Second component
> head(resEnrich$GO$BP[[2]]$both, n=5)
```

```
      GOBPID       Pvalue OddsRatio   ExpCount Count Size
1 GO:0045104 2.160448e-05 19.682018 0.36675127     5   17
2 GO:0031581 6.046162e-05 26.782609 0.23730964     4   11
3 GO:0030318 3.943872e-04 14.405351 0.36675127     4   17
4 GO:0070488 4.615930e-04       Inf 0.04314721     2    2
5 GO:0072602 4.615930e-04       Inf 0.04314721     2    2
6 GO:0034329 1.079591e-03  4.298411 2.09263959     8   97
                                          Term
1 intermediate filament cytoskeleton organization
2                         hemidesmosome assembly
3                      melanocyte differentiation
4                          neutrophil aggregation
5                        interleukin-4 secretion
6                          cell junction assembly
                              In_geneSymbols
1                  DST,KRT14,KRT16,PKP1,SYNM
2                     DST,COL17A1,KRT5,KRT14
3                        EDN3,KIT,SOX10,MLPH
4                              S100A8,S100A9
5                                GATA3,VTCN1
6 DST,CDH3,COL17A1,GPM6B,KRT5,KRT14,SFRP1,UGT8
```

```
> # Third component, when gene selection was based  on the absolute projection values
> head(resEnrich$GO$BP[[3]]$both)
```

```
      GOBPID       Pvalue OddsRatio   ExpCount Count Size
1 GO:0048285 2.180659e-24 16.848654 3.18165337    32  150
2 GO:0051301 7.647658e-16 14.529070 2.14748665    21  107
```

```
3 GO:0007067 4.855310e-12 17.087428 1.18268701    14   65
4 GO:0007076 9.304760e-06       Inf 0.06363307     3    3
5 GO:0000086 1.957014e-05  8.181957 1.18781726     8   56
6 GO:0006271 5.656494e-05 27.266751 0.23332125     4   11
                                                 Term
1                                  organelle fission
2                                      cell division
3                                             mitosis
4                    mitotic chromosome condensation
5              G2/M transition of mitotic cell cycle
6 DNA strand elongation involved in DNA replication

1 BIRC5,BUB1,CCNA2,CDK1,CDC20,CDC25A,CENPE,IGF1,KIFC1,MYBL2,NEK2,AURKA,CCNB2,KIF23,DLGAP5,NDC8
2                                                                                BUB1,CCNA2,CDK1,CDC20,CDC25A
3
4
5
6
```

The function `runEnrich` also writes these enrichment results in HTML files located in the sub-directory "GOstatsEnrichAnalysis" of the result path.

3.4.4 Association with sample variables

Recall that a component is a direction in the gene space whose axis are defined by the samples. The mixing matrix A contains the coordinates of the components on the sample axis, we call these values the *sample contributions*.

The association of qualitative variables (e.g sample characteristics like tumor grade) with the components can be studied by comparing the contributions of the groups of samples they define. Depending on the number of groups formed by a given variable, their distribution can be compared either using a Wilcoxon (two groups) or a Kruskall-Wallis test (at least three groups). The function `qualVarAnalysis` tests whether the groups of samples formed by the qualitative variables are differently distributed on the components in terms of contribution value and plots the corresponding densities or boxplots using *ggplot2*.

If the levels of some variables in the `phenoData` of your *IcaSet* object are ordered (e.g, increasing tumor stage T1 T2 T3...), we advise you to declare these variables as factors whose levels are correctly ordered.

```
> ### Qualitative variables
> ## Compute Wilcoxon and Kruskall-Wallis tests to compare the distribution
> ## of the samples according to their grade and ER status on all components.
> resQual <- qualVarAnalysis(params=params, icaSet=icaSetMainz,
+                            keepVar=c("er","grade"),
+                            adjustBy="none", typePlot="boxplot",
+                            path="qualVarAnalysis/", filename="qualVar")
```

```
[1] "Plot distribution of samples on components according to variable er"
[1] "Comp 1"
[1] "Comp 2"
[1] "Comp 3"
[1] "Comp 5"
[1] "Plot distribution of samples on components according to variable grade"
[1] "Comp 1"
[1] "Comp 2"
[1] "Comp 3"
[1] "Comp 5"
```

The function creates an HTML file "qualVarAnalysis/qualVar.htm", containing p-values and links toward boxplots. If you would like to plot densities rather than boxplots, please use `'typePlot=density'`.

An example of boxplot is represented below for the second component and the ER status. As suggested by the heatmap, the distribution of the samples on this component is strongly associated with their ER status, the latter coming up at the positive end of the component.

When a variable is quantitative, its association with a component can be studied by computing its correlation with the sample contributions. The function `quantVarAnalysis` allows to compute the correlation tests and to draw the corresponding scatter plots using *ggplot2*.

```
> ### Quantitative variables
> ## Compute pearson correlations between variable 'age' and the sample contributions
> ## on all components.
> ## We are interested in correlations exceeding 0.3 in absolute value, and plots will only be
> ## for correlations exceeding this threshold.
> resQuant <- quantVarAnalysis(params=params, icaSet=icaSetMainz, keepVar="age",
+                              typeCor="pearson", cutoffOn="cor",
+                              cutoff=0.3, adjustBy="none",
+                              path="quantVarAnalysis/", filename="quantVar")
```

```
[1] "Scatter plot of samples contributions vs variable age"
[1] "Comp 2"
```

The absolute correlation between age and sample contributions exceeds 0.3 only for the second component.

```
> resQuant$cor[2]
```

```
[1] 0.3670033
```

The corresponding scatter plot is available in Figure **??**. A tendency of the women whose tumors are located at the positive end of the component to be younger indeed appears.

The function creates a HTML file "quantVar.htm" containing correlations values, p-values, and links toward scatter plots.

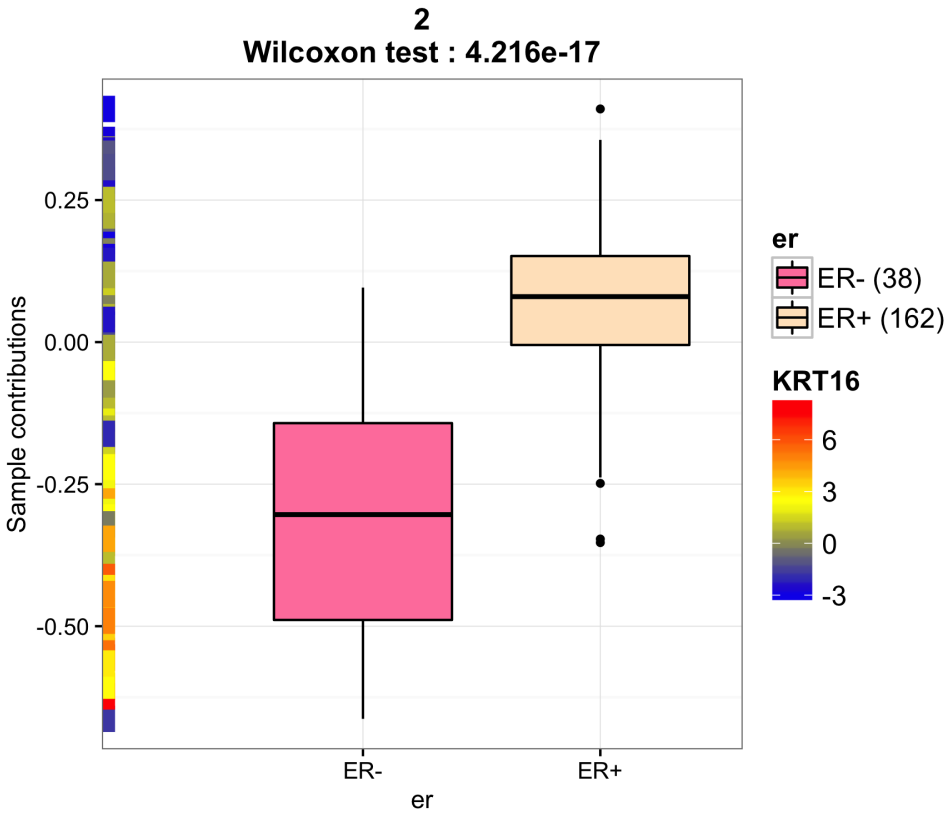


Figure 2: Example of boxplot representing the distribution of ER status on the third component. The Wilcoxon test p-value is available in the title of the plot. The legend indicates that the ER+ tumors are represented in beige while ER- are represented in light pink. The number of tumors in each group is given between brackets. The witness gene is *KRT16*. Each tumor sample is represented as a square point in the vertical line at the left end of the boxplots whose color denotes its amount of expression of the *KRT16* gene. The scale of these colors is denoted by a legend at the upper right of the graph.

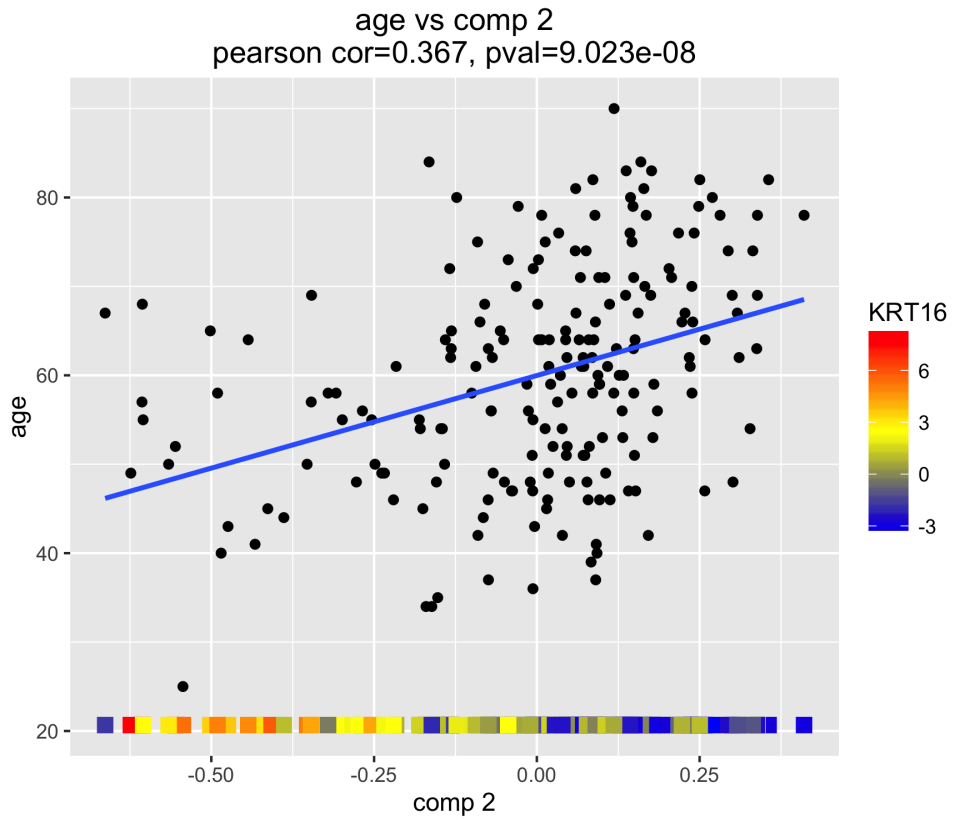


Figure 3: Scatter plot of AGE vs sample contributions. The witness gene is *KRT16*. At the bottom of the plot, each sample is represented by a square point whose colour denotes the expression value of the *KRT16* gene. The scale of these colors is denoted by a legend at the upper right of the graph. Note that the gene expression profiles were centered to have mean zero.

3.4.5 Clustering of the samples according to each component

Selection of samples associated with a component The selection of the samples associated with a component may be needed for experimental needs. ICA provides a continuous signal describing the activity of the components on the samples through the mixing matrix A. Genes which are contributors on the components can also be selected through their projections in matrix S (using an arbitrary threshold). Since the signal is continuous the selection of the samples contributing to a component rely on some arbitrary choices regarding:

- the data on which the clustering has to be applied on: clustering in one dimension on columns of A, or clustering on the expression matrix restricted to the contributing genes of the components?
- the number of clusters to use: two clusters if it is considered as a strictly bimodal signal, or three clusters if we assume the existence of a group of samples with an average behavior?
- the method of clustering to use: k-means, clustering based on mixture Gaussian modelling, hierarchical clustering, ...

We recommend to cluster the samples by using their contributions to the component.
If you would like to perform the clustering on the original data restricted to the contributing genes, please remind that they won't necessarily represent the whole pattern of expression captured by the component, the latter having been defined on all the features and not on a subset of them.

Study the bimodality of sample contributions in matrix `A` The distribution of the sample contributions on a component (contained in matrix A) is often bimodal, each mode corresponding to samples that over- or under-express the contributing genes of the component. The sample contributions can be visualized with histograms, overlaid by Gaussian mixtures computed, in this package, using package *mclust* **??**. When a strong bimodal distribution is observed, the intersection of the two Gaussian infered by function `Mclust` may be used to cluster the tumors. Here is an example by imposing two Gaussian on every vector of sample contributions:

```
> resmix <- plotAllMix(A=A(icaSetMainz), nbMix=2, nbBreaks=50)
```

The position of sub-groups of samples can be plotted in this histogram, in order to see if they are located at a specific end of the components. The function `plotPosAnnotInComp` allows to do so. The samples distributed at one end of component generally have either a strong over- or under-expression of its contributing genes.

Here is the example of the distribution of the tumors according to their ER status on the second component.

```
> ## plot the positions of the samples on the second component according to their ER status
> ## (in a file "er.pdf")
> plotPosAnnotInComp(icaSet=icaSetMainz, params=params, keepVar=c("er"), keepComp=2,
+                    funClus="Mclust")
```

Again, we can see that the negative end of the IC defines a cluster of tumors almost exclusively constituted of ER- tumors, while the ER+ tumors are primarily located on its right side. The expression profile of the gene witness, *KRT16*, indicates that the negative side corresponds to the over-expression of this gene and its counterparts compared to the positive side.

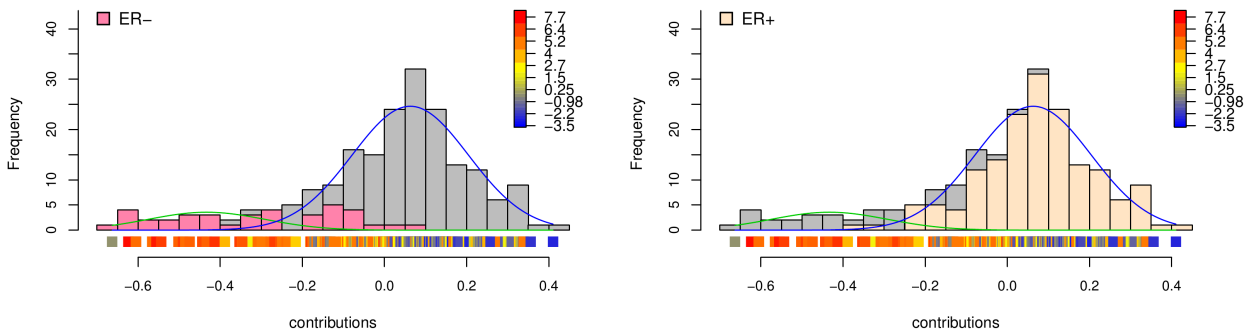


Figure 4: Distribution of ER status on the second component.. The histogram of each group is superimposed on the global histogram including contributions of all tumor samples. Two Gaussians were fitted on the distribution by mixture modeling using package *mclust*. The *p*-value at the top of the histogram provides the result of a chi-square test of association between each group and the clusters of samples formed by the two Gaussians.

Cluster samples, function `clusterSamplesByComp` The function `clusterSamplesByComp` allows to cluster the samples using either the mixing matrix A or the original data matrix restricted to the contributing individuals. The clustering can be performed using centroid-based clustering (function `kmeans`), hierarchical clustering (through functions `hclust` and `agnes`), Gaussian mixture models (using function `Mclust` or package *mclust*), or Partitioning Around Medoids (PAM) (functions `pam` and `pamk`).

The second component displays a bimodal distribution, we cluster the samples using the vector of sample contributions:

```
> ## clustering of the samples in 1-dim using the vector
> ## of sample contributions of the two first components
> ## and Gaussian mixture modeling (Mclust)
> clus1 <- clusterSamplesByComp(params=params, icaSet=icaSetMainz[,,,1:2],
+                               funClus="Mclust", clusterOn="A", nbClus=2, filename="comp1Mclus
> ## The obtained clusters are written in the file "comp1Mclus.txt" of the result path.
> clus1$clus[[2]][1:5]

MAINZ_BC6001 MAINZ_BC6002 MAINZ_BC6003 MAINZ_BC6004 MAINZ_BC6005
           2            2            2            2            2
```

It is also possible to perform several clusterings, using different algorithms or levels, with function `clusterSamplesOnComp_multiple`. We can for example compare the clustering performed with k-means applied to the vector of sample contributions and to the expression matrix restricted to the contributing genes:

```
> clus2 <- clusterSamplesByComp_multiple(params=params, icaSet=icaSetMainz[,,1:2],
+                                        funClus="kmeans", clusterOn=c("A","S"), level="features"
+                                        nbClus=2, filename="comparKmeans")
> ## The obtained clusters and their comparison with adjusted Rand indices are written
```

```
> ## in file "comparKmeans.txt" of the result path.
>
> ## Both clustering results are stored in a common data.frame
> head(clus2$clus)
```

```
            1_kmeans_onA 2_kmeans_onA 1_kmeans_onS 2_kmeans_onS
MAINZ_BC6001            1            1            1            1
MAINZ_BC6002            1            2            1            2
MAINZ_BC6003            1            2            2            1
MAINZ_BC6004            1            2            1            2
MAINZ_BC6005            1            2            1            2
MAINZ_BC6006            2            2            1            2
```

```
> ## Access Rand index
> clus2$comparClus
```

```
            kmeans_onA kmeans_onS
1_kmeans_onA      1.000      0.531
1_kmeans_onS      0.531      1.000
2_kmeans_onA      1.000      0.779
2_kmeans_onS      0.779      1.000
```

Once a sample clustering has been computed, one can be interested in its association with the qualitative variables. Function `clusVarAnalysis` enables to perform the chi-square tests of independence to study the association between the clustering obtained on each component and the qualitative variables. It also draws the barplot to show the distribution of the variable levels across the clusters:

```
> ## Test the association between the clustering obtained by Mclust for the first
> ## component and the variables:
> clus2var <- clusVarAnalysis(icaSet=icaSetMainz[,,1:2], params=params,
+                             keepVar=c("er","grade"),
+                             resClus=clus1$clus, funClus="Mclust", adjustBy="none",
+                             doPlot=TRUE, path="clus2var/", filename="resChitests-Mcluscomp1")
> ## Look at the filename which contains p-values and links to the barplots
> ## p-values are also contained in the ouput of the function:
> clus2var
>
>
```

3.4.6 Comparison of *IcaSet* objects, function `runCompareIcaSets`

Visualization of the correspondence between independent components with correlation-based graphs We can study the association between ICs computed on n different datasets using

correlation graphs. In these graphs, each IC is represented as a node whose color indicates the dataset, and the edge thickness is proportional to the amount of correlation between the two ICs it links.
Hereafter we will denote by $C_{M,n}$ the n^{th} component from dataset M.

The relationship between the components is restricted to correlation maximum: an edge connecting a component $C_{A,i}$ to a component $C_{B,j}$ means that component $C_{A,i}$ is the most correlated component to $C_{B,j}$ among all the components $C_{A,i'(i'\neq i)}$ from the dataset A. The reciprocity of the link (i.e. the presence of an edge binding $C_{B,j}$ to $C_{A,i}$) reinforces the association between the two components.

In R, the graph can be visualized with function `tkplot`, using the "fruchterman.reingold" layout which attends to attribute the length of the edge according to one of its attribute, here the absolute correlation coefficient between the two components it links.
The edge thickness is also attributed according to the absolute correlation value (the higher the absolute correlation value is, the thicker the edge thickness is).
Highly reproducible components appear in the graph as a subset of n interconnected nodes of different colors (quasi-cliques). A way to highlight these quasi-cliques is obtained by coloring in black only edges linking reciprocal node pairs (a node pair is said to be reciprocal if there are edges between them in both directions). Non-reciprocal edges appear in grey. It allows to highlight the components with a high level of reproducibility.

Example: Comparison of four *IcaSet* objects As an example we will compare four ICA decompositions obtained on four different gene expression datasets of breast tumors (including the Mainz data used above).
We build an instance of *IcaSet* for each of the three datasets:

```
> ## load three other breast cancer datasets also based on  Affymetrix HG-U133a microarray
> library(breastCancerUPP)
> library(breastCancerTRANSBIG)
> library(breastCancerVDX)
> data(upp)
> data(transbig)
> data(vdx)
> ## function to build IcaSet instances from these three datasets
> treat <- function(es, annot="hgu133a.db") {
+     es <- selectFeatures_IQR(es,10000)
+     exprs(es) <- t(apply(exprs(es),1,scale,scale=FALSE))
+     colnames(exprs(es)) <- sampleNames(es)
+     resJade <- runICA(X=exprs(es), nbComp=5, method = "JADE", maxit=10000)
+     resBuild <- buildIcaSet(params=buildMineICAParams(), A=data.frame(resJade$A), S=data.fram
+                             dat=exprs(es), pData=pData(es), refSamples=character(0),
+                             annotation=annot, typeID= typeIDmainz,
+                             chipManu = "affymetrix", mart=mart)
+     icaSet <- resBuild$icaSet
+ }
```

```
> icaSetUpp <- treat(upp, annot="hgu133plus2.db")
> icaSetVdx <- treat(vdx)
> icaSetTransbig <- treat(transbig)
```

Each *IcaSet* was annotated at the gene level using Gene Symbols. We will therefore compute correlation between gene projection values stored in slot `SByGene` of each *IcaSet*.

Pearson correlation is used as a measure of association between the gene projections. The correlation graph can be build with function `runCompareIcaSets`:

```
> resGraph <- runCompareIcaSets(icaSets=list(icaSetMainz, icaSetUpp,
+                               icaSetTransbig, icaSetVdx),
+                               labAn=c("Mainz", "Upp","Transbig","Vdx"),
+                               type.corr="pearson", level="genes",
+                               cutoff_zval=0, fileNodeDescr="nodeDescr.txt",
+                               fileDataGraph="dataGraph.txt", tkplot=TRUE)
```

Get the colors attributed to each dataset using the element `nodeAttrs` of `resGraph`:

```
> barplot(names.arg=unique(resGraph[[2]]$labAn),height=rep(1,4),
+         col=unique(resGraph[[2]]$col))
```

The Mainz dataset is represented in blue. Three cliques of four components appear in the correlation-based graph, they include the first three components of the Mainz dataset. The latter are therefore reproducible across the four datasets and thus capture coexpression patterns shared across different breast cancer cohorts. We showed that the first component was associated with the cell cycle, the second with immune reaction, and the third one with epiderm development. The third component also included *EGFR* and several keratins among its contributing genes, and defined a cluster of samples constituted by a subset of the ER- breast tumors. The latter typically corresponds to the subtype of breast cancer known as "basal-like".

Here we chose to base the correlation on all genes. By modifying the argument `cutoff_zval`, we could have chosen to base the correlations on genes with contribution values higher than a given threshold. Using `cutoff_zval=1`, only the projections whose scaled values are not located within the circle of radius 1 when considering a pair of components are used to compute the correlation. In practice, the function will be much faster when `cutoff_zval=0`, since in that case pairs of components are not treated individually.

Created files nodeDescr.txt and dataGraph.txt may be used as inputs into Cytoscape **?** which could be a way to obtain a more elegant correlation graph.

Intersection and union between contributing genes When cliques appear in the correlation-based graph, you may want to compare the genes having high projections on the components included in the clique. The function `compareGenes` allows to compare components of different icaSets and returns the common genes ordered by their median rank across the components. Intersection or union of the genes can be considered.

We study the common contributing genes of the components included in two different cliques of the graph using `compareGenes`:

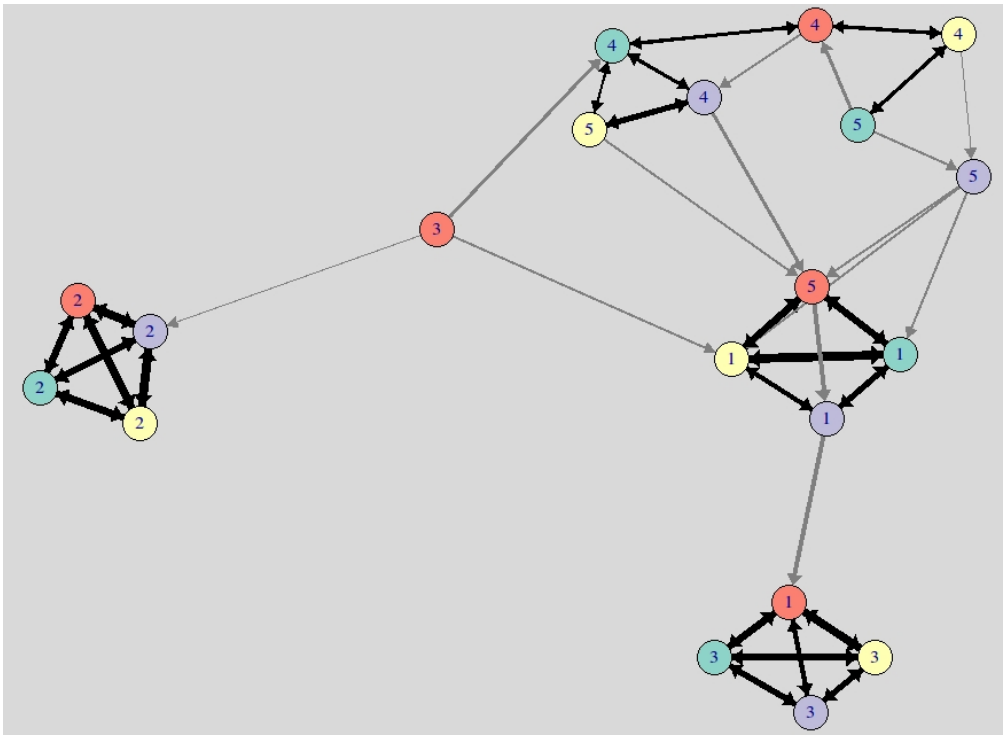


Figure 5: Correlation-based graph representing association between independent components obtained on four expression data of breast cancer samples. Each node denotes an IC and their colors represent the dataset they originate from. Edge thickness denotes the amount of correlation between the two ICs it links. Black edges denote reciprocal nodes.

```
> ## comparison of four components included in the clique of the correlation-based graph
> # that includes the second component of Mainz.
> inter <- compareGenes(keepCompByIcaSet =   c(2,2,2,2),
+                       icaSets = list(icaSetMainz, icaSetTransbig, icaSetUpp, icaSetVdx),
+                       lab=c("Mainz", "Transbig", "Upp", "Vdx"), cutoff=3,
+                       type="intersection",  annotate=F)
> head(inter)
```

	min_rank	median_rank	ranks	scaled_proj
IGL@	1	1.0	1,1,1,1	-8.9,-9.2,9.2,-9.3
IGKV4-1	2	2.0	2,2,3,2	-8.4,-8.7,7.9,-8.9
IGLV2-23	3	3.0	3,3,4,3	-7,-8.3,7.7,-8.8
NKG7	2	5.0	5,8,2,5	-6.4,-6.1,8.4,-7.6
IGHM	4	5.5	4,7,9,4	-6.6,-6.5,6.7,-7.7
TNFRSF17	4	6.0	6,4,6,6	-6.3,-7.8,7.1,-7.6

```
> ## comparison of four components included in the clique of the correlation-based graph
> # that includes the third component of Mainz.
> inter <- compareGenes(keepCompByIcaSet = c(3,3,3,1),
+                       icaSets = list(icaSetMainz, icaSetTransbig, icaSetUpp, icaSetVdx),
+                       lab=c("Mainz", "Transbig", "Upp", "Vdx"), cutoff=3,
+                       type="intersection",  annotate=F)
> head(inter)
```

	min_rank	median_rank	ranks	scaled_proj
GABRP	1	1.0	1,1,1,3	7.7,8.7,8.1,7.2
MIA	2	4.0	5,5,3,2	6.3,7.2,7.3,7.3
SERPINB5	5	6.5	7,7,6,5	6.2,7.1,5.9,6.5
KRT14	2	8.5	13,2,4,21	5.7,7.3,7,5.1
KRT16	2	9.5	2,15,15,4	7.3,5.8,5.4,7
KRT81	4	9.5	4,31,12,7	6.3,4.6,5.6,6.2

The common contributing genes of the first clique are strongly associated in the immune reaction and many of them are markers of lymphocytes, while the common contributing genes of the second clique include several keratins (*KRT5 KRT14, KRT15, KRT16, KRT17, KRT23, ...*) and other known markers of the basal-like breast subtype.