The DaMiRseq package - Data Mining for RNA-Seq data: normalization, feature selection and classification

Mattia Chiesa[1] ***and Luca Piacentini***[1]

[1]Immunology and Functional Genomics Unit, Centro Cardiologico Monzino, IRCCS, Milan, Italy;

December 12, 2017

Abstract

RNA-Seq is increasingly the method of choice for researchers studying the transcriptome. The strategies to analyze such complex high-dimensional data rely on data mining and statistical learning techniques. The *DaMiRseq* package offers a tidy pipeline that includes data mining procedures for data handling and implementation of prediction learning methods to build classification models. The package accepts any kind of data presented as a table of raw counts and allows the inclusion of variables that occur with the experimental setting. A series of functions enables data cleaning by filtering genomic features and samples, data adjustment by identifying and removing the unwanted source of variation (*i.e.* batches and confounding factors) and to select the best predictors for modeling. Finally, a "Stacking" ensemble learning technique is applied to build a robust classification model. Every step includes a checkpoint for assessing the effects of data management using diagnostic plots, such as clustering and heatmaps, RLE boxplots, MDS or correlation plots.

Package

DaMiRseq 1.3.1

Contents

1 Introduction . . . 3

2 Workflow . . . 4

2.1 Input data . . . 4

2.2 Import Data . . . 4

2.3 Preprocessing and Normalization . . . 5

2.3.1 Filtering by Expression . . . 6

2.3.2 Filtering By Coefficient of Variation (CV) . . . 6

2.3.3 Normalization . . . 7

2.3.4 Sample Filtering . . . 7

2.4 Adjusting Data . . . 8

2.4.1 Identification of Surrogate Variables . . . 8

2.4.2 Correlation between sv and known covariates . . . 9

2.4.3 Cleaning expression data . . . 10

2.5 Exploring Data . . . 11

2.6 Feature Selection . . . 19

2.6.1 Variable selection in Partial Least Squares (PLS) . . . 19

2.6.2 Removing highly correlated features . . . 19

2.6.3 Ranking and selecting most relevant features . . . 20

2.7 Classification . . . 24

2.8 Exporting output data . . . 26

3 Adjusting the data: a necessary step? . . . 26

4 New implementations . . . 29

4.1 Version 1.3.1 . . . 29

5 Session Info . . . 30

1 Introduction

RNA-Seq is a powerful high-throughput assay that uses next-generation sequencing (NGS) technologies to profile, discover and quantify RNAs. The whole collection of RNAs defines the transcriptome, whose plasticity, allows the researcher to capture important biological information: the transcriptome, in fact, is sensitive to changes occurring in response to environmental challenges, different healthy/disease state or specific genetic/epigenetic context. The high-dimensional nature of NGS makes the analysis of RNA-Seq data a demanding task that the researcher may tackle by using data mining and statistical learning procedures. Data mining usually exploits iterative and interactive processes that include, preprocessing, transforming and selecting data so that only relevant features are efficiently used by learning methods to build classification models.
Many software packages have been developed to assess differential expression of genomic features (i.e. genes, transcripts, exons etc.) of RNA-seq data. (see Bioconductor_RNASeq-packages). Here, we propose the *DaMiRseq* package that offers a systematic and organized analysis workflow to face classification problems.
Briefly, we summarize the **philosophy of *DaMiRseq*** as follows. The pipeline has been thought to direct the user, through a step-by-step data evaluation, to properly select the best strategy for each specific classification setting. It is structured into three main parts: (1) *normalization*, (2) *feature selection*, and (3) *classification*. The package can be used with any technology that produces read counts of genomic features.

The normalization step integrates conventional preprocessing and normalization procedures with data adjustment based on the estimation of the effect of "unwanted variation". Several factors of interest such as environments, phenotypes, demographic or clinical outcomes may influence the expression of the genomic features. Besides, an additional unknown source of variation may also affect the expression of any particular genomic feature and lead to confounding results and inaccurate data interpretation. The estimation of these unmeasured factors, also known as surrogate variables (sv), is crucial to fine-tune expression data in order to gain accurate prediction models [1, 2].

RNA-Seq usually consists of many features that are either irrelevant or redundant for classification purposes. Once an expression matrix of n features x m observations is normalized and corrected for confounding factors, the pipeline provides methods to help the user to reduce and select a subset of n that will be subsequently used to build the prediction models. This approach, which exploits the so-called "Feature Selection" techniques, presents clear benefits since: it (1) limits overfitting, (2) improves classification performance of predictors, (3) reduces time training processing, and (4) allows the production of more cost-effective models [3, 4].

The reduced expression matrix, consisting of the most informative variables with respect to class, is then used to draw a "meta-learner" by combining different classifiers: Random Forest (RF), Naïve Bayes (NB), 3-Nearest Neighbours (3kNN), Logistic Regression (LR), Linear Discriminant Analysis (LDA), Support Vectors Machines (SVM), Neural Networks (NN) and Partial Least Squares (PLS); this method may be referred to as a "Stack Generalization" or, simply, "Stacking" ensemble learning technique [5]. The idea behind this method is that "weaker" classifiers may have different generalization performances, leading to future misclassifications; by contrast, combining and weighting the prediction of several classifiers may reduce the risk of classification errors [6, 7]. Moreover, the weighted voting method, used to assess the goodness of each weak classifiers, allows meta-learner to reach consistently high classification accuracies, better than or comparable with best weak classifiers [8].

2 Workflow

2.1 Input data

DaMiRseq expects as input two kind of data:

- **Raw counts Data** - They have to be in the classical form of a n x m expression table of integer values coming from a RNA-Seq experiment: each row represents a genomic feature (n) while each column represents a sample (m). The expression values must be un-normalized raw read counts, since *DaMiRseq* implements normalization and transformation procedures of raw counts; the RNA-seq workflow in Bioconductor describes several techniques for preparing count matrices. Unique identifiers are needed for both genomic features and samples.
- **Class and variables Information** - This file contains the information related to classes/conditions (mandatory) and to known variables (optional), such as demographic or clinical data, biological context/variables and any sequencing or technical details. **The column containing the class/condition information must be labelled 'class'**. In this table, each row represents a sample and each column represents a variable (class/condition and factorial and/or continuous variables). Rows and identifiers must correspond to columns in 'Raw Counts Data' table.

In this vignette we describe the *DaMiRseq* pipeline, using as sample data a subset of Genotype-Tissue Expression (GTEx) RNA-Seq database (dbGap Study Accession: phs000424.v6.p1) [9]. Briefly, GTEx project includes the mRNA sequencing data of 53 tissues from 544 *post-mortem* donors, using 76 bp paired-end technique on Illumina HiSeq 2000: overall, 8555 samples were analyzed. Here, we extracted data and some additional sample information (*i.e.* sex, age, collection center and death classification based on the Hardy scale) for two similar brain subregions: Anterior Cingulate Cortex (Bromann Area 24) and Frontal Cortex (Brodmann Area 9). These areas are close to each other and are deemed to be involved in decision making as well as in learning. This dataset is composed of 192 samples: 84 Anterior Cingulate Cortex (ACC) and 108 Frontal Cortex (FC) samples for 56318 genes.
We, also, provide a data frame with classes and variables included.

2.2 Import Data

DaMiRseq package uses data extracted from *SummarizedExperiment* class object. This object is usually employed to store either expression data produced by high-troughput technology and other information occuring with the experimental setting. The `SummarizedExperiment` object may be considered a matrix-like holder where rows and colums represent, respectively, features and samples. If data are not stored in a `SummarizedExperiment` object, the `DaMiRseq.makeSE` function helps the user to build a `SummarizedExperiment` object starting from expression and variable data table. The function tests if expression data are in the form of raw counts, *i.e.* positive integer numbers, if 'class' variable is included in the data frame and if "NAs" are present in either the counts and the variables table. The `DaMiRseq.makeSE` function needs two files as input data: 1) a raw counts table and 2) a class and (if present) variable information table. In this vignette, we will use the dataset described in Section 2.1 but the user could import other count and variable table files into R environment as follows:

```
library(DaMiRseq)
## only for example:
# rawdata.path <- system.file(package = "DaMiRseq","extdata")
# setwd(rawdata.path)
# filecounts <- list.files(rawdata.path, full.names = TRUE)[1]
# filecovariates <- list.files(rawdata.path, full.names = TRUE)[2]
# count_data <- read.delim(filecounts)
# covariate_data <- read.delim(filecovariates)
# SE<-DaMiR.makeSE(count_data, covariate_data)
```

Here, we load by the `data()` function a prefiltered sample expression data of the GTEx RNA-Seq database made of 21363 genes and 40 samples (20 ACC and 20 FC):

```
data(SE)
assay(SE)[1:5, c(1:5, 21:25)]
```

```
##                 ACC_1 ACC_2 ACC_3 ACC_4 ACC_5 FC_1 FC_2 FC_3 FC_4 FC_5
## ENSG00000227232   327   491   226   285  1011  465  385  395  219  398
## ENSG00000237683   184    57    35    57   138  290  293   93   84  145
## ENSG00000268903    29    15     7    26    33   84   39   22   31   39
## ENSG00000241860    25    12     6     5    26    6   17   13    4   12
## ENSG00000228463   248   126    99    76   172  170  173  157   95  150
```

```
colData(SE)
```

```
## DataFrame with 40 rows and 5 columns
##         center      sex      age     death    class
##       <factor> <factor> <factor> <integer> <factor>
## ACC_1   B1, A1        M    60-69         2      ACC
## ACC_2   B1, A1        F    40-49         3      ACC
## ACC_3   B1, A1        F    60-69         2      ACC
## ACC_4   B1, A1        F    50-59         2      ACC
## ACC_5   C1, A1        M    50-59         2      ACC
## ...        ...      ...      ...       ...      ...
## FC_16   C1, A1        M    60-69         2       FC
## FC_17   B1, A1        M    60-69         2       FC
## FC_18   C1, A1        F    50-59         2       FC
## FC_19   B1, A1        M    50-59         2       FC
## FC_20   C1, A1        F    50-59         4       FC
```

Data are stored in the `SE` object of class *SummarizedExperiment*. Expression and variable information data may be retrieved, respectively, by the `assay()` and `colData()` accessor functions [1]. The *"colData(SE)"* data frame, containing the variables information, includes also the ***'class'*** column (mandatory) as reported in the Reference Manual.

[1] See *SummarizedExperiment* [10], for more details.

2.3 Preprocessing and Normalization

After importing the counts data, we ought to filter out non-expressed and/or highly variant, inconsistent genes and, then, perform normalization. Furthermore, the user can also decide to exclude from the dataset samples that show a low correlation among biological replicates and,

thus, may be suspected to hold some technical artifact. The `DaMiR.normalization` function helps solving the first issues, while `DaMiR.sampleFilt` allows the removal of inconsistent samples.

2.3.1 Filtering by Expression

Users can remove genes, setting up the minimum number of read counts permitted across samples:

```
data_norm <- DaMiR.normalization(SE, minCounts=10, fSample=0.7,
                                 hyper = "no")

## 2007 Features have been filtered out by espression. 19356 Features remained.
## Performing Normalization by 'vst' with dispersion parameter:  parametric
```

In this case, 19066 genes with read counts greater than 10 (`minCounts = 10`) in at least 70% of samples (`fSample = 0.7`), have been selected, while 2297 have been filtered out. The dataset, consisting now of 19066 genes, is then normalized by the `varianceStabilizing Transformation` function of the *DESeq2* package [11]. Using `assay()` function, we can see that "VST" transformation produces data on the log2 scale normalized with respect to the library size.

2.3.2 Filtering By Coefficient of Variation (CV)

We named "hypervariants" those genes that present anomalous read counts, by comparing to the mean value across the samples. We identify them by calculating distinct CV on sample sets that belong to each 'class'. Genes with all 'class' CV greater than `th.cv` are discarded.
Note. Computing a 'class' restricted CV may prevent the removal of features that may be specifically associated with a certain class. This could be important in some biological contexts, such as immune genes whose expression under definite conditions may unveil peculiar class-gene associations.
This time, we run again the `DaMiR.normalization` function by enabling the "hypervariant" gene detection by setting `hyper = "yes"` and `th.cv=3` (default):

```
data_norm <- DaMiR.normalization(SE, minCounts=10, fSample=0.7,
                                 hyper = "yes", th.cv=3)

## 2007 Features have been filtered out by espression. 19356 Features remained.
## 13 'Hypervariant' Features have been filtered out. 19343 Features remained.
## Performing Normalization by 'vst' with dispersion parameter:  parametric

print(data_norm)

## class: SummarizedExperiment
## dim: 19343 40
## metadata(0):
## assays(1): ''
## rownames(19343): ENSG00000227232 ENSG00000237683 ... ENSG00000198695
##   ENSG00000198727
## rowData names(0):
## colnames(40): ACC_1 ACC_2 ... FC_19 FC_20
## colData names(5): center sex age death class
```

```
assay(data_norm)[c(1:5), c(1:5, 21:25)]
```

```
##                    ACC_1    ACC_2    ACC_3    ACC_4    ACC_5     FC_1
## ENSG00000227232 8.199283 9.353454 8.759033 8.452277 9.142766 8.885701
## ENSG00000237683 7.457537 6.592786 6.435384 6.466654 6.603629 8.248528
## ENSG00000268903 5.508724 5.337338 5.043412 5.693841 5.273067 6.708270
## ENSG00000228463 7.837276 7.537126 7.667278 6.787960 6.854035 7.555618
## ENSG00000241670 5.420923 5.687098 6.086880 5.624320 5.227604 5.640934
##                     FC_2     FC_3     FC_4     FC_5
## ENSG00000227232 8.377586 8.946374 8.420556 8.799498
## ENSG00000237683 8.016036 7.068686 7.189774 7.471898
## ENSG00000268903 5.737260 5.582227 6.083030 5.997092
## ENSG00000228463 7.344416 7.718235 7.340514 7.514437
## ENSG00000241670 4.659126 4.792983 5.264038 5.752181
```

The `th.cv = 3` allows the removal of a further 14 "hypervariant" genes from the gene expression data matrix. The number of genes is now reduced to 19052.

2.3.3 Normalization

After filtering, a normalization step is performed; two normalization methods are embedded in *DaMiRseq*: the *Variance Stabilizing Transformation* (VST) and the *Regularized Log Transformation* (rlog). As described in the *DESeq2* vignette, VST and rlog have similar effects on data but the VST is faster than rlog, expecially when the number of samples increases; for these reasons, `varianceStabilizingTransformation` is the default normalization method, while `rlog` can be, alternatively, chosen by user.

```
# Time Difference, using VST or rlog for normalization:
#
#data_norm <- DaMiR.normalization(dds, minCounts=10, fSample=0.7, th.cv=3)
# VST: about 80 seconds
#
#data_norm <- DaMiR.normalization(dds, minCounts=10, fSample=0.7, th.cv=3,
#                                 type="rlog")
# rlog: about 8890 seconds (i.e. 2 hours and 28 minutes!)
```

In this example, we run `DaMiR.normalization` function twice, just modifying `type` arguments in order to test the processing time; with `type = "vst"` (default - the same parameters used in Section 2.3.2) `DaMiR.normalization` needed 80 seconds to complete filtering and normalization, while with `type = "rlog"` required more than 2 hours. Data were obtained on a workstation with an esa core CPU (2.40 GHz, 16 GB RAM) and 64-bit Operating System.

2.3.4 Sample Filtering

This step introduces a sample quality checkpoint. The assumption is that global gene expression should exhibit high correlation among biological replicates; conversely, low correlated samples may be suspected to hold some technical artifacts (*e.g.* poor RNA quality or library preparation), despite pass sequencing quality controls. If not identified and removed, these samples may negatively affect the entire downstream analysis. `DaMiR.sampleFilt` assesses

the mean absolute correlation of each sample and removes those samples with a correlation lower than the value set in `th.corr` argument. This threshold may be specific for different experimental settings but should be as high as possible.

```
data_filt <- DaMiR.sampleFilt(data_norm, th.corr=0.9)

## 0 Samples have been excluded by averaged Sample-per-Sample correlation.
##  40 Samples remained.

dim(data_filt)

## [1] 19343    40
```

In this study case, zero samples were discarded because their mean absolute correlation is higher than 0.9. Data were stored in a `SummarizedExperiment` object, which contains a normalized and filtered expression *matrix* and an updated `DataFrame` with the variables of interest.

2.4 Adjusting Data

After data normalization, we propose to test for the presence of surrogate variables (sv) in order to remove the effect of putative confounding factors from the expression data. The algorithm cannot distinguish among real technical batches and important biological effects (such as environmental, genetic or demographic variables) whose correction is not desirable. Therefore, we enable the user to evaluate whether any of the retrieved sv is correlated or not with one or more known variables. Thus, this step gives the user the opportunity to choose the most appropriate number of sv to be used for expression data adjustment [1, 2].

2.4.1 Identification of Surrogate Variables

Surrogate variables identification, basically, relies on the SVA algorithm by Leek et al. [12] [2]. A novel method, which allows the identification of the the maximum number of sv to be used for data adjustment, has been introduced in our package. Specifically, we compute eigenvalues of data and calculate the squares of each eigenvalues. The ratio of each "squared eigenvalue" to the sum of them were then calculated. These values represent a surrogate measure of the "Fraction of Explained Variance" (fve) that we would obtain by principal component analysis (PCA). Their cumulative sum can be, finally, used to select sv. The method to be applied can be selected in the `method` argument of the `DaMiR.SV` function. The option `"fve"`, `"be"` and `"leek"` selects, respectively, our implementation or one of the two methods proposed in the *sva* package.

[2] See *sva* package

```
 sv <- DaMiR.SV(data_filt)

## The number of SVs identified, which explain 95 % of Variance, is: 5
```

Using default values (`"fve"` method and `th.fve = 0.95`), we obtained a matrix with 4 sv that is the number of sv which returns 95% of variance explained. Figure 1 shows all the sv computed by the algorithm with respect to the corresponding fraction of variance explained.

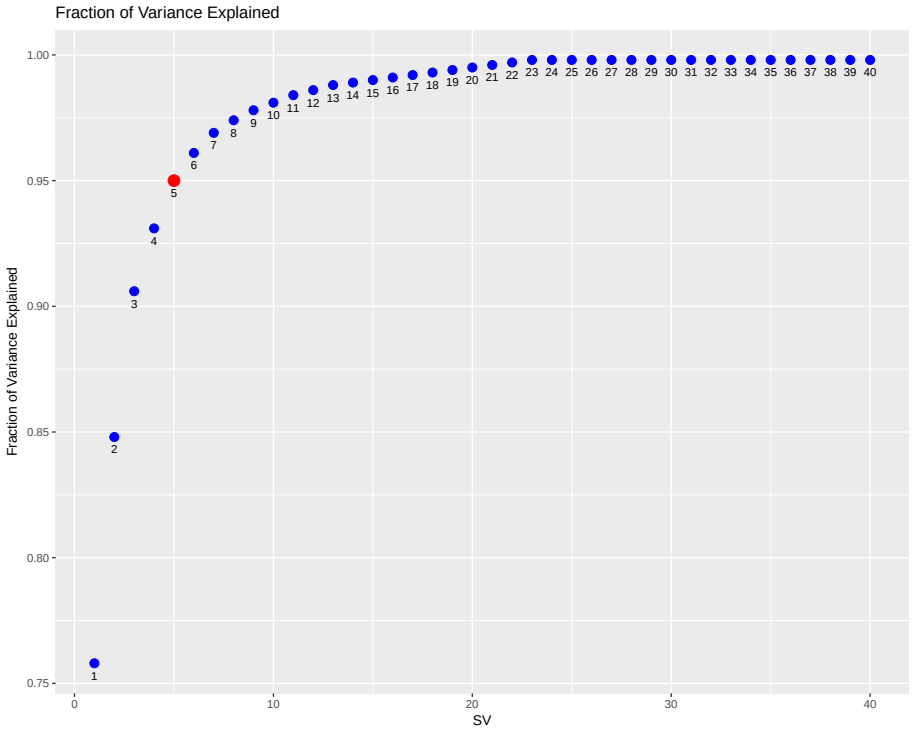

Figure 1: Fraction of Variance Explained
This plot shows the relationship between each identified sv and the corresponding fraction of variance explained. A specific blue dot represents the proportion of variance, explained by a sv together with the prior ones. The red dot marks the upper limit of sv that should be used to adjust the data. Here, 4 is the maximum number of sv obtained as it corresponds to $\leq$ 95% of variance explained.

2.4.2 Correlation between sv and known covariates

Once the sv have been calculated, we may inquire whether these sv capture an unwanted source of variation or may be associated with known variables that the user does not wish to correct. For this purpose, we correlate the sv with the known variables stored in the "data_filt" object, to decide if all of these sv or only a subset of them should be used to adjust the data.

```
DaMiR.corrplot(sv, colData(data_filt), sig.level = 0.01)
```

The `DaMiR.corrplot` function produces a correlation plot where significant correlations (in the example the threshold is set to `sig.level = 0.01`) are shown within colored circles (blue or red gradient). In Figure 2, we can see that the first three sv do not significantly correlate with any of the used variables and, presumably, recovers the effect of unmeasured variables. The fourth sv presents, instead, a significant correlation with the "center" variable. The effect of "center" might be considered a batch effect and we are interested in adjusting the data for a such confounding factor.
Note. The correlation with "class" should always be non significant. In fact, the algorithm for sv identification (embedded into the `DaMiR.SV` function) decomposes the expression variation with respect to the variable of interest (*e.g.* class), that is what we want to preserve by correction [1]. Conversely, the user should consider the possibility that hidden factors may present a certain association with the 'class' variable. In this case, we suggest not to remove the effect of these sv so that any overcorrection of the expression data is avoided.

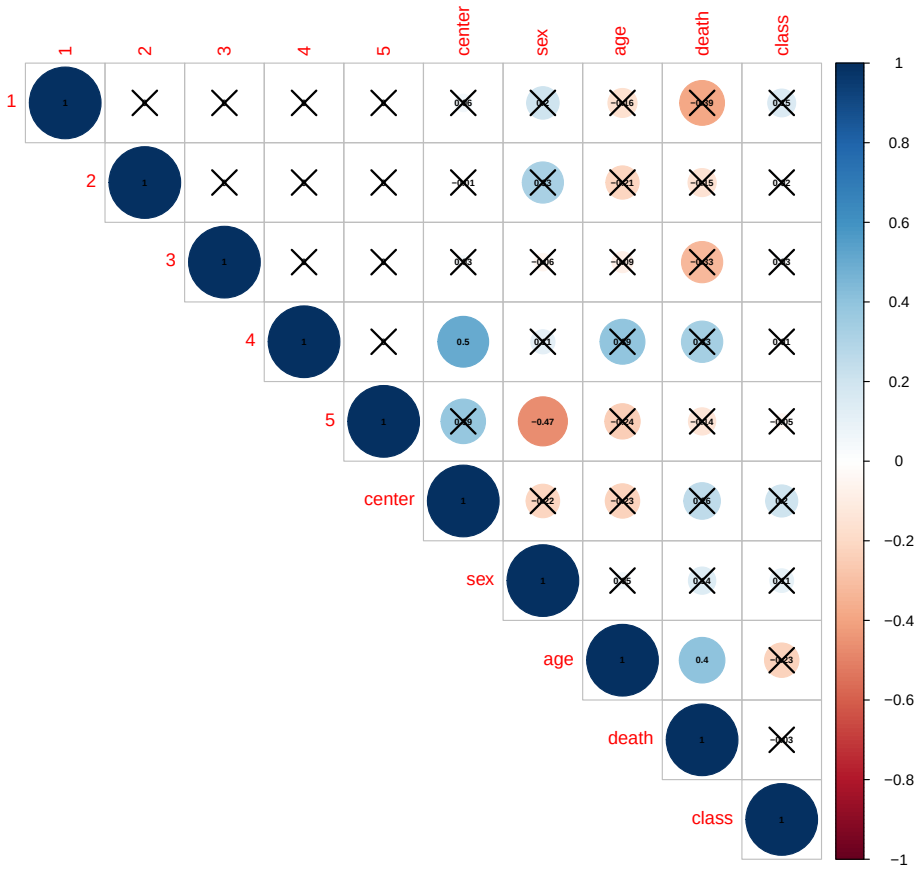

Figure 2: Correlation Plot between sv and known variables
This plot highligths the correlation between sv and known covariates, using both color gradient and circle size. The color ranges from dark red (correlation = -1) to dark blue (correlation = 1) and the circle size is maximum for a correlation equal to 1 or -1 and decreases up to zero. Black crosses help to identify non-significant correlations. This plot shows that the first to the third sv do not significantly correlate with any variable, while the fourth is significantly correlated with the "center" variable.

2.4.3 Cleaning expression data

After sv identification, we need to adjust our expression data. To do this, we exploited the `removeBatchEffect` function of the *limma* package which is useful for removing unwanted effects from the expression data matrix [13]. Thus, for the case study, we adjusted our expression data by setting `n.sv = 4` which instructs the algorithm to use the 4 surrogate variables taken from the sv matrix, produced by `DaMiR.SV` function (see Section 2.4.1).

```
data_adjust<-DaMiR.SVadjust(data_filt, sv, n.sv=4)
assay(data_adjust[c(1:5), c(1:5, 21:25)])
```

```
##                   ACC_1    ACC_2    ACC_3    ACC_4    ACC_5     FC_1
## ENSG00000227232 8.290885 9.516298 8.863302 8.347377 9.023696 8.701106
## ENSG00000237683 7.397286 7.393545 6.596542 6.860804 6.546196 7.809301
## ENSG00000268903 5.652611 5.828845 5.074154 5.679763 5.321159 6.246796
## ENSG00000228463 7.762130 7.472577 7.599897 6.909466 6.900161 7.484543
## ENSG00000241670 5.565704 5.650219 6.048199 5.394274 5.276316 5.566595
##                   FC_2     FC_3     FC_4     FC_5
```

```
## ENSG00000227232 8.492597 8.939917 8.535647 9.126107
## ENSG00000237683 7.904156 7.024480 6.999900 7.093168
## ENSG00000268903 5.769680 5.577975 5.704665 5.483088
## ENSG00000228463 7.236965 7.593969 7.512370 7.713632
## ENSG00000241670 4.762891 4.847959 5.047871 5.521930
```

Now, 'data_adjust' object contains a numeric matrix of log2-expression values with sv effects removed.

2.5 Exploring Data

Quality Control (QC) is an essential part of any data analysis workflow, because it allows checking the effects of each action, such as filtering, normalization, and data cleaning. In this context, the function `DaMiR.Allplot` helps identifying how different arguments or specific tasks, such as filtering or normalization, affect the data. Several diagnostic plots are generated:

Heatmap - A distance matrix, based on sample-by-sample correlation, is represented by heatmap and dendrogram using *pheatmap* package. In addition to 'class', all covariates are shown, using color codes; this helps to simultaneously identify outlier samples and specific clusters, related with class or other variables;

MultiDimensional Scaling (MDS) plots - MDS plot, drawn by *ggplot2* package [14], provides a visual representation of pattern of proximities (*e.g.* similarities or distances) among a set of samples, and allows the identification of natural clusters. For the 'class' and for each variable a MDS plot is drawn.

Relative Log Expression (RLE) boxplot - This plot, drawn by *EDASeq* package [15], helps to visualize the differences between the distributions across samples: medians of each RLE boxplot should be ideally centered around zero and a large shift from zero suggests that samples could have quality problems. Here, different colors means different classes.

In this vignette, `DaMiR.Allplot` is used to appreciate the effect of data adjusting (see Section 2.4). First, we check how data appear just after normalization: the heatmap and RLE plot in Figure 3 (upper and lower panel, respectively) and MDS plots in Figures 4 and 5 do not highlight the presence of specific clusters.
Note. If a variable contains missing data (i.e. "NA" values), the function cannot draw the plot showing variable information. The user is, however, encouraged to impute missing data if s/he considers it meaningful to plot the covariate of interest for "diagnosis" purposes.

```
# After gene filtering and normalization
DaMiR.Allplot(data_filt, colData(data_filt))
```

The `df` argument has been supplied using `colData()` function that returns the data frame of covariates stored into the "data_filt" object. Here, we used all the variables included into the data frame (*e.g.* center, sex, age, death and class), although it is possible to use only a subset of them to be plotted.

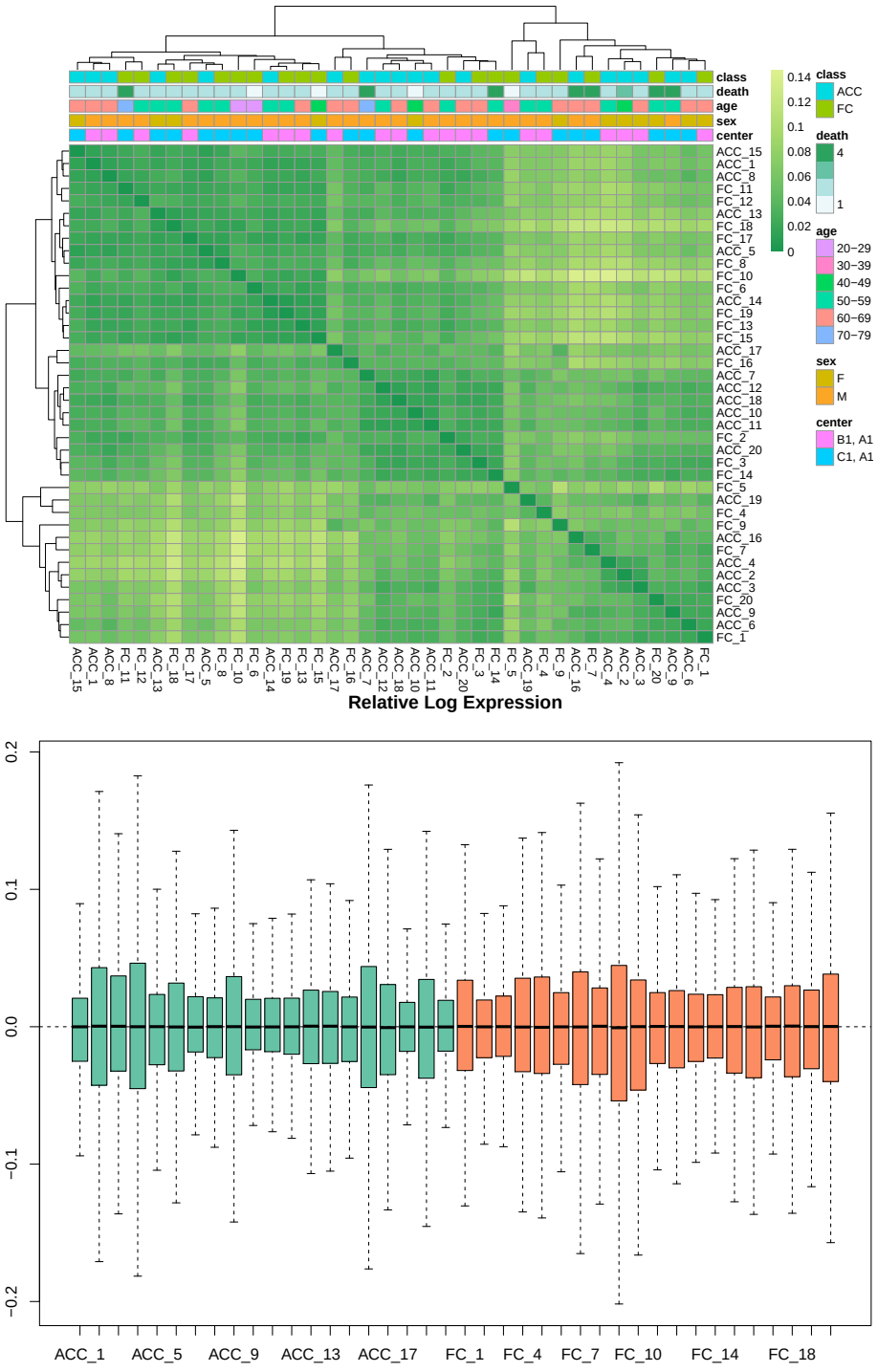

Figure 3: Heatmap and RLE

Heatmap (upper panel): colors in heatmap highlight the distance matrix, obtained by Spearman's correlation metric: color gradient ranges from *dark green*, meaning 'minimum distance' (*i.e.* dissimilarity = 0, correlation = 1), to *light green green*. On the top of heatmap, horizontal bars represent class and covariates. Each variable is differently colored (see legend). On the top and on the left side of the heatmap the dendrograms are drawn. Clusters can be easily identified.

RLE (lower panel): a boxplot of the distribution of expression values computed as the difference between the expression of each gene and the median expression of that gene accross all samples. Here, since all medians are very close to zero, it appears that all the samples are well-normalized and do not present any quality problems.

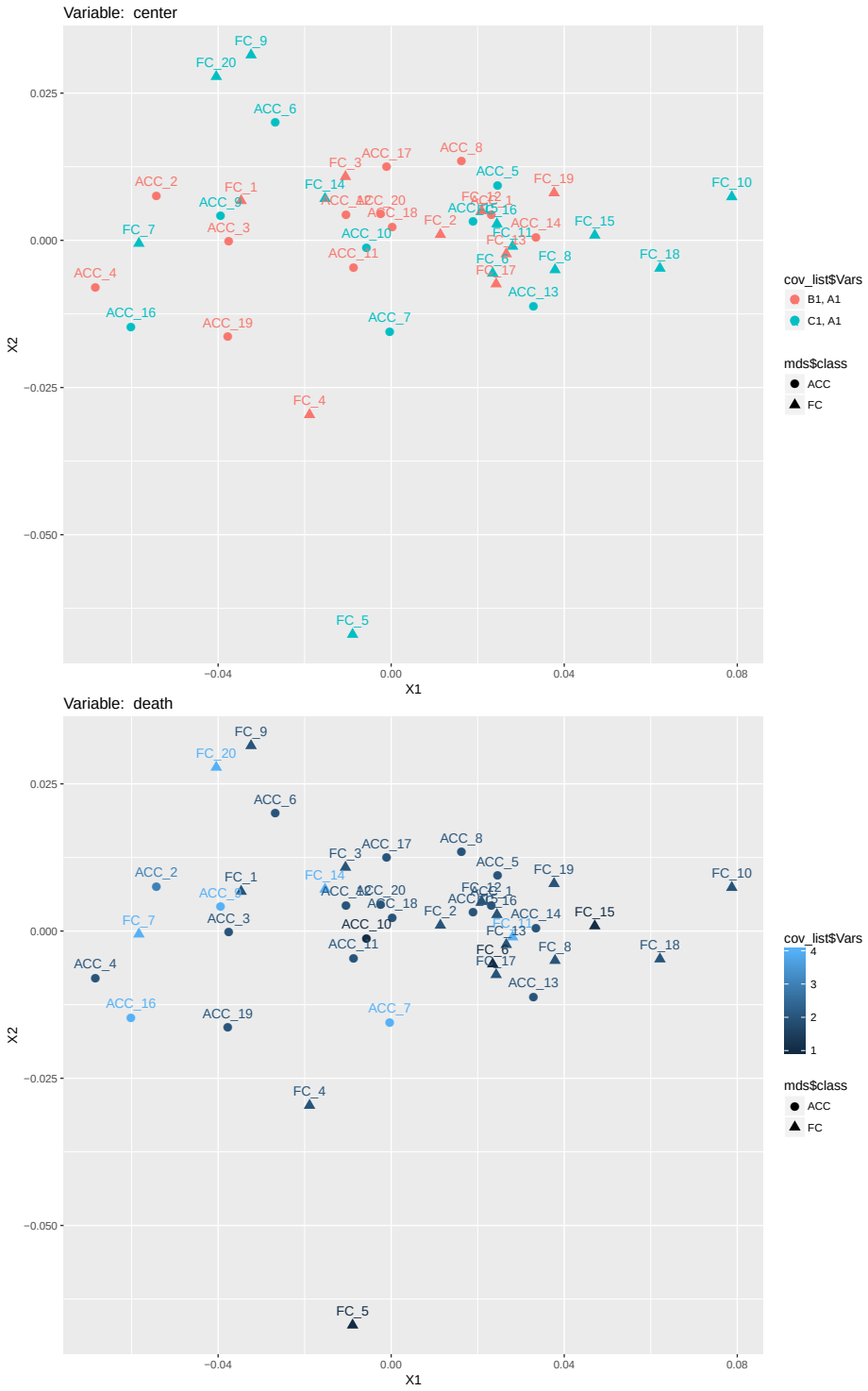

Figure 4: **MultiDimentional Scaling plot**
An unsupervised MDS plot is drawn. Samples are colored according to the 'Hardy death scale' (upper panel) and the 'center' variable (lower panel).

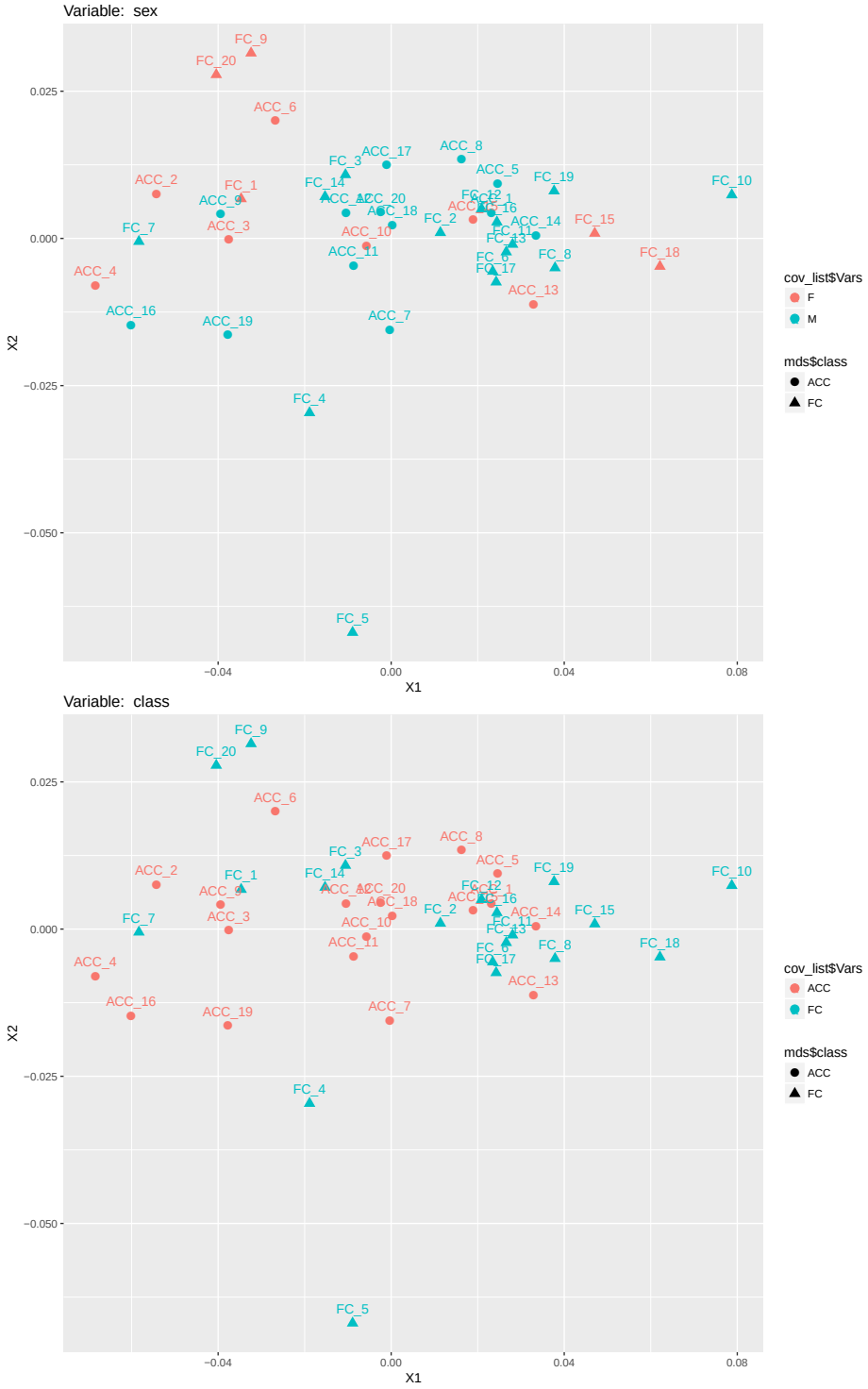

Figure 5: MultiDimentional Scaling plot
An unsupervised MDS plot is drawn. Samples are colored according to 'sex' variable (upper panel) and 'class' (lower panel).

After removing the effect of "noise" from our expression data, as presented in Section 2.4, we may appreciate the result of data adjustiment for sv: now, the heatmap in Figure 6 and MDS plots in Figures 7 and 8 exhibit specific clusters related to *'class'* variable. Moreover, the effect on data distribution is irrelevant: both RLE in Figures 3 and 6 show minimal shifts from the zero line, whereas RLE of adjusted data displays lower dispersion.

```
# After sample filtering and sv adjusting
DaMiR.Allplot(data_adjust, colData(data_adjust))
```

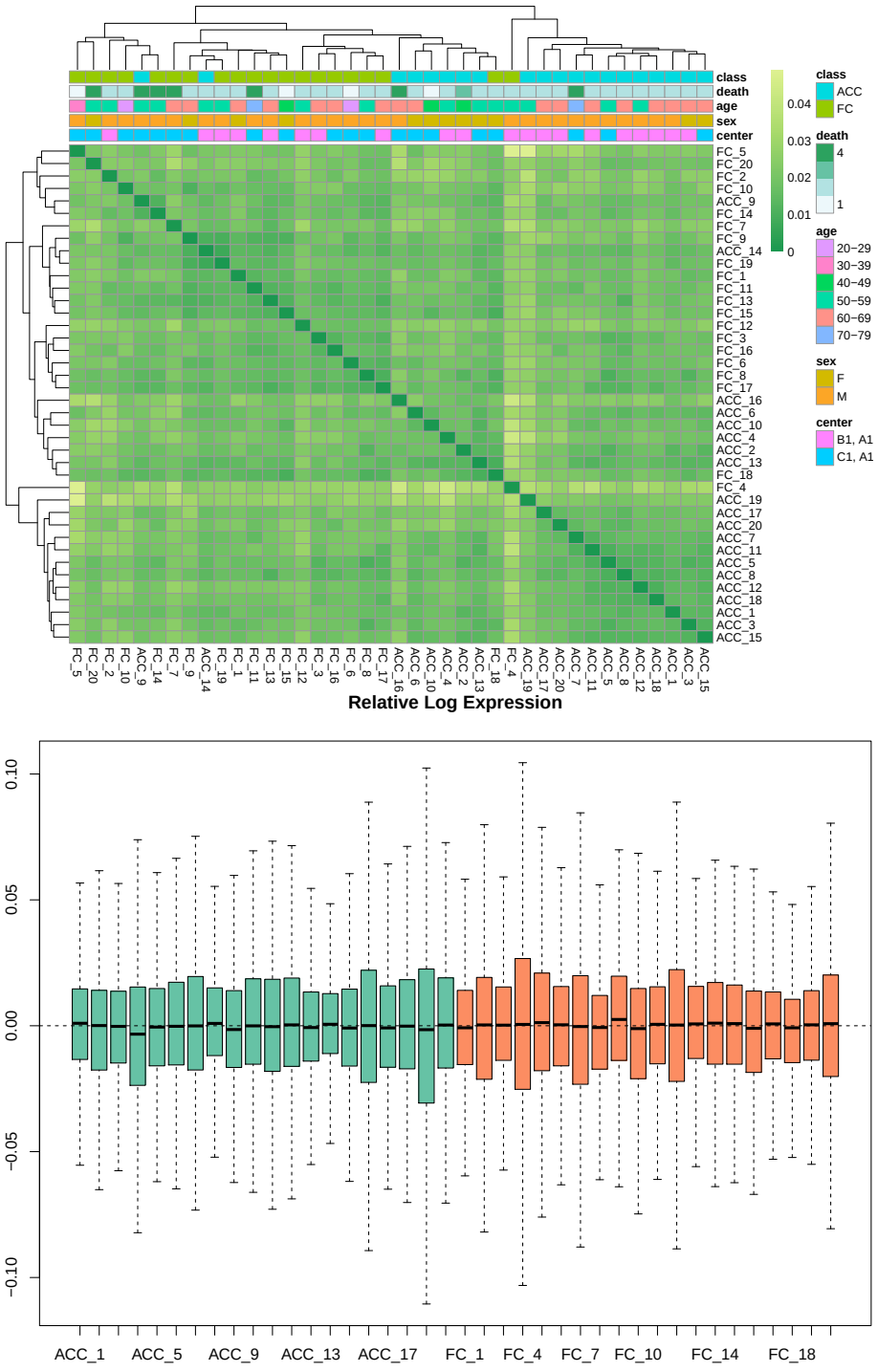

Figure 6: Heatmap and RLE
Heatmap (upper panel): colors in heatmap highlight the distance matrix, obtained by Spearman's correlation metric: color gradient ranges from *dark green*, meaning 'minimum distance' (*i.e.* dissimilarity = 0, correlation = 1), to *light green green*. On the top of heatmap, horizontal bars represent class and variables. Each variable is differently colored (see legend). The two dendrograms help to quickly identify clusters. RLE (lower panel): Relative Log Expression boxplot. A boxplot of the distribution of expression values computed as the difference between the expression of each gene and the median expression of that gene accross all samples is shown. Here, all medians are very close to zero, meaning that samples are well-normalized.

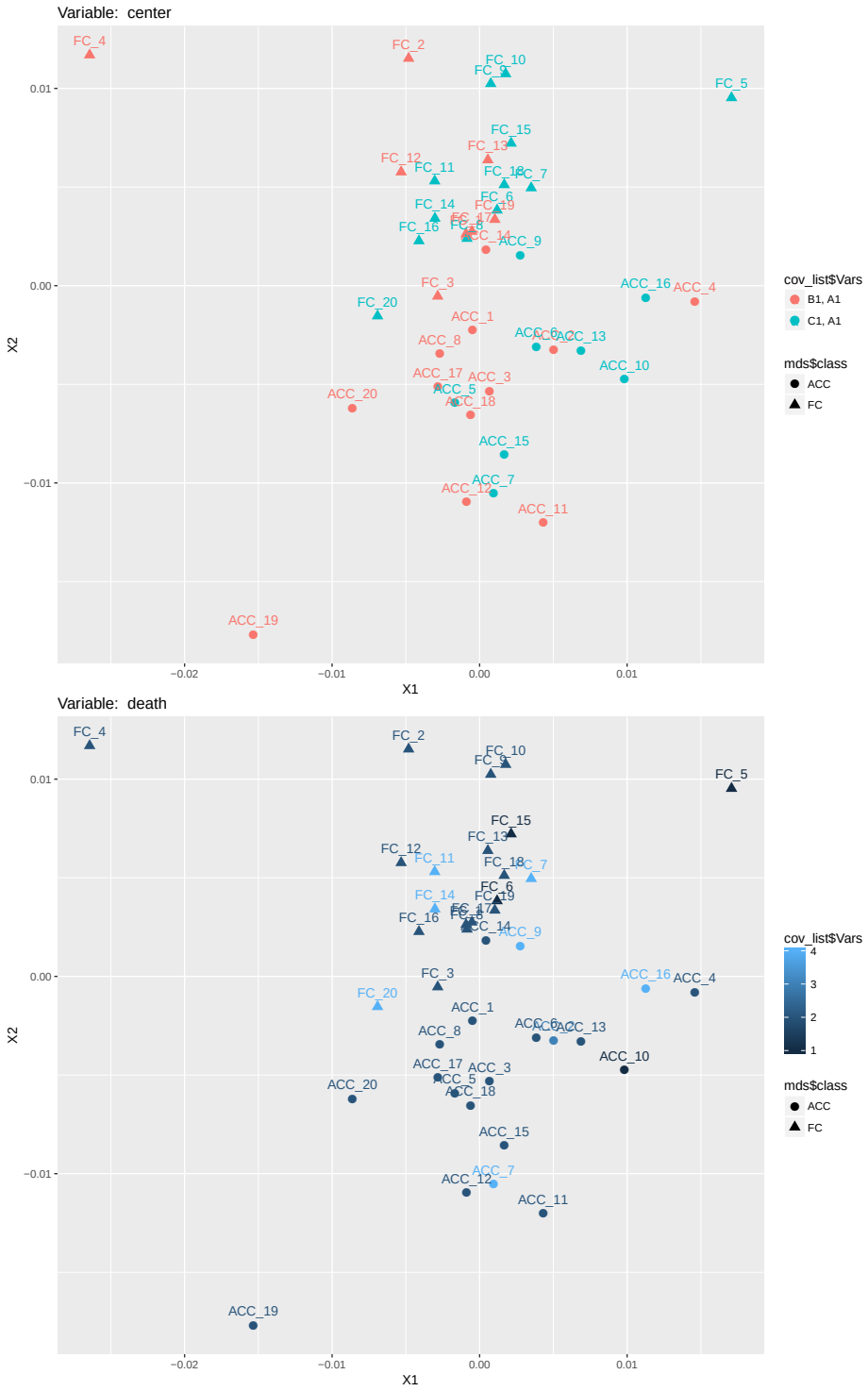

Figure 7: MultiDimentional Scaling plot
An unsupervised MDS plot is drawn. Samples are colored according to the 'Hardy death scale' (upper panel) and the 'center' variable (lower panel).

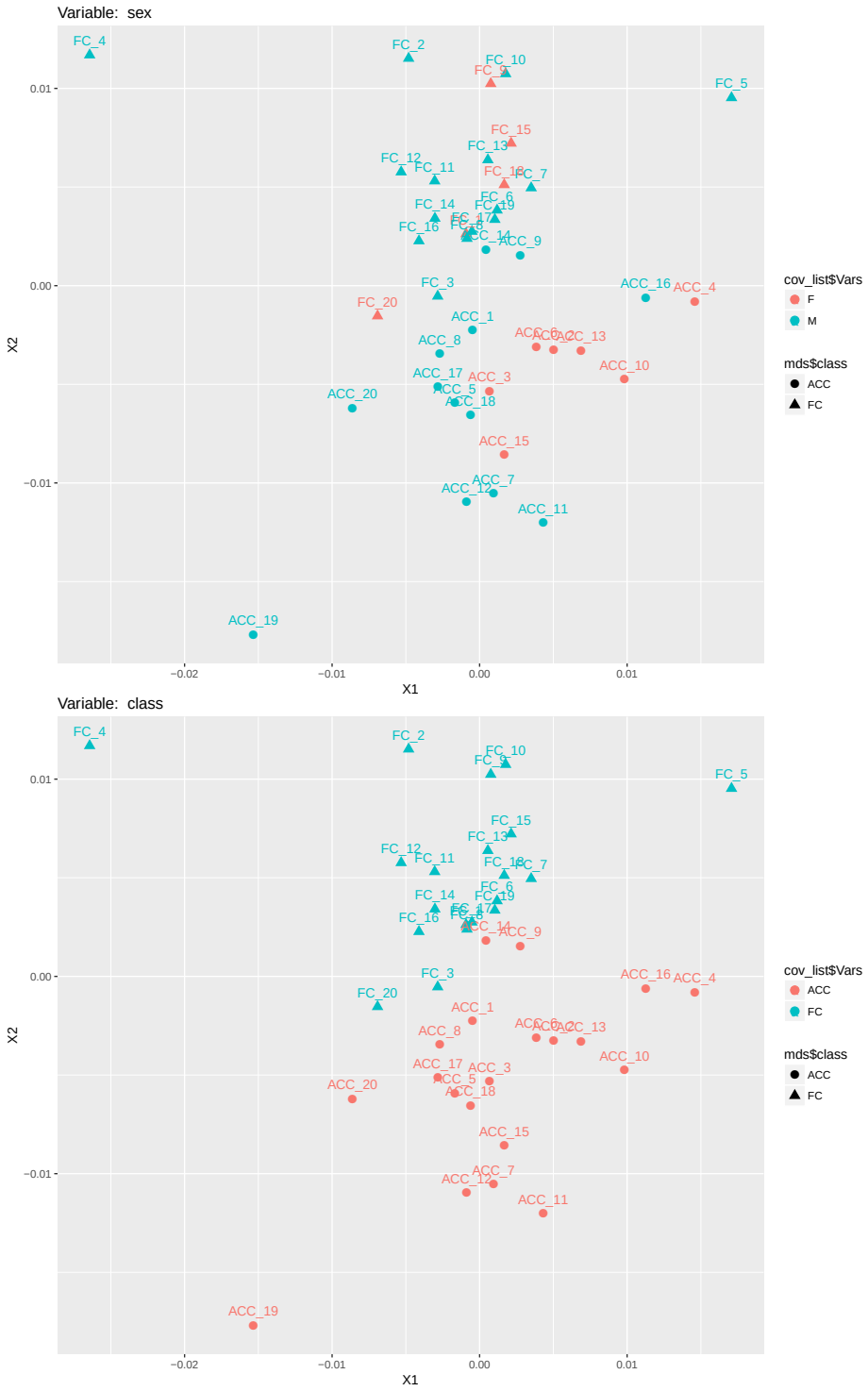

Figure 8: MultiDimentional Scaling plot
An unsupervised MDS plot is drawn. Samples are colored according to 'sex' variable (upper panel) and 'class' (lower panel).

2.6 Feature Selection

The previous step(s) returned a fully filtered, normalized, adjusted expression matrix with the effect of sv removed. However, the number of features in the dataset is still high and greatly exceeds the number of observations. We have to deal, here, with the well-known issue for high-dimensional data known as the "curse of dimensionality". Adding noise features that are not truly associated with the response (*i.e.* class) may lead, in fact, to a worsening model accuracy. In this situation, the user needs to remove those features that bear irrelevant or redundant information. The feature selection technique implemented here does not alter the original representation of the variables, but simply selects a subset of them. It includes three different steps briefly described in the following paragraphs.

2.6.1 Variable selection in Partial Least Squares (PLS)

The first step allows the user to exclude all non-informative class-related features using a backward variable elimination procedure [16]. The `DaMiR.FSelect` function embeds a principal component analysis (PCA) to identify principal components (PCs) that correlate with "class". The correlation coefficient is defined by the user through the `th.corr` argument. The higher the correlation, the lower the number of PCs returned. Importantly, users should pay attention to appropriately set the `th.corr` argument since the total number of retrieved features depends, indeed, on the number of the selected PCs.
The number of class-correlated PCs is then internally used by the function to perform a backward variable elimination-PLS and remove those variables that are less informative with respect to class [17].
Note. Before running the `DaMiR.FSelect` function, we need to transpose our normalized expression data. It can be done by the base R function `t()`. However, we implemented the helper function `DaMiR.transpose` that transposes the data but also tries to prevent the use of tricky feature labels. The "-" and "." characters within variable labels (commonly found, for example, in gene symbols) may, in fact, cause errors if included in the model design as it is required to execute part of the code of the `DaMiR.FSelect` function. Thus, we, firstly, search and, eventually, replace them with non causing error characters.
We used the `set.seed(12345)` function that allows the user to make the results of the whole pipeline reproducible.

```
set.seed(12345)
data_clean<-DaMiR.transpose(assay(data_adjust))
df<-colData(data_adjust)
data_reduced <- DaMiR.FSelect(data_clean, df, th.corr=0.4)

## 19068 Genes have been discarded for classification 275 Genes remained.
```

The "data_reduced" object returns an expression matrix with potentially informative features. In our case study, the initial number of 19052 features has been reduced to 274.

2.6.2 Removing highly correlated features

Some of the returned informative features may, however, be highly correlated. To prevent the inclusion of redundant features that may decrease the model performance during the classification step, we apply a function that produces a pair-wise absolute correlation matrix. When

two features present a correlation higher than `th.corr` argument, the algorithm calculates the mean absolute correlation of each feature and, then, removes the feature with the largest mean absolute correlation.

```
data_reduced <- DaMiR.FReduct(data_reduced$data)

## 55 Highly correlated features have been discarded for classification.
##  220 Features remained.

DaMiR.MDSplot(data_reduced, df)
```

In our example, we used a Spearman's correlation metric and a correletion threshold of 0.85 (default). This reduction step filters out 54 highly correlated genes from the 274 returned by the `DaMiR.FSelect`. The figure below shows the MDS plot drawn by the use of the expression matrix of the remaining 220 genes.

Figure 9: MultiDimentional Scaling plot
A MDS plot is drawn, considering only most informative genes, obtained after feature selection: color code is referred to 'class'.

2.6.3 Ranking and selecting most relevant features

The above functions produced a reduced matrix of variables. Nonetheless, the number of reduced variables might be too high to provide faster and cost-effective classification models. Accordingly, we should properly select a subset of the most informative features. The `DaMiR.FSort` function implements a procedure to rank features by their importance. The method implements a multivariate filter technique (*i.e.* *RReliefF*) that assessess the relevance of features (for details see the `relief` function of the *FSelector* package) [18, 19].

The function produced a data frame with two columns, which reports features ranked by importance scores: a *RReliefF* score and *scaled.RReliefF* value; the latter is computed in this package to implement a "z-score" standardization procedure on *RReliefF* values.
Note. This step may be time-consuming if a data matrix with a high number of features is used as input. We observed, in fact, that there is a quadratic relationship between execution time of the algorithm and the number of features. The user is advised with a message about the estimated time needed to compute the score and rank the features. Thus, we strongly suggest to filter out non informative features by the `DaMiR.FSelect` and `DaMiR.FReduct` functions before performing this step.

```
  # Rank genes by importance:
  df.importance <- DaMiR.FSort(data_reduced, df)
```

```
## Please wait. This operation will take about 40 seconds (i.e. about 1 minutes).
```

```
  head(df.importance)
```

```
##                  RReliefF scaled.RReliefF
## ENSG00000164326 0.3138126        3.153470
## ENSG00000132386 0.3001450        2.957093
## ENSG00000137699 0.2838101        2.722391
## ENSG00000140015 0.2808527        2.679899
## ENSG00000131378 0.2646062        2.446469
## ENSG00000258754 0.2482877        2.212003
```

After the importance score is calculated, a subset of features can be selected and used as predictors for classification purpose. The function `DaMiR.FBest` is used to select a small subset of predictors:

```
  # Select Best Predictors:
  selected_features <- DaMiR.FBest(data_reduced, ranking=df.importance,
                                   n.pred = 5)
```

```
## 5 Predictors have been selected for classification
```

```
  selected_features$predictors
```

```
## [1] "ENSG00000164326" "ENSG00000132386" "ENSG00000137699" "ENSG00000140015"
## [5] "ENSG00000131378"
```

```
  # Dendrogram and heatmap:
  DaMiR.Clustplot(selected_features$data, df)
```

Here, we selected the first 5 genes (default) ranked by importance.
Note. The user may also wish to select "automatically" (*i.e.* not defined by the user) the number of important genes. This is possible by setting `autoselect="yes"` and a threshold for the *scaled.RReliefF*, *i.e.* `th.zscore` argument. These normalized values (rescaled to have a mean of 0 and standard deviation of 1) make it possible to compare predictors ranking obtained by running the pipeline with different parameters.

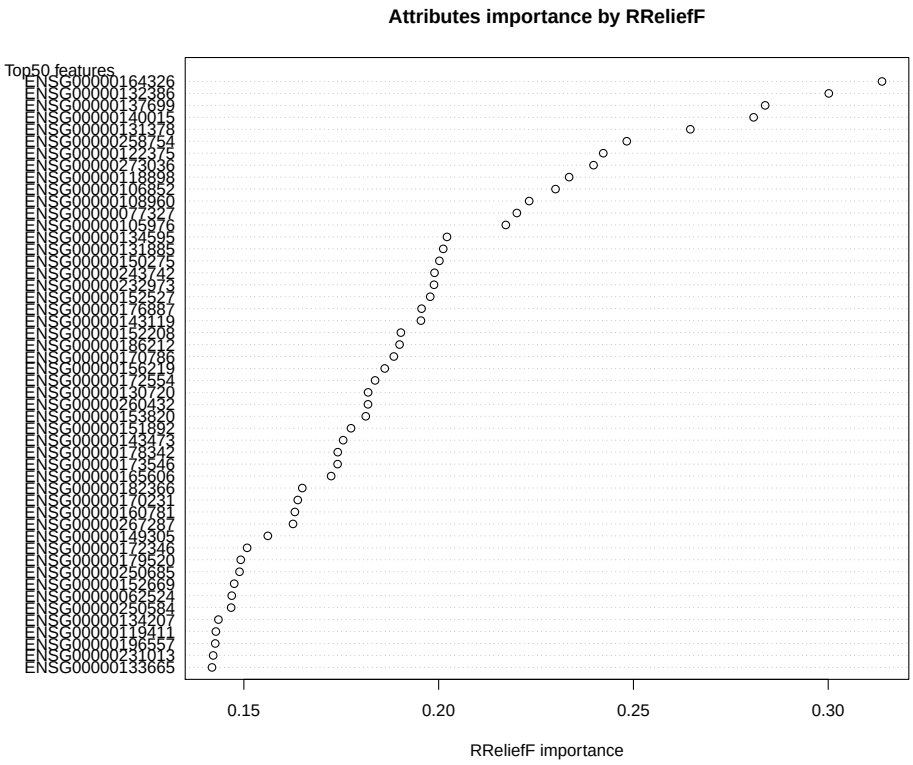

Figure 10: **Feature Importance Plot**
The dotchart shows the list of top 50 genes, sorted by RReliefF importance score. This plot may be used to select the most important predictors to be used for classification.

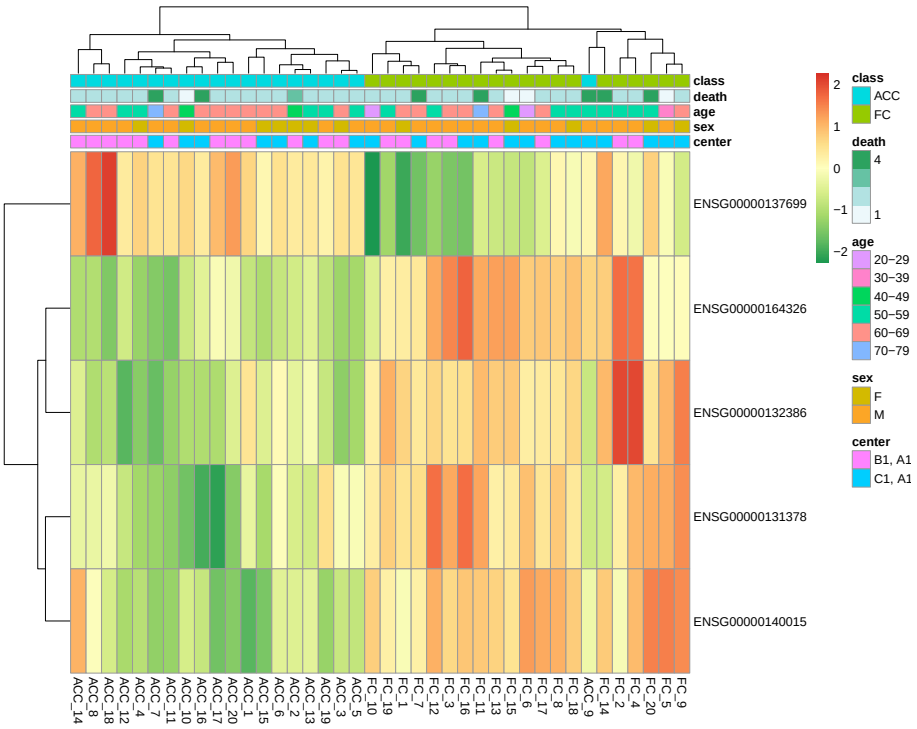

Figure 11: Clustergram
The clustergram is generated by using the expression values of the 5 predictors selected by `DaMiR.FBest` function. As for the heatmap generated by `DaMiR.Allplot`, 'class' and covariates are drawn as horizontal and color coded bars.

2.7 Classification

All the steps executed so far allowed the normalization, cleaning and reduction of the original expression matrix; the objective is to capture a subset of original data as informative as possible, in order to carry out a classification analysis. In this paragraph, we describe the statistical learning strategy we implemented to tackle both binary and multi-class classification problems.
A meta-learner is built, combining up to 8 different classifiers through a "Stacking" strategy. Currently, there is no gold standard for creating the best rule to combine predictions [6]. We decided to implement a framework that relies on the "weighted majority voting" approach [20]. In particular, our method estimates a weight for each used classifier, based on its own accuracy, and then use these weights, together with predictions, to fine-tune a decision rule (*i.e.* meta-learner). Briefly, first a training set (TR1) and a test set (TS1) are generated by "Bootstrap" sampling. Then, sampling again from subset TR1, another pair of training (TR2) and test set (TS2) were obtained. TR2 is used to train RF, NB, SVM, 3kNN, LDA, NN, PLS and/or LR classifiers (the number and the type are chosen by the user), whereas TS2 is used to test their accuracy and to calculate weights (w) by formula:

$$w_{classifier_i} = \frac{Accuracy_{classifier_i}}{\sum_{j=1}^{N} Accuracy_{classifier_j}} \quad \boxed{1}$$

where i is a specific classifiers and N is the total number of them (here, $N <= 8$). Using this approach:

$$\sum_{i=1}^{N} w_i = 1 \quad \boxed{2}$$

The higher the value of w_i, the more accurate is the classifier.
The performance of the meta-learner (labelled as "Ensemble") is evaluated by using TS1. The decision rule of the meta-learner is made by a linear combination of the products between weigths (w) and predictions (Pr) of each classifier; for each sample *k*, the prediction is computed by:

$$Pr_{(k,Ensemble)} = \sum_{i=1}^{N} w_i * Pr_{(k,classifier_i)} \quad \boxed{3}$$

$Pr_{(k,Ensemble)}$ ranges from 0 to 1. For binary classification analysis, 0 means high probability to belong to one class, while 1 means high probability to belong to the other class); predictions close to 0.5 have to be considered as made by chance. For multi-class analysis 1 means right prediction, while 0 means wrong prediction. This process is repeated several times to assess the robustness of the set of predictors used.

This procedure is implemented in the `DaMiR.EnsembleLearning` function, where `fSample.tr`, `fSample.tr.w` and `iter` arguments allow the algorithm tuning.
To speed up the execution time of the function, we set `iter = 30` (default is 100) but we suggest to use an higher number of iterations to obtain more accurate results.

```
  Classification_res <- DaMiR.EnsembleLearning(selected_features$data,
                          classes=df$class, fSample.tr = 0.5,
                          fSample.tr.w = 0.5, iter = 30)
```

```
## You select: RF LR kNN LDA NB SVM weak classifiers for creating
##       the Ensemble meta-learner.
## Ensemble classification is running.  30  iterations were chosen:
## Accuracy [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 99.5 99.67 99 96.5 94.67 99.33 96.83
##  St.Dev. 1.5 1.3 2 4.6 9.9 1.7 4.4
## MCC score [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 99 99.33 98 93.3 89.47 98.67 93.97
##  St.Dev. 3.1 2.5 4.1 8.6 19.5 3.5 8.3
## Specificity [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 99.67 99.67 98.67 98.67 93.67 99 96
##  St.Dev. 1.8 1.8 3.5 4.3 14.5 3.1 6.7
## Sensitivity [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 99.33 99.67 99.33 94.33 95.67 99.67 97.67
##  St.Dev. 2.5 1.8 2.5 8.2 7.3 1.8 6.3
```

The function returns a list containing the matrix of accuracies of each classifier in each iteration and, in the case of a binary classification problem, the specificity, the sensitivity and the Matthew's Correlation Coefficient (MCC). These objects can be accessed using the $ accessor.

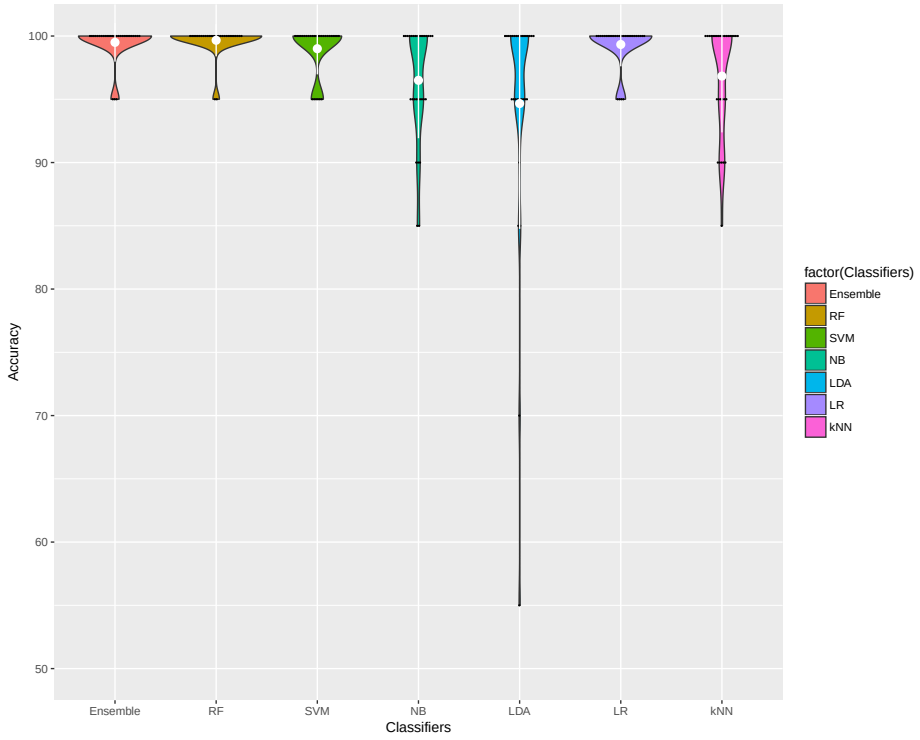

Figure 12: Accuracies Comparison
The violin plot highlights the classification accuracy of each classifier, computed at each iteration; a black dot represents a specific accuracy value while the shape of each "violin" is drawn by a Gaussian kernel density estimation. Averaged accuracies and standard deviations are represented by white dots and lines.

As shown in Figure 12 almost all single, weak classifiers show high or very high classification performancies, in terms of accuracy, specificity, sensitivity and MCC.

2.8 Exporting output data

DaMiRseq has been designed to allow users to export the outputs of each function, which consist substantially in `matrix` or `data.frame` objects. Export can be done, using the base R functions, such as `write.table` or `write.csv`. For example, we could be interested in saving normalized data matrix, stored in "data_norm" in a tab-delimited file:

```
outputfile <- "DataNormalized.txt"
write.table(data_norm, file = outputfile_norm, quote = FALSE, sep = "\t")
```

3 Adjusting the data: a necessary step?

In this section, we highlight how the early step of data correction could impact on the final classification results. Data transformation and global scaling approaches are traditionally applied to expression data but they could not be always effective to capture unwanted source of variation. High-dimensional data are, in fact, known to be deeply influenced by noises

and biases of high-throughput experiments. For this reason, we strongly suggest to check the presence of any confounding factor and assess their possible effect since they could dramatically alter the result. However, the step described in Section 2.4 could be skipped if we assume that the data are not affected by any batches (known or unknown), or if we do not want to take them into account. Thus, we performed, here, the same feature selection and classification procedure as applied before but without removing the putative noise effects from our expression data. In this case, VST normalized data will be used. Since the functions embedded into these steps require a random sampling to be executed, we set the same seed as in Section 2 (*i.e.* `set.seed(12345)`) to ensure a right comparison between results.

Note. For simplicity, here we do not produce all plots, except for the violin plot generated by `DaMiR.EnsembleLearning`, used to compare the performances, although the usage of `DaMiR.Allplot`, `DaMiR.corrplot`, `DaMiR.Clustplot` and `DaMiR.MDSplot` **is crucial** to check the effect of each process.

```
## Feature Selection
set.seed(12345)
data_clean_2<-DaMiR.transpose(assay(data_filt))
df_2<-colData(data_filt)

data_reduced_2 <- DaMiR.FSelect(data_clean_2, df_2, th.corr=0.4)

## 19258 Genes have been discarded for classification 85 Genes remained.

data_reduced_2 <- DaMiR.FReduct(data_reduced_2$data)

## 20 Highly correlated features have been discarded for classification.
##  65 Features remained.

df.importance_2 <- DaMiR.FSort(data_reduced_2, df_2)

## Please wait. This operation will take about 20 seconds (i.e. about 0 minutes).

head(df.importance_2)

##                  RReliefF scaled.RReliefF
## ENSG00000164326 0.3396526        3.673520
## ENSG00000258754 0.2575880        2.642819
## ENSG00000094796 0.2076106        2.015121
## ENSG00000169432 0.2042621        1.973065
## ENSG00000170290 0.1630930        1.455997
## ENSG00000130720 0.1523449        1.321006

selected_features_2 <- DaMiR.FBest(data_reduced_2, ranking=df.importance_2,
                                   n.pred=5)

## 5 Predictors have been selected for classification

selected_features_2$predictors

## [1] "ENSG00000164326" "ENSG00000258754" "ENSG00000094796" "ENSG00000169432"
## [5] "ENSG00000170290"

## Classification
Classification_res_2 <- DaMiR.EnsembleLearning(selected_features_2$data,
                                               classes=df_2$class,
                                               fSample.tr = 0.5,
                                               fSample.tr.w = 0.5,
```

```
                                                iter = 30)

## You select: RF LR kNN LDA NB SVM weak classifiers for creating
##       the Ensemble meta-learner.
## Ensemble classification is running.  30  iterations were chosen:
## Accuracy [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 95.67 92.33 95.5 95.67 88.17 95.33 93.17
##  St.Dev. 3.1 5.5 3.3 4.7 7 3.9 6.9
## MCC score [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 91.4 85.13 91.27 91.7 77.47 90.8 86.83
##  St.Dev. 6.2 10.5 6.2 8.6 13.5 7.7 13.2
## Specificity [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 96.67 91.67 97 94.33 87.67 95.33 93
##  St.Dev. 5.5 8.7 6 9 11.7 5.1 9.5
## Sensitivity [%]:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 94.67 93 94 97 88.67 95.33 93.33
##  St.Dev. 5.1 8.4 6.2 5.3 12 6.3 9.9
```

The consequence of data adjustment is already remarkable after the feature selection and reduction steps. The number of selected genes, indeed, decreased from 220 to 98 when data adjustment was not performed, suggesting that hidden factors may influence gene expression and likely mask class-related features. Furthermore, the ranking of the important features also differs if data correction is not applied. The two sets of 5 genes that are used to build the classification models shares, in fact, only 1 gene. This suggests that data adjustment affects both the number and the quality of the features that can be selected for classification. Therefore, the overall classification performances, without the appropriate data correction, hugely felt down below 90% of accuracy for all the classifiers.

Figure 13 shows the results of the variation to standard workflow of *DaMiRseq*, proposed in this Section. Taking as reference the "Standard Workflow", described in Section 2, we can observe that the performances significantly decrease.

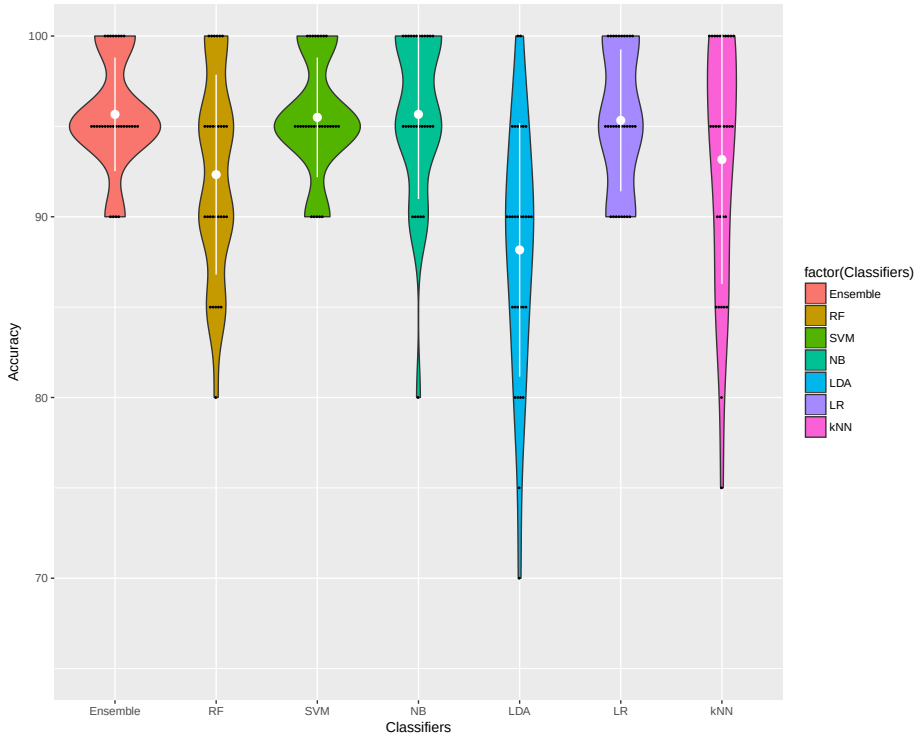

Figure 13: **Accuracies Comparison**
The violin plot shows the effect of the modification to *DaMiRseq* standard workflow, described in Section 3: without adjusting data (following the steps described in Section 2.4), performances usually decrease; this could be explained by the fact that some noise, probably coming from unknown source of variation, is present in the dataset.

4 New implementations

4.1 Version 1.3.1

Relevant modifications:

- *DaMiRseq* performs both binary and multi-class classification anlysis;
- The "Stacking" meta-learner can be composed by the user, setting the new parameter `cl_type` of the `DaMiR.EnsembleLearning` function. Any combination up to 8 classifiers ("RF", "NB", "kNN", "SVM", "LDA", "LR", "NN", "PLS") is now allowed;
- If the dataset is imbalanced, a "Down-Sampling" strategy is automatically applied;
- The `DaMiR.FSelect` function has the new argument, called `nPlsIter`, which allows the user to have a more robust features set. In fact, several feature sets are generated by the `bve_pls` fuction (embedded in `DaMiR.FSelect`), setting 'nPLSIter' parameter greater than 1. Finally, an intersection among all the feature sets is performed to return those features which constantly occur in all runs. However, by default, `nPlsIter = 1`.

Minor modifications and bugs fixed:

- `DaMiR.Allplot` accepts also 'matrix' objects;

- The `DaMiR.normalization` function estimates the dispersion, through the parameter `nFitType`; as in *DESeq2* package, the argument can be 'parametric' (default), 'local' and 'mean';
- In the `DaMiR.normalization` function, the gene filtering is desabled if `minCount = 0`;
- In the `DaMiR.EnsembleLearning` function, the method for implementing the Logistic Regression has been changed to allow multi-class comparisons; instead of the native `lm` function, the `bayesglm` method implemented in the *caret* `train` function is now used;
- The new parameter `second.var` of the `DaMiR.SV` function, allows the user to take into account a secondary variable of interest (factorial or numerical) that the user does not wish to correct for, during the sv identification.

5 Session Info

- R version 3.4.2 (2017-09-28), `x86_64-w64-mingw32`
- Locale: `LC_COLLATE=Italian_Italy.1252`, `LC_CTYPE=Italian_Italy.1252`, `LC_MONETARY=Italian_Italy.1252`, `LC_NUMERIC=C`, `LC_TIME=Italian_Italy.1252`
- Running under: `Windows 7 x64 (build 7601) Service Pack 1`
- Matrix products: default
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: arm 1.9-3, Biobase 2.38.0, BiocGenerics 0.24.0, caret 6.0-77, DaMiRseq 1.3.1, DelayedArray 0.4.0, GenomeInfoDb 1.14.0, GenomicRanges 1.30.0, ggplot2 2.2.1, IRanges 2.12.0, knitr 1.17, lattice 0.20-35, lme4 1.1-14, MASS 7.3-47, Matrix 1.2-11, matrixStats 0.52.2, S4Vectors 0.16.0, SummarizedExperiment 1.8.0
- Loaded via a namespace (and not attached): abind 1.4-5, acepack 1.4.1, annotate 1.56.0, AnnotationDbi 1.40.0, aroma.light 3.8.0, assertthat 0.2.0, backports 1.1.1, base64enc 0.1-3, bdsmatrix 1.3-2, bindr 0.1, bindrcpp 0.2, BiocParallel 1.12.0, BiocStyle 2.6.0, biomaRt 2.34.0, Biostrings 2.46.0, bit 1.1-12, bit64 0.9-7, bitops 1.0-6, blob 1.1.0, checkmate 1.8.5, class 7.3-14, cluster 2.0.6, coda 0.19-1, codetools 0.2-15, colorspace 1.3-2, compiler 3.4.2, corrplot 0.84, CVST 0.2-1, data.table 1.10.4-3, DBI 0.7, ddalpha 1.3.1, DEoptimR 1.0-8, DESeq 1.30.0, DESeq2 1.18.0, digest 0.6.12, dimRed 0.1.0, dplyr 0.7.4, DRR 0.0.2, e1071 1.6-8, EDASeq 2.12.0, entropy 1.2.1, evaluate 0.10.1, FactoMineR 1.38, flashClust 1.01-2, foreach 1.4.3, foreign 0.8-69, Formula 1.2-2, FSelector 0.21, genalg 0.2.0, genefilter 1.60.0, geneplotter 1.56.0, GenomeInfoDbData 0.99.1, GenomicAlignments 1.14.0, GenomicFeatures 1.30.0, glue 1.2.0, gower 0.1.2, grid 3.4.2, gridExtra 2.3, gtable 0.2.0, highr 0.6, Hmisc 4.0-3, htmlTable 1.9, htmltools 0.3.6, htmlwidgets 0.9, hwriter 1.3.2, igraph 1.1.2, ineq 0.2-13, ipred 0.9-6, iterators 1.0.8, kernlab 0.9-25, kknn 1.3.1, labeling 0.3, latticeExtra 0.6-28, lava 1.5.1, lazyeval 0.2.1, leaps 3.0, limma 3.34.1, locfit 1.5-9.1, lubridate 1.6.0, magrittr 1.5, memoise 1.1.0, mgcv 1.8-20, minqa 1.2.4, ModelMetrics 1.1.0, munsell 0.4.3, mvtnorm 1.0-6, nlme 3.1-131, nloptr 1.0.4, nnet 7.3-12, pheatmap 1.0.8, pkgconfig 2.0.1, pls 2.6-0, plsVarSel 0.9.1, plyr 1.8.4, prettyunits 1.0.2, prodlim 1.6.1, progress 1.1.2, purrr 0.2.4, R.methodsS3 1.7.1, R.oo 1.21.0, R.utils 2.5.0, R6 2.2.2, randomForest 4.6-12, RColorBrewer 1.1-2,

Rcpp 0.12.13, RcppRoll 0.2.2, RCurl 1.95-4.8, recipes 0.1.0, reshape2 1.4.2, rJava 0.9-9, rlang 0.1.2, rmarkdown 1.6, RMySQL 0.10.13, robustbase 0.92-8, rpart 4.1-11, rprojroot 1.2, Rsamtools 1.30.0, RSNNS 0.4-9, RSQLite 2.0, rtracklayer 1.38.0, RWeka 0.4-35, RWekajars 3.9.1-4, scales 0.5.0, scatterplot3d 0.3-40, sfsmisc 1.1-1, ShortRead 1.36.0, splines 3.4.2, stringi 1.1.5, stringr 1.2.0, survival 2.41-3, sva 3.26.0, tibble 1.3.4, timeDate 3012.100, tools 3.4.2, withr 2.1.0, XML 3.98-1.9, xtable 1.8-2, XVector 0.18.0, yaml 2.1.14, zlibbioc 1.24.0

References

[1] Jeffrey T Leek and John D Storey. Capturing heterogeneity in gene expression studies by surrogate variable analysis. *PLoS Genet*, 3(9):e161, 2007.

[2] Andrew E Jaffe, Thomas Hyde, Joel Kleinman, Daniel R Weinbergern, Joshua G Chenoweth, Ronald D McKay, Jeffrey T Leek, and Carlo Colantuoni. Practical impacts of genomic data "cleaning" on biological discovery using surrogate variable analysis. *BMC bioinformatics*, 16(1):1, 2015.

[3] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[4] Yvan Saeys, Iñaki Inza, and Pedro Larrañaga. A review of feature selection techniques in bioinformatics. *bioinformatics*, 23(19):2507–2517, 2007.

[5] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[6] Robi Polikar. Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45, 2006.

[7] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[8] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[9] GTEx Consortium et al. The genotype-tissue expression (gtex) pilot analysis: Multitissue gene regulation in humans. *Science*, 348(6235):648–660, 2015.

[10] Martin Morgan, Valerie Obenchain, Jim Hester, and Hervé Pagès. *SummarizedExperiment: SummarizedExperiment container*, 2016. R package version 1.4.0.

[11] Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for rna-seq data with deseq2. *Genome biology*, 15(12):1, 2014.

[12] Jeffrey T Leek, W Evan Johnson, Hilary S Parker, Andrew E Jaffe, and John D Storey. The sva package for removing batch effects and other unwanted variation in high-throughput experiments. *Bioinformatics*, 28(6):882–883, 2012.

[13] Matthew E Ritchie, Belinda Phipson, Di Wu, Yifang Hu, Charity W Law, Wei Shi, and Gordon K Smyth. limma powers differential expression analyses for rna-sequencing and microarray studies. *Nucleic acids research*, page gkv007, 2015.

[14] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer, 2016.

[15] Davide Risso, Katja Schwartz, Gavin Sherlock, and Sandrine Dudoit. Gc-content normalization for rna-seq data. *BMC bioinformatics*, 12(1):480, 2011.

[16] Tahir Mehmood, Kristian Hovde Liland, Lars Snipen, and Solve Sæbø. A review of variable selection methods in partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, 118:62–69, 2012.

[17] Ildiko E Frank. Intermediate least squares regression method. *Chemometrics and Intelligent Laboratory Systems*, 1(3):233–242, 1987.

[18] Igor Kononenko. Estimating attributes: analysis and extensions of relief. In *European conference on machine learning*, pages 171–182. Springer, 1994.

[19] Marko Robnik-Šikonja and Igor Kononenko. An adaptation of relief for attribute estimation in regression. In *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, pages 296–304, 1997.

[20] N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212 – 261, 1994. URL: http://www.sciencedirect.com/science/article/pii/S0890540184710091, doi:http://dx.doi.org/10.1006/inco.1994.1009.