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;

November 13, 2026

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 2.24.0

Contents

1 Citing DaMiRseq . . . 3

2 Introduction . . . 4

3 Data Handling . . . 5

3.1 Input data . . . 5

3.2 Import Data . . . 5

3.3 Preprocessing and Normalization . . . 6

3.3.1 Filtering by Expression . . . 7

3.3.2 Filtering By Coefficient of Variation (CV) . . . 7

3.3.3 Normalization . . . 8

3.3.4 Sample Filtering . . . 8

3.4 Adjusting Data . . . 9

3.4.1 Identification of Surrogate Variables . . . 9

3.4.2 Correlation between sv and known covariates . . . 10

3.4.3 Cleaning expression data . . . 11

3.5 Exploring Data . . . 11

3.6 Exporting output data . . . 22

4 Two specific supervised machine learning workflows . . . 23

4.1 Finding a small set of informative, robust features . . . 24

4.1.1 Feature Selection . . . 24

4.1.2 Classification . . . 29

4.2 Building the optimal prediction model . . . 31

4.2.1 Training and testing inside the cross-validation . . . 32

4.2.2 Selection and Prediction . . . 34

5 Normalizing and Adjusting real independent test sets . . . 35

6 Adjusting the data: a necessary step? . . . 39

7 Check new implementations! . . . 42

7.1 Version 2.0.0, devel: 2.1.0 . . . 42

7.2 Version 1.6, devel: 1.5.2 . . . 42

7.3 Version 1.4.1 . . . 43

7.4 Version 1.4 . . . 43

8 Session Info . . . 43

1 Citing DaMiRseq

For citing DaMiRseq:

```
citation("DaMiRseq")
```

```
## To cite package 'DaMiRseq' in publications use:
##
##   Mattia Chiesa, Gualtiero I. Colombo and Luca Piacentini. DaMiRseq
##   - an R/Bioconductor package for data mining of RNA-Seq data:
##   normalization, feature selection and classification
##   Bioinformatics, https://doi.org/10.1093/bioinformatics/btx795 ,
##   2018
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {DaMiRseq - an R/Bioconductor package for data mining of
##          RNA-Seq data: normalization, feature selection and classification},
##     author = {Mattia Chiesa and Gualtiero I. Colombo and Luca Piacentini},
##     journal = {Bioinformatics},
##     volume = {34},
##     number = {8},
##     pages = {1416-1418},
##     year = {2018},
##     doi = {10.1093/bioinformatics/btx795},
##   }
```

2 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].

3 Data Handling

3.1 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.

3.2 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 3.1 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)[2]
# filecovariates <- list.files(rawdata.path, full.names = TRUE)[1]
# count_data <- read.delim(filecounts)
# covariate_data <- read.delim(filecovariates, stringAsFactor = T)
# 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.

3.3 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.

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

3.3.1 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.

3.3.2 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.
Here, 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.

3.3.3 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(SE, minCounts=10, fSample=0.7, th.cv=3)
# VST: about 80 seconds
#
#data_norm <- DaMiR.normalization(SE, 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 3.3.2) `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. **Note.** A general note on data normalization and its implications for the analysis of high-dimensional data can be found in the *Chiesa et al.* Supplementary data [12].

3.3.4 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.

3.4 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].

3.4.1 Identification of Surrogate Variables

Surrogate variables identification, basically, relies on the SVA algorithm by Leek et al. [13] [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. Interested readers can find further explanations about the 'fve' and comparison with other methods in the 'Supplementary data' of *Chiesa et al.* [12].

```
 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.

[2] See *sva* package

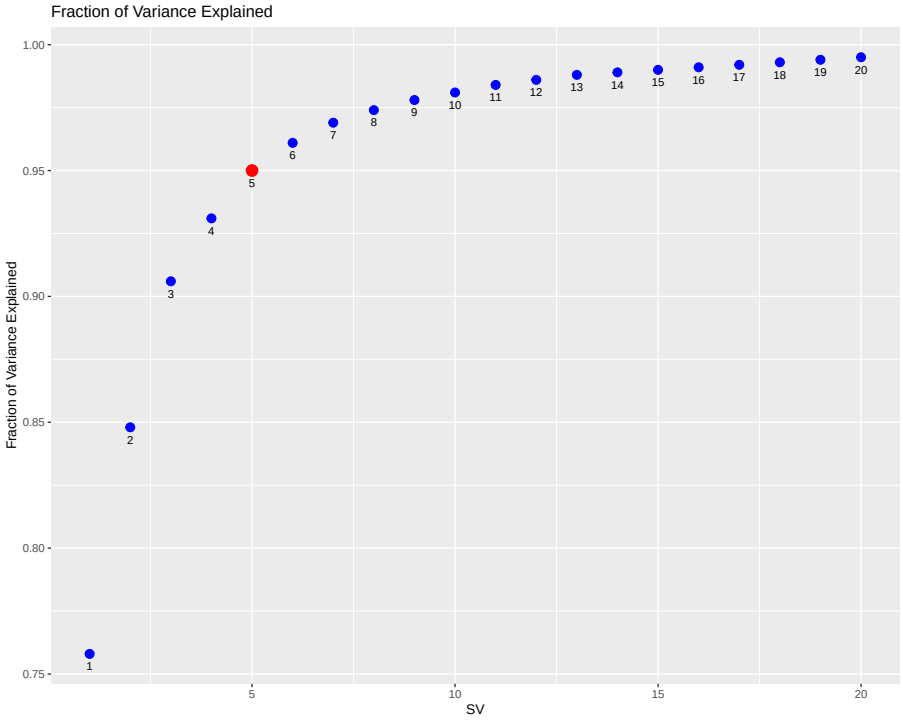


Figure 1: Fraction of Variance Explained. This plot shows the relationship between each identified sv and the corresponding fraction of variance explained. A specific blue dot represents the proportion of variance, explained by a sv together with the prior ones. The red dot marks the upper limit of sv 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.

3.4.2 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. 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 **??**, 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 a. 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.

Note b. The `DaMiR.corrplot` function performs a standard correlation analysis between SVs and known variables. Correlation functions need to transform factors into numbers in order to work. Importantly, by default, R follows an alphabetical order to assign numbers to factors. Therefore, the correlation index will make sense when the known variables are::

- continuous covariates, such as the "age" variable in the package's sample data
- ordinal factors, in which factors can be graded accordingly to a specific ordinal rank, for example: "1=small", "2=medium","3=large";
- dichotomous categorical variables, where the rank is not important but the maximum number of factors is 2; e.g., sex = M or F, clinical variable = YES or NO

On the other hand, if a variable consists of factors with more than 2 levels and an ordinal rank can not be defined (e.g. color = "red" or "blue" or "green"), it is likely that the correlation index will give rise to a misleading interpretation, e.g. the absence of correlation even though there may be a significant association between the multi-level factor versus the surrogate variables. In this case, we warmly recommend to perform a linear regression (for example by the `lm()` to assess the relationship between each surrogate variable and the multi-level factorial variable(s) to be evaluated. For simplicity, we assumed that, herein, we did not have the latter type of variable.

3.4.3 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 [14]. 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 3.4.1).

```
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 8.290889 9.516290 8.863299 8.347355 9.023698 8.701100
## ENSG00000237683 7.397297 7.393483 6.596511 6.860754 6.546220 7.809292
## ENSG00000268903 5.652618 5.828790 5.074126 5.679716 5.321179 6.246789
## ENSG00000228463 7.762128 7.472593 7.599905 6.909482 6.900152 7.484546
## ENSG00000241670 5.565704 5.650229 6.048206 5.394268 5.276310 5.566598
##                     FC_2     FC_3     FC_4     FC_5
## ENSG00000227232 8.492608 8.939920 8.535679 9.126093
## ENSG00000237683 7.904175 7.024476 6.999965 7.093138
## ENSG00000268903 5.769698 5.577973 5.704730 5.483063
## ENSG00000228463 7.236964 7.593969 7.512355 7.713643
## ENSG00000241670 4.762900 4.847968 5.047882 5.521918
```

Now, 'data_adjust' object contains a numeric matrix of log2-expression values with sv effects removed. An example of the effective use of our 'fve' method has been obtained for the detection of sv in a dataset of adipose tissue samples from abdominal aortic aneurysm patients by *Piacentine et al.* [15].

3.5 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 [16], 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 [17], 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.

Sample-by-Sample expression distribution - This plot, drawn by *ggplot2* package, helps to visualize the differences between the real expression distributions across samples: shapes of every samples should be the same; indeed, samples with unusual shapes are likely outliers.

Average expression distribution by class - This plot, drawn by *ggplot2* package, helps to visualize the differences between the average expression distribution for each class.

In this vignette, `DaMiR.Allplot` is used to appreciate the effect of data adjusting (see Section 3.4). First, we check how data appear just after normalization: the heatmap and RLE plot in Figure 2 (upper and lower panel, respectively) and MDS plots in Figures 3 and 4 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.

```
# 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.

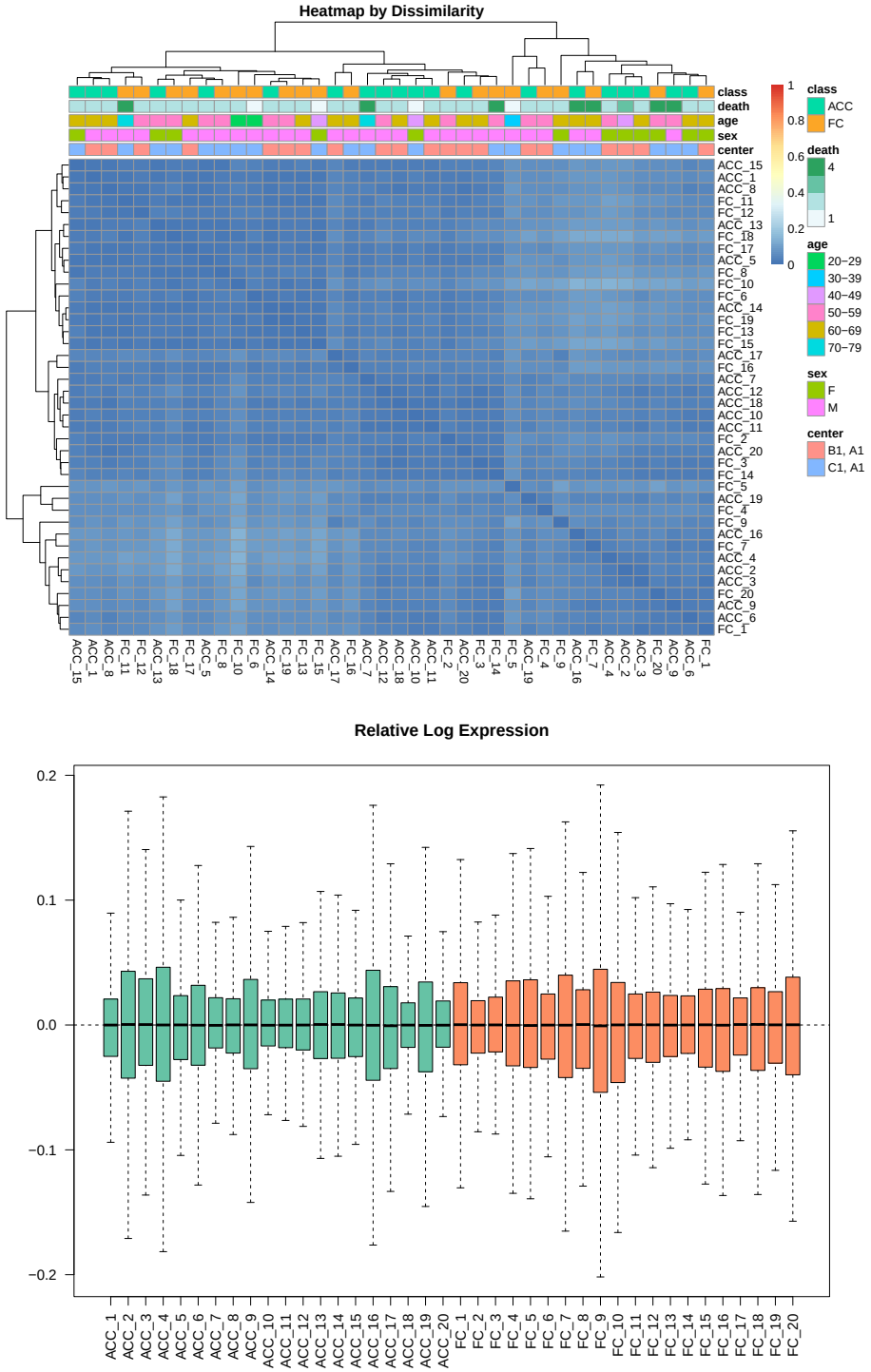


Figure 2: Heatmap and RLE. Heatmap (upper panel): colors in heatmap highlight the distance matrix, obtained by Spearman's correlation metric: color gradient ranges from *dark green*, meaning 'minimum distance' (*i.e.* dissimilarity = 0, correlation = 1), to *light green green*. On the top of heatmap, 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.

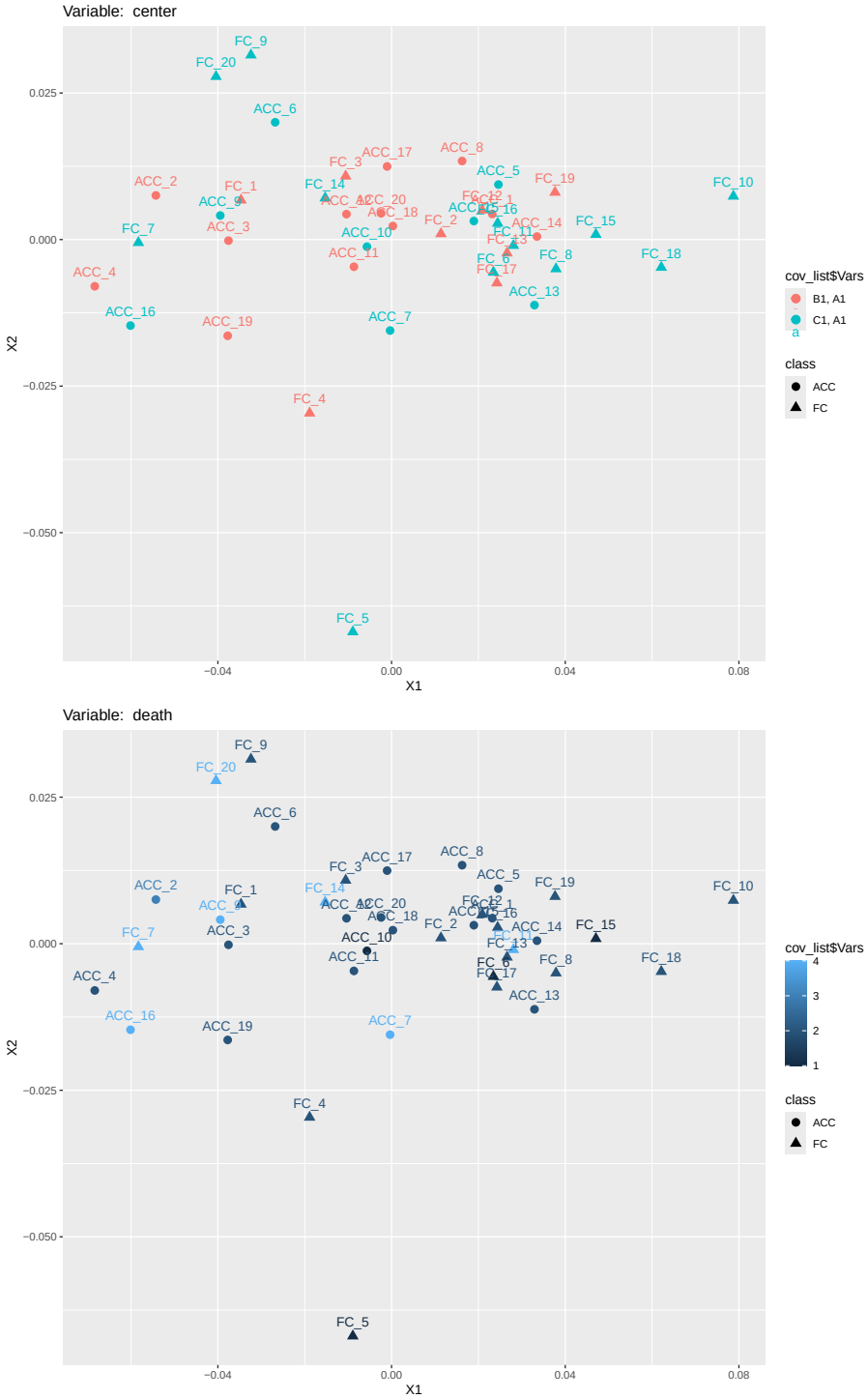


Figure 3: MultiDimentional Scaling plot. An unsupervised MDS plot is drawn. Samples are colored according to the 'Hardy death scale' (upper panel) and the 'center' variable (lower panel).

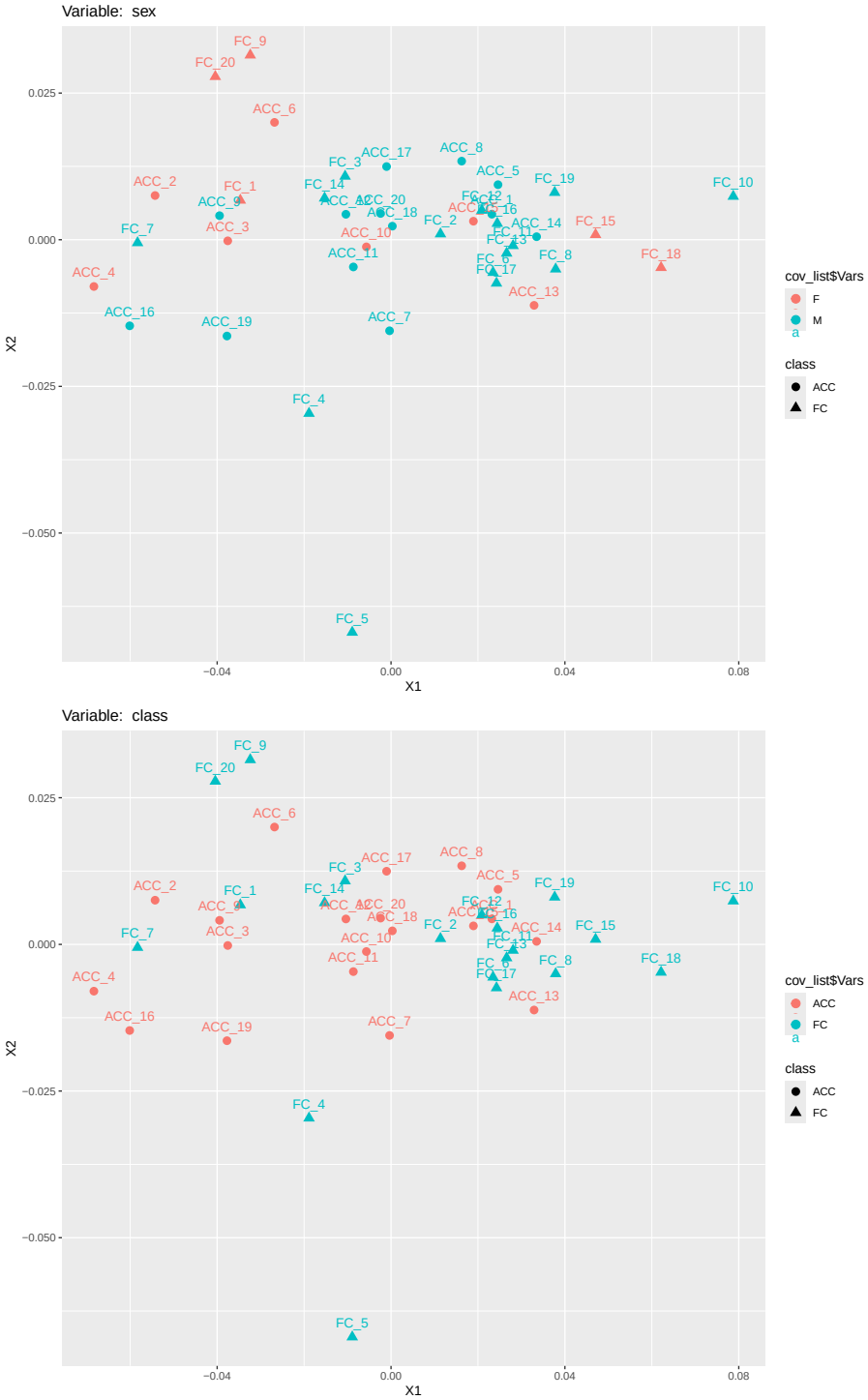


Figure 4: MultiDimentional Scaling plot. An unsupervised MDS plot is drawn. Samples are colored according to 'sex' variable (upper panel) and 'class' (lower panel).

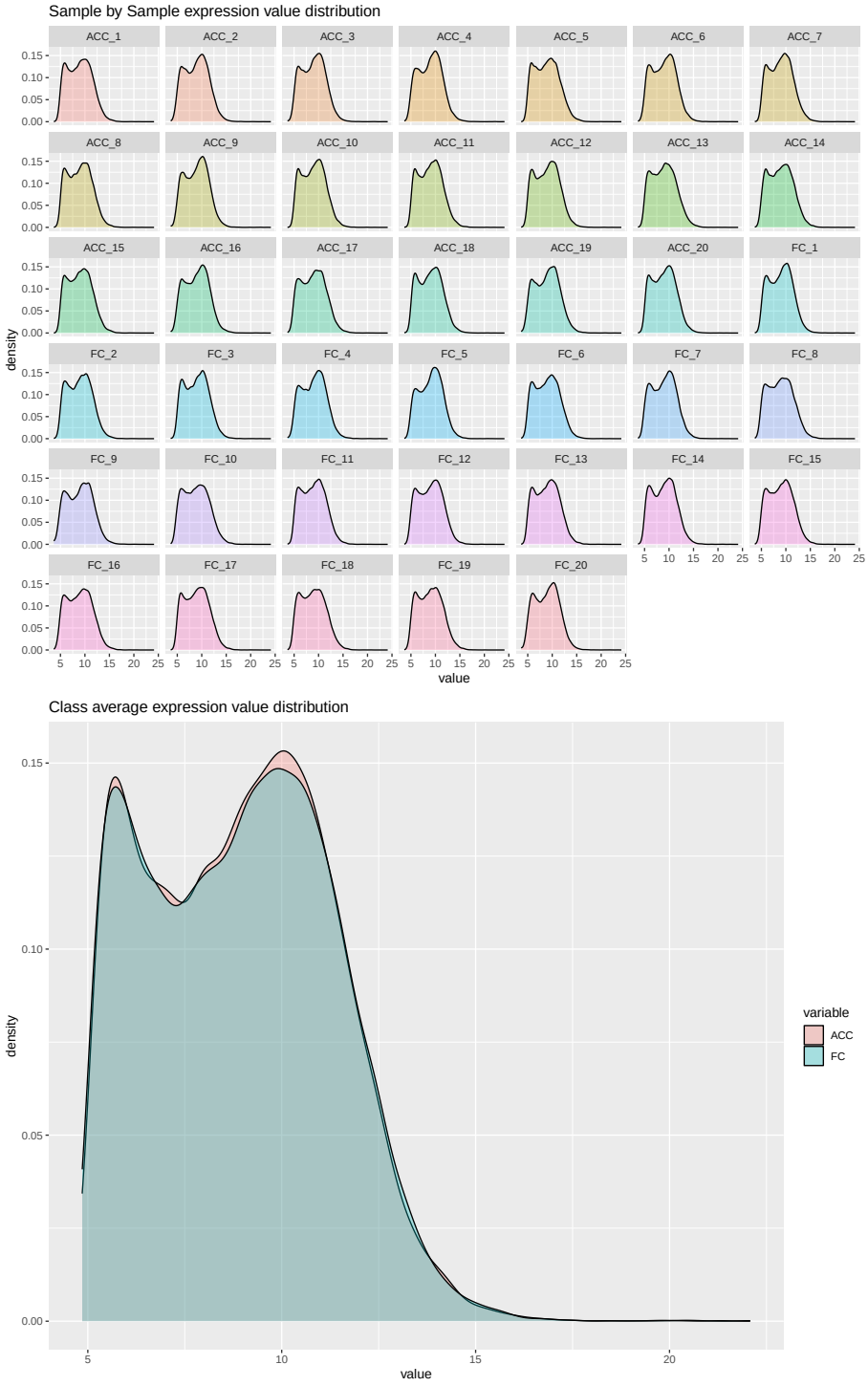


Figure 5: Gene Expression distribution. Sample-by-Sample expression distribution (upper panel) helps user to find outliers and to control the effect of normalization, filtering and adjusting steps; class average expression distribution (lower panel) highlights global expression differences between classes.

After removing the effect of "noise" from our expression data, as presented in Section 3.4, 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 2 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))
```

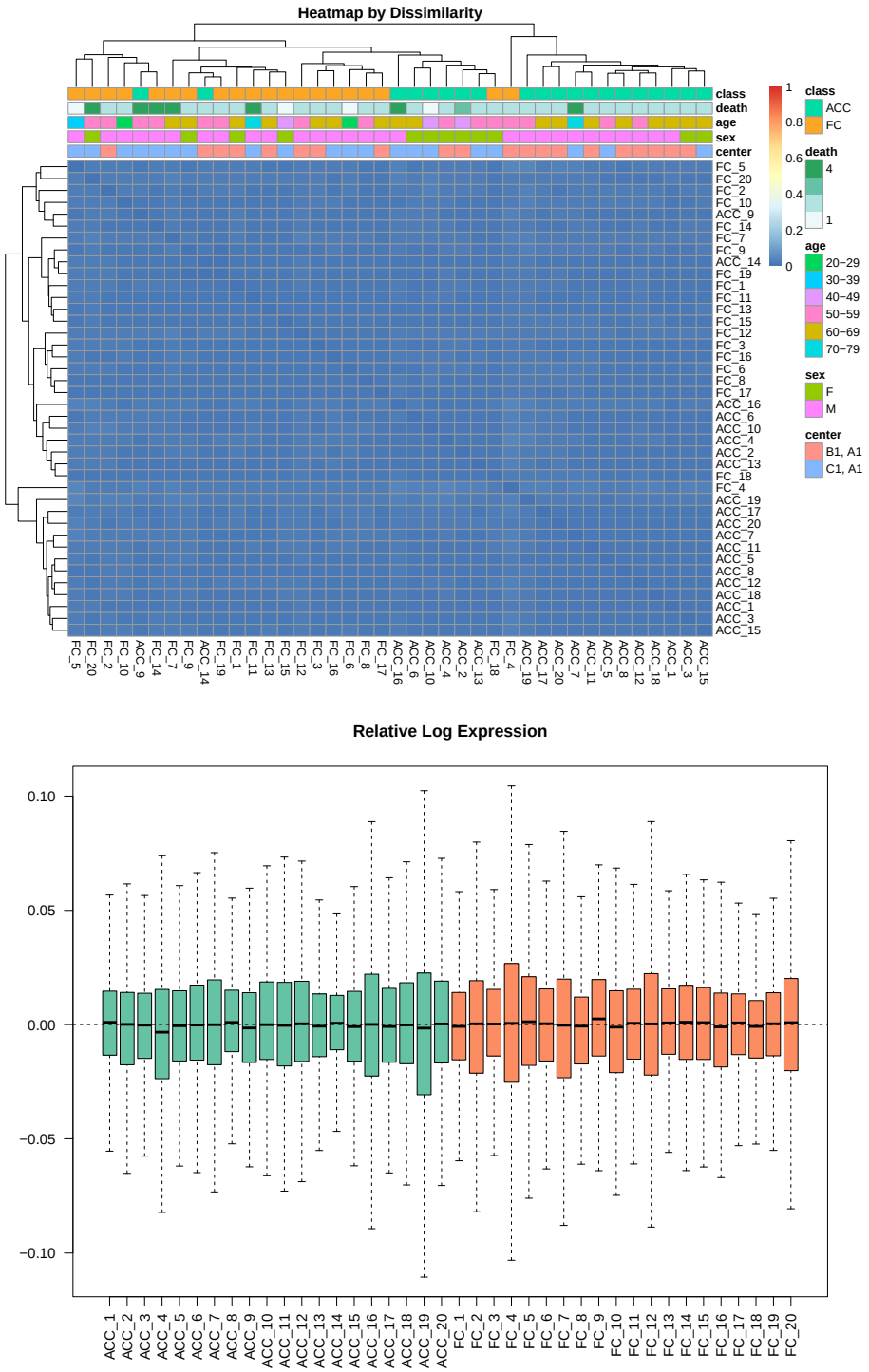


Figure 6: Heatmap and RLE. Heatmap (upper panel): colors in heatmap highlight the distance matrix, obtained by Spearman's correlation metric: color gradient ranges from *dark green*, meaning 'minimum distance' (*i.e.* dissimilarity = 0, correlation = 1), to *light green green*. On the top of heatmap, 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).

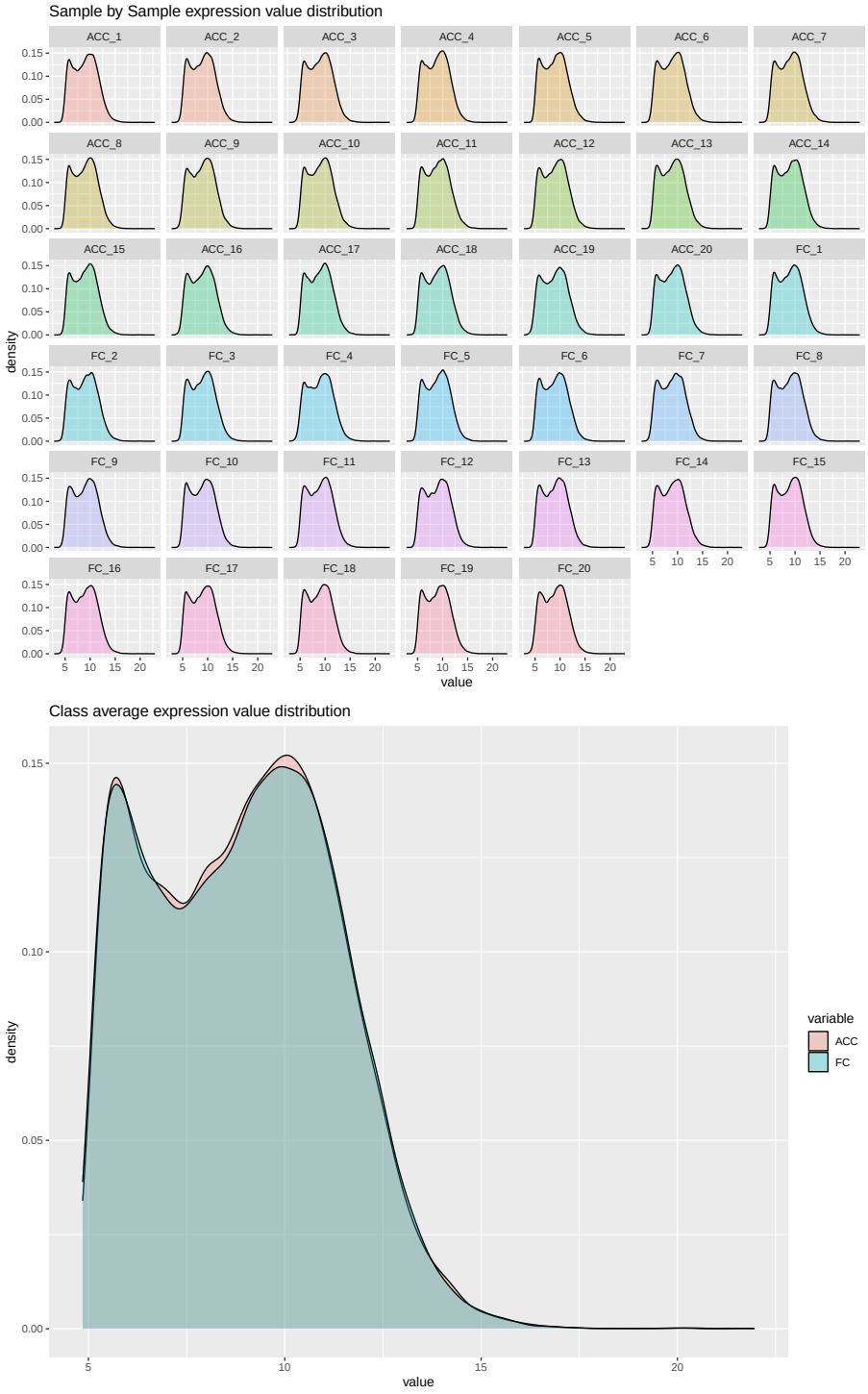


Figure 9: Gene Expression distribution. Sample-by-Sample expression distribution (upper panel) helps user to find outliers and to control the effect of normalization, filtering and adjusting steps; class average expression distribution (lower panel) highlights global expression differences between classes.

3.6 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")
```

4 Two specific supervised machine learning workflows

As we described in the previuos sections, RNA-Seq experiments, are used to generate hundreds to thousand of features at once. However, most of them are non informative to discriminate phenotypes and useless for further investigations.
In this context, supervised machine learning is a powerful tool that gathers several algorithms, to select the most informative features in high-dimensional data and design accurate prediction models. To achieve these aims, the supervised learning algorithms need 'labeled data', where each observation of the dataset comes with *a priori* knowledge of the class membership.
Since version 2.0.0 of the software, *DaMiRseq* offers a solution to solve two distinct problems, in supervised learning analysis: (i) finding a small set of robust features, and (ii) building the most reliable model to predict new samples.

- **Finding a small set of robust features to discriminate classes.**
  This task seeks to select and assess the reliability of a feature set from high-dimensional data. Specifically, we first implemented a 4-step feature selection strategy (orange box in Figure 10, panel A), in order to get the most relevant features. Then, we tested the robustness of the selected features by performing a bootstrap strategy, in which an 'ensemble learner' classifier is built for each iteration (green box in Figure 10, panel A). In Section 4.1, we described this analysis in details.

- **Building the most reliable model to predict new samples.**
  An important goal in machine learning is to develop a mathematical model, able to correctly associate each observation to the corresponding class. This model, also known as classification or prediction model (Figure 10, panel B), aimed at ensuring the highest prediction accuracy with as few features as possible.
  First, several different models are generate by iteratively (i) splitting data in training and validation sets; (ii) performing feature selection on the training set (orange box in Figure 10, panel B); (iii) building a classification model on the training set (pink box in Figure 10, panel B); and, (iv) testing the classification model on the validation set (purple box in Figure 10, panel B). Finally, taking into account the performance of all generated models, the most reliable one is selected (red box in Figure 10, panel B). This model will be used for any further prediction on independent test sets (light blue box in Figure 10, panel B). We will refer to this model as 'optimal model'.
  In Section 4.2, we thoroughly described how to perform this analysis.

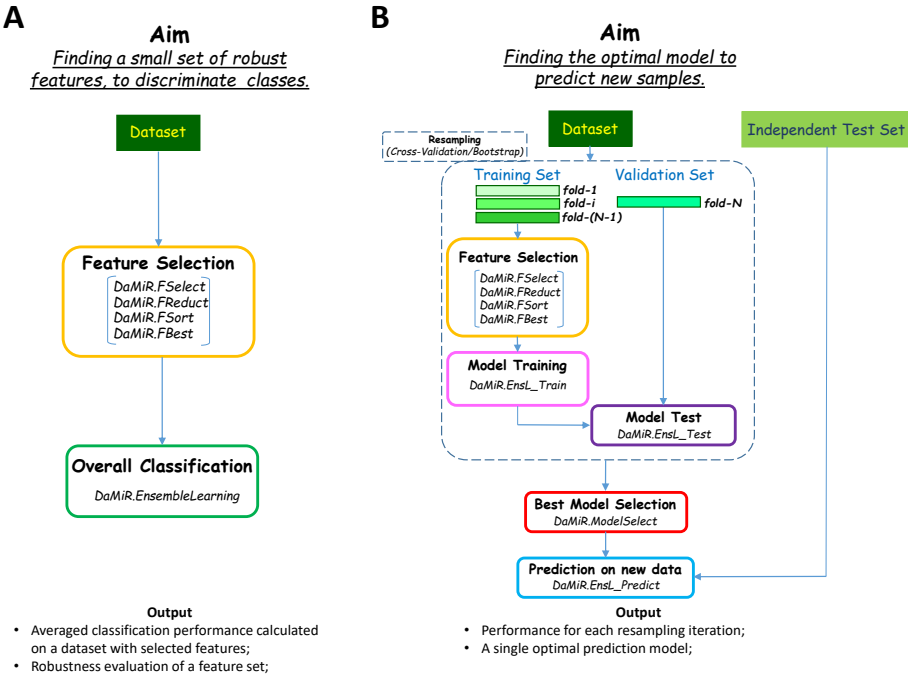


Figure 10: The DaMiRseq machine learning workflows. Each elliptic box represents a specific step, where the aims and the corresponding functions are specified. In panel A, we provided the workflow to find a small set of informative features, described in Section 4.1. In panel B, we provided the workflow to find the best prediction model, described in Section 4.2.

4.1 Finding a small set of informative, robust features

This Section, where we will describe how to get a small set of robust features from an RNA-Seq dataset, is organized in two parts: in Section 4.1.1, all the feature selection steps and the corresponding functions are reported in detail; while, in Section 4.1.2 we will focus on the classification step that we performed for assessing the robustness of the feature set. Mathematical details about the classifier implementation are also provided.

4.1.1 Feature Selection

The steps implemented in the Section 3 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.

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 [18]. 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 [19].
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)

## You are performing feature selection on a binary class object.
## 19049 Genes have been discarded for classification 294 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.

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)

## 66 Highly correlated features have been discarded for classification.
##  228 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.

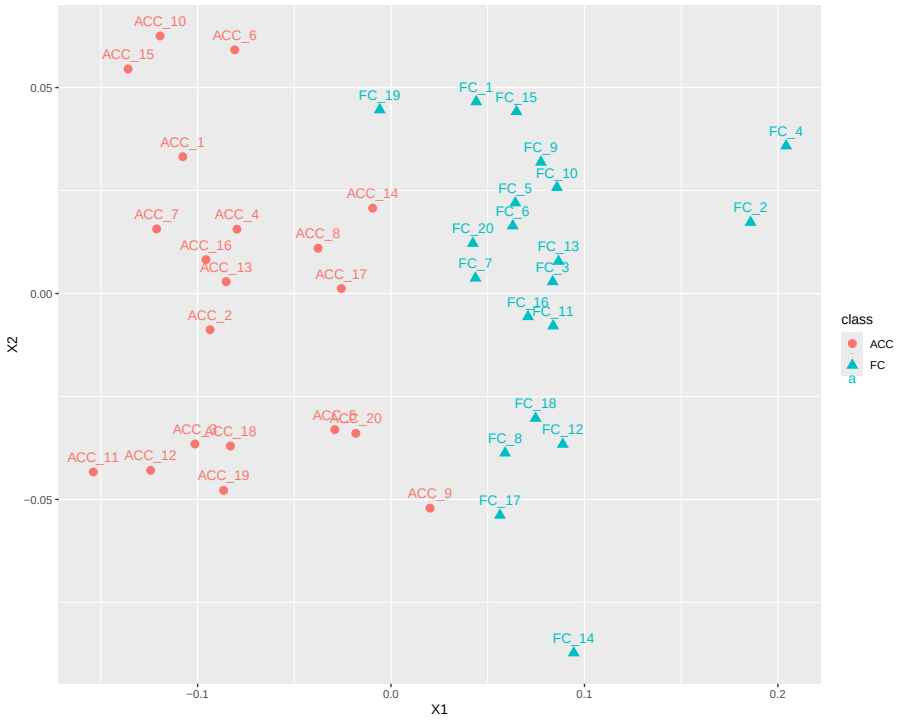


Figure 11: MultiDimentional Scaling plot. A MDS plot is drawn, considering only most informative genes, obtained after feature selection: color code is referred to 'class'.

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) [20, 21]. 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 43 seconds (i.e. about 1 minutes).

  head(df.importance)

##                  RReliefF scaled.RReliefF
## ENSG00000164326 0.3225273        3.517257
```

```
## ENSG00000140015 0.3106638        3.343049
## ENSG00000131378 0.2598659        2.597116
## ENSG00000151892 0.2596664        2.594186
## ENSG00000137699 0.2584854        2.576844
## ENSG00000258754 0.2504441        2.458762
```

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" "ENSG00000140015" "ENSG00000131378" "ENSG00000151892"
## [5] "ENSG00000137699"

  # 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. Further information about the 'feature selection' step and comparison with other methods can be found in the Supplementary Article Data by *Chiesa et al.* [12] with an example code. In adition, an example of the effective use of our feature selection process has been obtained for the detection of P. aeruginosa transcriptional signature of Human Infection in *Cornforth et al.* [22].

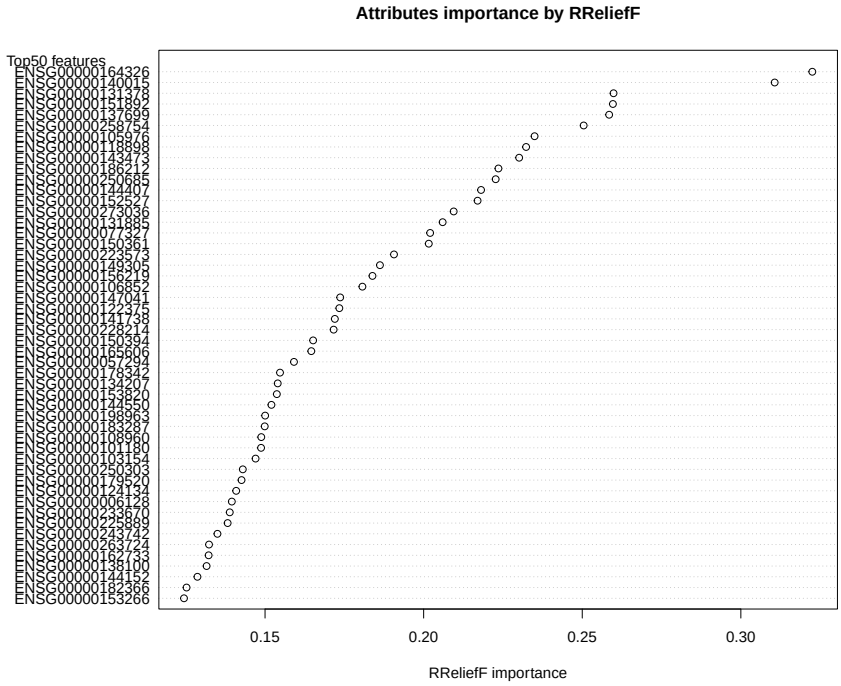


Figure 12: Feature Importance Plot. The dotchart shows the list of top 50 genes, sorted by RReliefF importance score. This plot may be used to select the most important predictors to be used for classification.

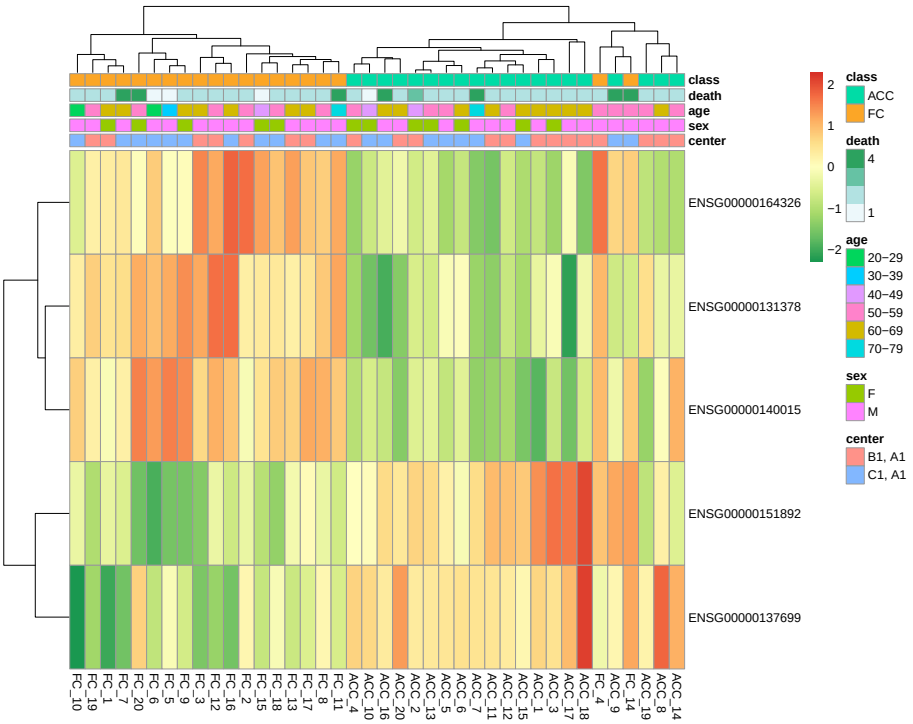


Figure 13: Clustergram. The clustergram is generated by using the expression values of the 5 predictors selected by `DaMiR.FBest` function. As for the heatmap generated by `DaMiR.Allplot`, 'class' and covariates are drawn as horizontal and color coded bars.

4.1.2 Classification

All the steps executed so far allowed the 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 [23]. 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:

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

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

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

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:

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

$Pr_{(k,Ensemble)}$ 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.

The above mentioned procedure is implemented in the `DaMiR.EnsembleLearning` function, where `fSample.tr`, `fSample.tr.w` and `iter` arguments allow the algorithm tuning.
This function performs a Bootstrap resampling strategy with `iter` iterations, in which several meta-classifiers are built and tested, by generating `iter` training sets and `iter` test sets in a random way. Then each classification metrics (acc,sen) is calculated on the `iter` test sets. Finally, the average performance (and standard deviation) is provided (text and violin plots). 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. 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, PPV, NPV and the Matthew's Correlation Coefficient (MCC). These objects can be accessed using the $ accessor.

```
  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: 0.95 0.95 0.97 0.94 0.91 0.96 0.95
##  St.Dev. 0.03 0.04 0.03 0.04 0.08 0.02 0.04
## MCC score:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.91 0.9 0.93 0.89 0.83 0.92 0.9
##  St.Dev. 0.06 0.07 0.06 0.08 0.15 0.05 0.07
## Specificity:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.96 0.96 0.98 0.91 0.91 0.96 0.94
##  St.Dev. 0.05 0.05 0.04 0.07 0.09 0.05 0.06
## Sensitivity:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.96 0.95 0.96 0.98 0.93 0.96 0.96
##  St.Dev. 0.05 0.05 0.05 0.04 0.09 0.05 0.05
## PPV:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.95 0.95 0.98 0.9 0.9 0.96 0.94
##  St.Dev. 0.06 0.05 0.05 0.09 0.1 0.05 0.06
## NPV:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.95 0.94 0.96 0.98 0.93 0.96 0.96
##  St.Dev. 0.06 0.06 0.06 0.05 0.1 0.05 0.06
```

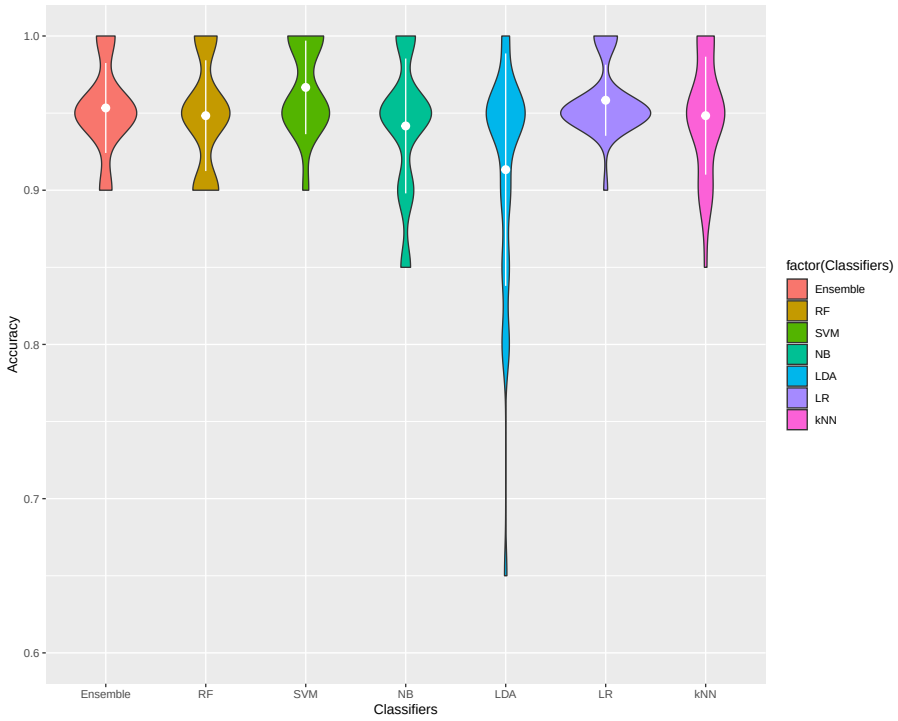


Figure 14: Accuracies Comparison. The violin plot highlights the classification accuracy of each classifier, computed at each iteration; a black dot represents a specific accuracy value while the shape of each "violin" is drawn by a Gaussian kernel density estimation. Averaged accuracies and standard deviations are represented by white dots and lines.

As shown in Figure 14 almost all single, weak classifiers show high or very high classification performancies, in terms of accuracy, specificity, sensitivity and MCC.

Figure 14 highlights that the five selected features ensured high and reproducible performance, whatever the classifier; indeed, the average accuracy wass always greater than 90% and pperformance deviated by no more than 4% from the mean value.

4.2 Building the optimal prediction model

In this section, we present the workflow to generate an effective classification model that can be later used to predict the class membership of new samples. Basically, *DaMiRseq* implements a supervised learning procedure, depicted in Figure 10, panel B, where each step is performed by a specific function.
Indeed, the `DaMiR.EnsL_Train` and the `DaMiR.EnsL_Test` functions allow training and testing one single model at once, respectively.
The `DaMiR.ModelSelect` function implements six strategies to perform the model selection, taking into account a particular classification metrics (e.g., Accuracy) along with the number of predictors. The idea behind this function is to search for the most reliable model rather than the best ever; this should help to avoid over-fitted training data, leading to poor performance in further predictions.
Users can choose a specific strategy, combining one of the three value of the `type.sel` argument (`type.sel = c("mode", "median", "greater")`) and one of the two values of the

`npred.sel` argument (`npred.sel = c("min", "rnd")`). Let us assume we generated N different model during a resampling strategy, each one characterized by a certain accuracy and number of selected features. Then, the combination of:

- `type.sel = "mode"` and `npred.sel = "min"`, will select the model with minimum number of features, among those with the accuracy equal to the mode (*i.e.*, the most frequent value) of all N accuracies;
- `type.sel = "mode"` and `npred.sel = "rnd"`, will select randomly one model, among those with the accuracy equal to the mode (*i.e.*, the most frequent value) of all N accuracies;
- `type.sel = "median"` and `npred.sel = "min"`, will select the model with minimum number of features, among those with the accuracy equal to the median of all N accuracies;
- `type.sel = "median"` and `npred.sel = "rnd"`, will select randomly one model, among those with the accuracy equal to the median of all N accuracies;
- `type.sel = "greater"` and `npred.sel = "min"`, will select the model with minimum number of features, among those with the accuracy greater than a fixed value, specified by `th.sel`;
- `type.sel = "greater"` and `npred.sel = "rnd"`, will select randomly one model, among those with the accuracy greater than a fixed value, specified by `th.sel`;

Finally, the `DaMiR.EnsL_Predict` function allows performing the class prediction of new samples.
Note. Currently, `DaMiR.EnsL_Train`, `DaMiR.EnsL_Test` and `DaMiR.EnsL_Predict` work only on binary classification problems.

4.2.1 Training and testing inside the cross-validation

In order to simulate a typical genome-wide setting, we performed this analysis on the `data_adjust` dataset, which is composed of 40 samples (20 ACC and 20 FC) and 19343 features.
First, we randomly selected 5 ACC and 5 FC samples (`Test_set`), which will be later used for the final prediction step. The remaining samples composed the case study dataset (`Learning_set`).

```
# Dataset for prediction
set.seed(10101)
nSampl_cl1 <- 5
nSampl_cl2 <- 5

## May create unbalanced Learning and Test sets
# idx_test <- sample(1:ncol(data_adjust), 10)

# Create balanced Learning and Test sets
idx_test_cl1<-sample(1:(ncol(data_adjust)/2), nSampl_cl1)
idx_test_cl2<-sample(1:(ncol(data_adjust)/2), nSampl_cl2) + ncol(data_adjust)/2
idx_test <- c(idx_test_cl1, idx_test_cl2)

Test_set <- data_adjust[, idx_test, drop=FALSE]
Learning_set <- data_adjust[, -idx_test, drop=FALSE]
```

Then, we implemented a 3-fold Cross Validation, as resampling strategy. Please, note that this choice is an unsuitable setting in every real machine learning analysis; therefore, we strongly recommend to adopt more effective resampling strategies, as bootstrap632 or 10-fold cross validation, to obtain more accurate results.

```
# Training and Test into a 'nfold' Cross Validation
nfold <- 3
cv_sample <- c(rep(seq_len(nfold), each=ncol(Learning_set)/(2*nfold)),
               rep(seq_len(nfold), each=ncol(Learning_set)/(2*nfold)))

# Variables initialization
cv_models <- list()
cv_predictors <- list()
res_df <- data.frame(matrix(nrow = nfold, ncol = 7))
colnames(res_df) <- c("Accuracy",
                      "N.predictors",
                      "MCC",
                      "sensitivity",
                      "Specificty",
                      "PPV",
                      "NPV")
```

For each iteration, we (i) split the dataset in training (`TR_set`) and validation set (`Val_set`); (ii) performed the features selection and built the model (`ensl_model`) on the training set; and, (iii) tested and evaluated the model on the validation set (`res_Val`). Regarding the feature selection, we used the *DaMiRseq* procedure, described in Section 4.1.1; however, any other feature selection strategies can be jointly utilized, such as the *GARS* package.

```
for (cv_fold in seq_len(nfold)){

  # Create Training and Validation Sets
  idx_cv <- which(cv_sample != cv_fold)
  TR_set <- Learning_set[,idx_cv, drop=FALSE]
  Val_set <- Learning_set[,-idx_cv, drop=FALSE]

  #### Feature selection
  data_reduced <- DaMiR.FSelect(t(assay(TR_set)),
                                as.data.frame(colData(TR_set)),
                                th.corr=0.4)
  data_reduced <- DaMiR.FReduct(data_reduced$data,th.corr = 0.9)
  df_importance <- DaMiR.FSort(data_reduced,
                               as.data.frame(colData(TR_set)))
  selected_features <- DaMiR.FBest(data_reduced,
                                   ranking=df_importance,
                                   autoselect = "yes")
  # update datasets
  TR_set <- TR_set[selected_features$predictors,, drop=FALSE]
  Val_set <- Val_set[selected_features$predictors,drop=FALSE]

  ### Model building
  ensl_model <- DaMiR.EnsL_Train(TR_set,
```

```
                                    cl_type = c("RF", "LR"))
  # Store all trained models
  cv_models[[cv_fold]] <- ensl_model

  ### Model testing
  res_Val <- DaMiR.EnsL_Test(Val_set,
                             EnsL_model = ensl_model)

  # Store all ML results
  res_df[cv_fold,1] <- res_Val$accuracy[1] # Accuracy
  res_df[cv_fold,2] <- length(res_Val$predictors) # N. of predictors
  res_df[cv_fold,3] <- res_Val$MCC[1]
  res_df[cv_fold,4] <- res_Val$sensitivity[1]
  res_df[cv_fold,5] <- res_Val$Specificty[1]
  res_df[cv_fold,6] <- res_Val$PPV[1]
  res_df[cv_fold,7] <- res_Val$NPV[1]

  cv_predictors[[cv_fold]] <- res_Val$predictors
}
```

4.2.2 Selection and Prediction

Finally, we searched for the 'optimal model' to be used for predicting new samples. In this simulation, we set `type.sel = "mode"` and `npred.sel = "min"` to select the model with the lowest number of seleted features ensuring the most frequent accuracy value.

```
# Model Selection
res_df[,1:5]

idx_best_model <- DaMiR.ModelSelect(res_df,
                                    type.sel = "mode",
                                    npred.sel = "min")
```

The selected model (the red cross in Figure **??**) reached an accuracy greater than 90% with less than 10 predictors(out of 19183) on its validation set, enabling to predict correctly all the samples composing the independent test set. In addition, subsetting the `cv_predictors` object will return the predictors of the optimal model.

```
# Prediction on the the independent test set
res_predict <- DaMiR.EnsL_Predict(Test_set,
                  bestModel = cv_models[[idx_best_model]])

# Predictors
cv_predictors[[idx_best_model]]

# Prediction assessment for Ensemble learning
id_classifier <- 1 # Ensemble Learning
table(colData(Test_set)$class, res_predict[,id_classifier])

# Prediction assessment for Logistic regression
id_classifier <- 3 # Logistic regression
```

```
table(colData(Test_set)$class, res_predict[,id_classifier])
```

5 Normalizing and Adjusting real independent test sets

The main issue when dealing with prediction in a high-dimensional context is how to normalize and adjust new sample data. The problem arises because (i) normalization procedures are data dependent and (ii) factor-based algorithms to adjust data are supervised methods, i.e. we must know the variable of interest we wish to adjust for. But this is not the case of novel samples for which we are not supposed to know their class. In this latter case, however, we can apply normalization and adjustment on new data by making use of the knowledge obtained from the learning set.
Hereafter, we will propose two methods, called **'Precise'** and **'Quick'**, that demonstrate to work satisfactory displaying high normalization capability, leading to good prediction accuracy on new samples. Briefly, with the 'Precise' method the dispersion function is estimated on the Learning set; while with the 'Quick' methods,dispersion and data transformation is performed directly (and more quickly) on the independent test set.
To illustrate these two options, we exploited the example data of DaMiRseq package by splitting raw counts at the beginning and using the test set as it would be a completely independent data set. We encourage the user to use also other kind of data to test our proposed choices.

```
data(SE)
# create Independent test set and Learning set (raw counts)
idx_test <- c(18,19,39,40)
Ind_Test_set <- SE[, idx_test, drop=FALSE]
Learning_set <- SE[, -idx_test, drop=FALSE]

# DaMiRseq pipeline on Learning Set
data_norm <- DaMiR.normalization(Learning_set,
                                 minCounts=10,
                                 fSample=0.7,
                                 hyper = "yes",
                                 th.cv=3)

## 1919 Features have been filtered out by espression. 19444 Features remained.
## 14 'Hypervariant' Features have been filtered out. 19430 Features remained.
## Performing Normalization by 'vst' with dispersion parameter:  parametric

sv <- DaMiR.SV(data_norm)

## The number of SVs identified, which explain 95 % of Variance, is: 4

data_adjust <- DaMiR.SVadjust(data_norm, sv, n.sv=4)
```

Then, we normalized the independent test set by the `vst` (the same normalization used for the Learning set) and the `precise` method. Then, we have also adjusted for batch effects taking advantage of `data_adjust`, the dataset previously corrected by the estimated SVs.

```
# remove not expressed genes from Learning_set and Ind_Test_set
expr_LearningSet <- Learning_set[rownames(data_norm)]
```

```
expr_Ind_Test_set <- Ind_Test_set[rownames(data_norm)]

# Independent test set Normalization
norm_ind_ts <- DaMiR.iTSnorm(expr_LearningSet,
                             expr_Ind_Test_set,
                             normtype = "vst",
                             method = "precise")

## You selected the vst normalization and the precise method.

# Independent test set batch Adjusting
adj_norm_ind_ts <- DaMiR.iTSadjust(data_adjust, norm_ind_ts)
```

In Figure 15 and in Figure 16, the effect of normalization and batch correction, applied on the independent test set, are shown, respectively

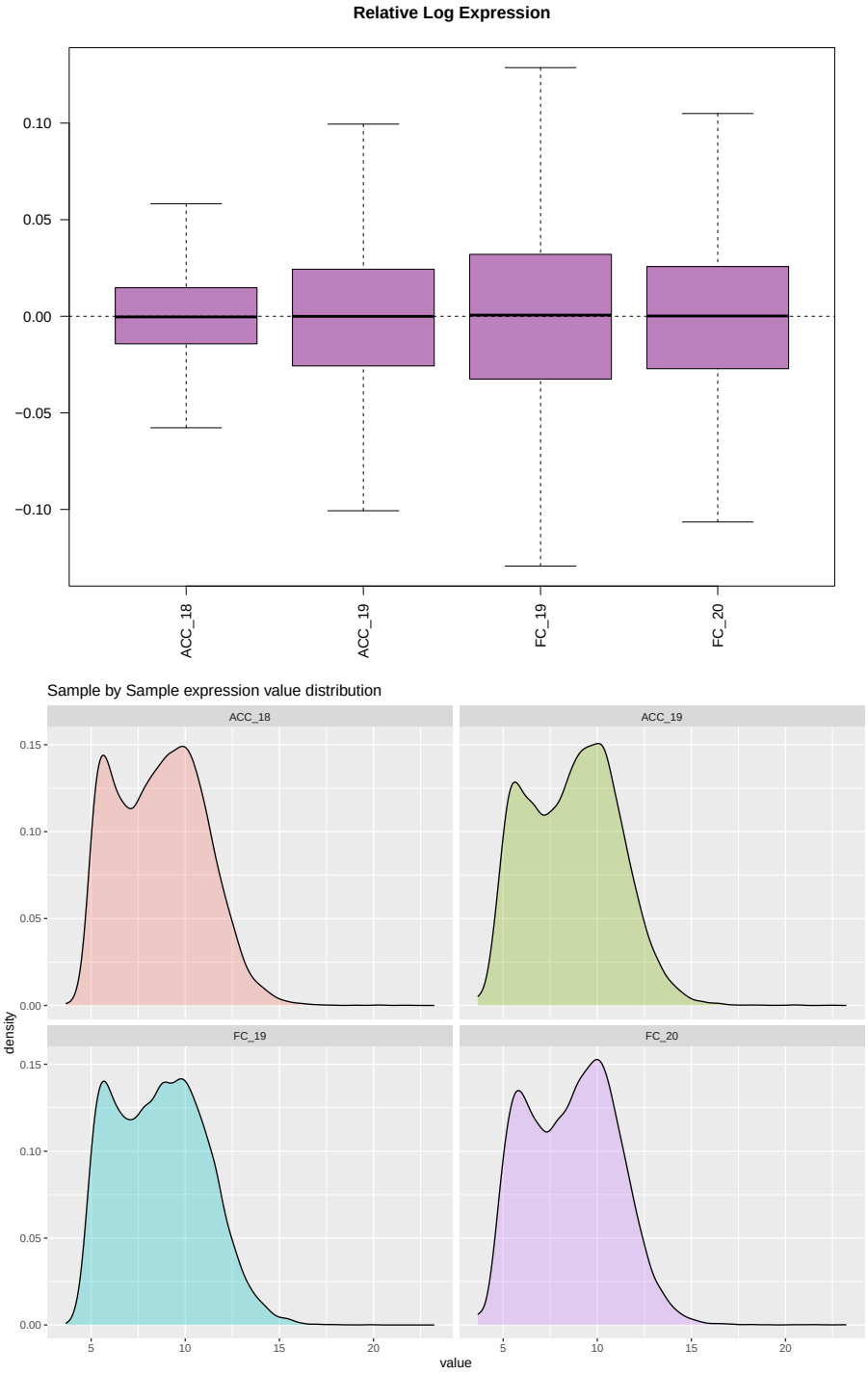


Figure 15: RLE and Gene Expression distribution after normalization. A RLE boxplot of the distribution of expression values computed as the difference between the expression of each gene and the median expression of that gene accross all samples. Here, since all medians are very close to zero, it appears that all the samples are well-normalized and do not present any quality problems (upper panel). Sample-by-Sample expression distribution (lower panel)

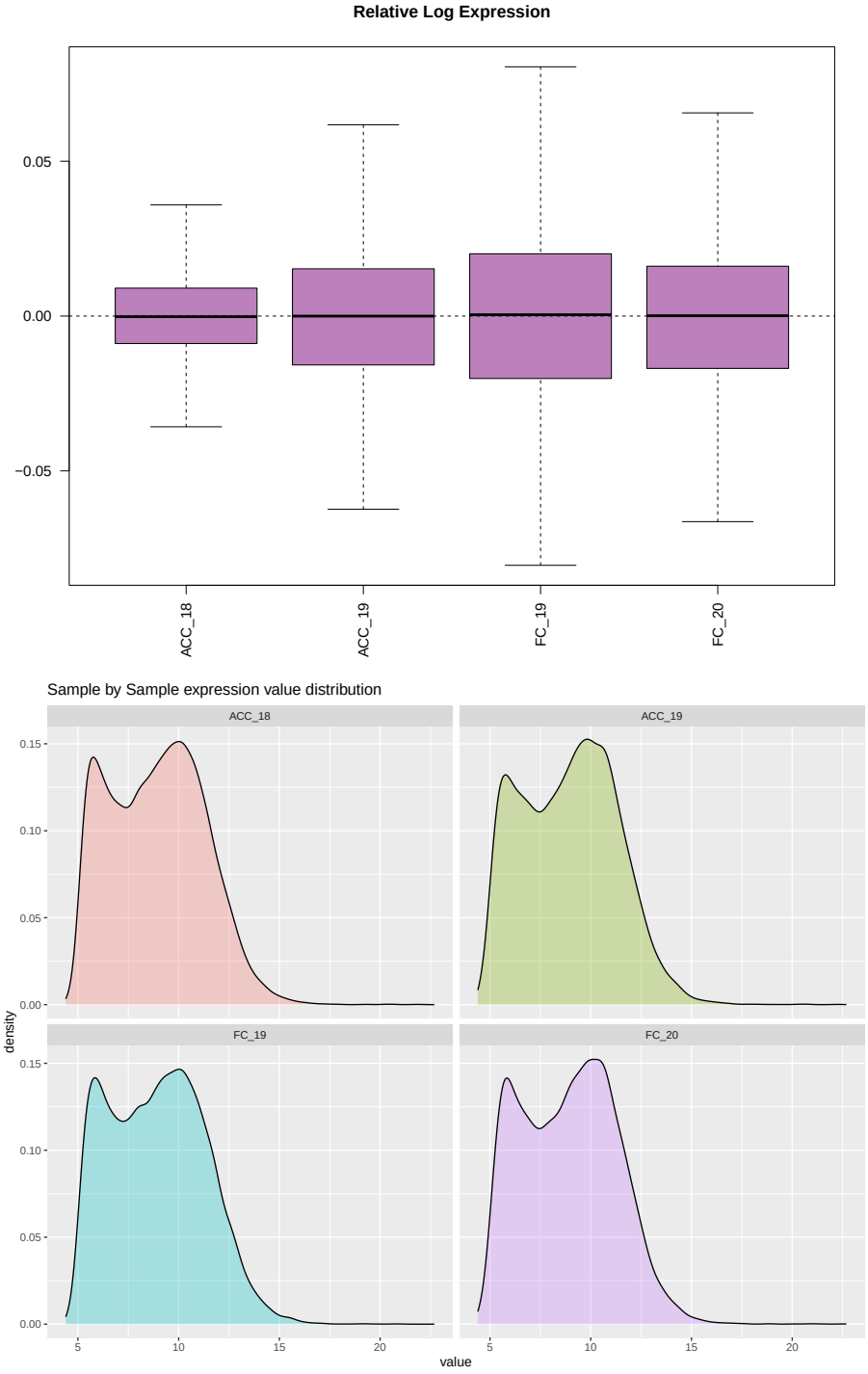


Figure 16: RLE and Gene Expression distribution after batch correction. A RLE boxplot of the distribution of expression values computed as the difference between the expression of each gene and the median expression of that gene accross all samples. Here, since all medians are very close to zero, it appears that all the samples are well-normalized and do not present any quality problems (upper panel). Sample-by-Sample expression distribution (lower panel)

Finally, users can predict the class of the independent test set samples and assessed the performance, by directly using the `DaMiR.EnsL_Predict` function and the best model built on the Learning Set. Here, for simplicity, we used the model built on the workflow implemented in Section 4.2.

```
# Prediction on independent test set

prediction <- DaMiR.EnsL_Predict(t(adj_norm_ind_ts),
                                 bestModel = cv_models[[idx_best_model]])
prediction

# confusion matrix for the Ensemble Learner
table(Ind_Test_set@colData$class, prediction[,1])
```

6 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 3.4 could be skipped if we assume that the data are not affected by any batches (known or not), 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 3 (*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)

## You are performing feature selection on a binary class object.
## 19238 Genes have been discarded for classification 105 Genes remained.

data_reduced_2 <- DaMiR.FReduct(data_reduced_2$data)

## 31 Highly correlated features have been discarded for classification.
##  74 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 0.3793656        3.884531
## ENSG00000258754 0.2665921        2.439328
## ENSG00000169432 0.2602079        2.357513
## ENSG00000105976 0.2432797        2.140576
## ENSG00000077327 0.2092149        1.704032
## ENSG00000170290 0.1932171        1.499019

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" "ENSG00000169432" "ENSG00000105976"
## [5] "ENSG00000077327"

## 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: 0.96 0.94 0.96 0.94 0.87 0.95 0.91
##  St.Dev. 0.04 0.05 0.04 0.04 0.11 0.04 0.06
## MCC score:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.92 0.88 0.93 0.9 0.75 0.91 0.83
##  St.Dev. 0.07 0.1 0.07 0.08 0.21 0.08 0.12
## Specificity:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.98 0.96 0.99 0.96 0.89 0.98 0.93
##  St.Dev. 0.04 0.07 0.04 0.06 0.12 0.04 0.08
## Sensitivity:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.94 0.93 0.95 0.94 0.88 0.93 0.91
##  St.Dev. 0.06 0.06 0.06 0.07 0.12 0.07 0.08
## PPV:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.98 0.96 0.99 0.96 0.88 0.98 0.93
##  St.Dev. 0.05 0.08 0.04 0.07 0.14 0.05 0.09
## NPV:
##  Ensemble RF SVM NB LDA LR kNN
##  Mean: 0.93 0.92 0.94 0.93 0.86 0.92 0.9
```

```
##  St.Dev. 0.07 0.07 0.07 0.08 0.17 0.08 0.1
```

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, felt down around 90% of accuracy for all the classifiers.

Figure 17 shows the results of the variation to standard workflow of *DaMiRseq*, proposed in this Section. Taking as reference the "Standard Workflow", described in Section 3, we can observe that the performances significantly decrease.

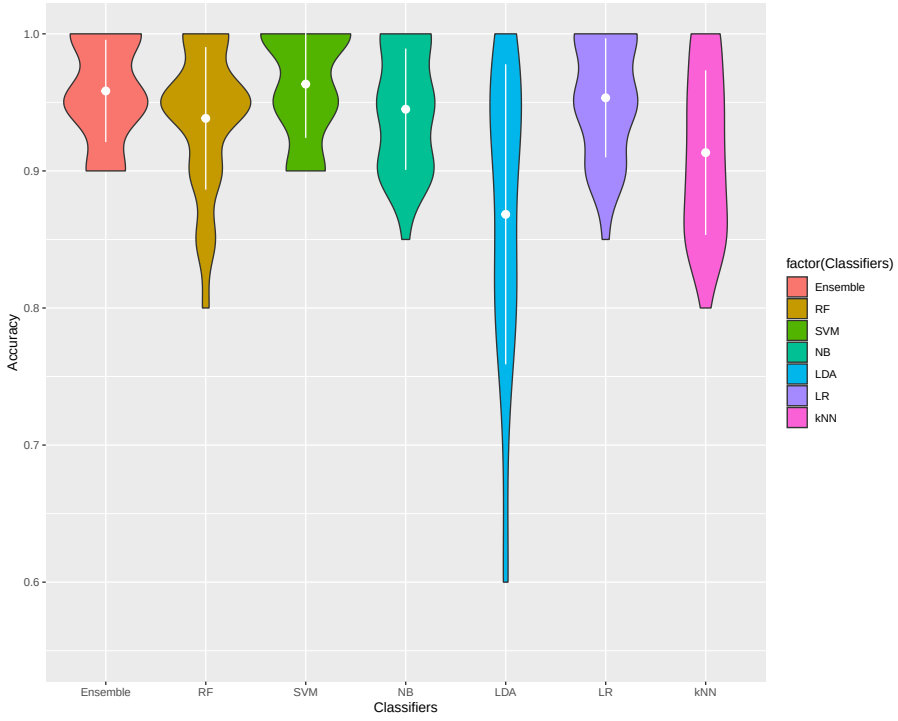


Figure 17: Accuracies Comparison. The violin plot shows the effect of the modification to *DaMiRseq* standard workflow, described in Section 6: without adjusting data (following the steps described in Section 3.4), performances usually decrease; this could be explained by the fact that some noise, probably coming from unknown source of variation, is present in the dataset.

7 Check new implementations!

7.1 Version 2.0.0, devel: 2.1.0

Since version 2.0.0 of the software, *DaMiRseq* offers a solution to solve two distinct problems, in supervised learning analysis: (i) finding a small set of robust features, and (ii) building the most reliable model to predict new samples

Relevant modifications:

- Since version 2.0.0 of the software, *DaMiRseq* offers a solution to solve two distinct problems, in supervised learning analysis: (i) finding a small set of robust features, and (ii) building the most reliable model to predict new samples;
- The functions `DaMiR.EnsembleLearning2cl_Training`, `EnsembleLearning2cl_Test` and `EnsembleLearning2cl_Predict` were deprecated and replaced by `DaMiR.EnsL_Train`, `DaMiR.EnsL_Test` and `DaMiR.EnsL_Predict`, respectively;
- We have created a new function (`DaMiR.ModelSelect`) to select the best model in a machine learning analysis;
- We have created two new functions (`DaMiR.iTSnorm` and `DaMiR.iTSadjust`) to normalize and adjust the gene espression of independent test sets;
- Two types of expression value distribution plot were added to the `DaMiR.Allplot` function.

Minor modifications and bugs fixed:

- Now, the `DaMiR.FSelect` function properly handles multi factorial experimental settings;
- The `DaMiR.FBest` function cannot select less than 2 predictors, whatever the mode;
- The axes labels in the RLE plot (`DaMiR.Allplot` function) are better oriented.

7.2 Version 1.6, devel: 1.5.2

Relevant modifications:

- The `DaMiR.normalization` embeds also the 'logcpm' normalization, implemented in the *edgeR* package.
- Now, `DaMiR.EnsembleLearning` calculates also the Positive Predicted Values (PPV) and the Negative Predicted Values (NPV);
- Three new functions have been implemented for the binary classification task: `DaMiR.EnsembleLearning2cl_Training`, `DaMiR.EnsembleLearning2cl_Test` and `DaMiR.EnsembleLearning2cl_Predict`. The first one allows the user to implement the training task and to select the model with the highest accuracy or the average accuracy; the second function allows the user to test the selected classification model on a test set defined by the user; the last function allows the user to predict the class of new samples.

Minor modifications and bugs fixed:

- Removed black dots in the violin plots.

7.3 Version 1.4.1

- Adjusted Sensitivity and Specificity calculations.

7.4 Version 1.4

Relevant modifications:

- *DaMiRseq* performs both binary and multi-class classification analysis;
- 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` function (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 disabled 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.

8 Session Info

- R version 4.6.2 (2026-10-29), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=en_US.UTF-8`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=en_US.UTF-8`, `LC_ADDRESS=en_US.UTF-8`, `LC_TELEPHONE=en_US.UTF-8`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=en_US.UTF-8`
- Time zone: `Etc/UTC`
- TZcode source: `system (glibc)`
- Running under: `Ubuntu 26.04.1 LTS`
- Matrix products: default

- BLAS: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3`
- LAPACK: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.32.so` ; LAPACK version3.12.0
- Base packages: base, datasets, graphics, grDevices, methods, stats, stats4, utils
- Other packages: Biobase 2.72.0, BiocGenerics 0.58.1, caret 7.0-1, DaMiRseq 2.24.0, generics 0.1.4, GenomicRanges 1.64.0, ggplot2 4.0.3, IRanges 2.46.0, knitr 1.52, lattice 0.23-1, MatrixGenerics 1.24.0, matrixStats 1.5.0, S4Vectors 0.50.3, Seqinfo 1.2.0, SummarizedExperiment 1.42.0
- Loaded via a namespace (and not attached): abind 1.4-8, annotate 1.90.0, AnnotationDbi 1.74.0, arm 1.15-3, aroma.light 3.42.0, backports 1.5.1, base64enc 0.1-6, bdsmatrix 1.3-7, BiocFileCache 3.2.0, BiocIO 1.22.0, BiocManager 1.30.27, BiocParallel 1.46.0, BiocStyle 2.40.0, biomaRt 2.68.6, Biostrings 2.80.2, bit 4.6.0, bit64 4.8.6, bitops 1.1-0, blob 1.3.0, boot 1.3-32, buildtools 1.0.0, cachem 1.1.0, checkmate 2.3.4, cigarillo 1.2.1, class 7.3-24, cli 3.6.6, cluster 2.1.8.3, coda 0.19-4.1, codetools 0.2-20, colorspace 2.1-4, compiler 4.6.2, corrplot 0.95, crayon 1.5.3, curl 8.1.0, data.table 1.18.6.1, DBI 1.3.0, dbplyr 2.6.0, DelayedArray 0.38.2, deldir 2.0-4, DESeq2 1.52.0, digest 0.6.39, dplyr 1.2.1, DT 0.34.0, e1071 1.7-17, EDASeq 2.46.0, edgeR 4.10.7, emmeans 2.0.4, entropy 1.3.2, estimability 2.0.0, evaluate 1.0.5, FactoMineR 2.18, farver 2.1.2, fastmap 1.2.0, filelock 1.0.3, flashClust 1.1-4, foreach 1.5.2, foreign 0.8-92, Formula 1.2-6, FSelector 0.34, future 1.82.0, future.apply 1.21.0, genalg 0.2.1, genefilter 1.94.0, GenomicAlignments 1.48.0, GenomicFeatures 1.64.0, ggrepel 0.9.8, globals 0.20.0, glue 1.8.1, gower 1.0.2, grid 4.6.2, gridExtra 2.3.1, gtable 0.3.6, hardhat 1.4.3, highr 0.12, Hmisc 5.3-0, hms 1.1.4, htmlTable 2.5.0, htmltools 0.5.9, htmlwidgets 1.6.4, httr 1.4.9, httr2 1.3.0, hwriter 1.3.2.1, igraph 2.3.4, ineq 0.2-13, interp 1.1-6, ipred 0.9-16, irlba 2.4.1, iterators 1.0.14, jpeg 0.1-11, KEGGREST 1.52.2, kknn 1.4.1, labeling 0.4.3, latticeExtra 0.6-31, lava 1.11, leaps 3.2, lifecycle 1.0.5, limma 3.68.5, listenv 1.1.0, lme4 2.1-0, locfit 1.5-9.12, lubridate 1.9.5, magrittr 2.0.5, maketools 1.3.2, MASS 7.3-66, Matrix 1.7-6, memoise 2.0.1, mgcv 1.9-5, minqa 1.2.8, ModelMetrics 1.2.2.2, multcompView 0.1-12, mvtnorm 1.4-2, nlme 3.1-172, nloptr 2.2.1, nnet 7.3-21, otel 0.3.0, parallel 4.6.2, parallelly 1.50.0, pheatmap 1.0.13, pillar 1.11.1, pkgconfig 2.0.3, pls 2.9-0, plsVarSel 0.10.0, plyr 1.8.9, png 0.1-9, praznik 13.0.0, prettyunits 1.2.0, pROC 1.19.1, prodlim 2026.03.11, progress 1.2.3, proxy 0.4-29, purrr 1.2.2, pwalign 1.8.0, R.methodsS3 1.8.2, R.oo 1.27.1, R.utils 2.13.0, R6 2.6.1, randomForest 4.7-1.2, rbibutils 2.4.1, RColorBrewer 1.1-3, Rcpp 1.1.2, RCurl 1.98-1.20, Rdpack 2.6.6, recipes 1.4.0, reformulas 0.4.4, reshape2 1.4.5, restfulr 0.0.17, rJava 1.0-18, rjson 0.2.23, rlang 1.3.0, rmarkdown 2.32, rpart 4.1.27, Rsamtools 2.28.0, RSNNS 0.4-18, RSQLite 3.53.3, rstudioapi 0.19.0, rtracklayer 1.72.0, RWeka 0.4-50, RWekajars 3.9.3-2, S4Arrays 1.12.1, S7 0.2.2, scales 1.4.0, scatterplot3d 0.3-45, ShortRead 1.70.0, showtext 0.9-8, showtextdb 3.0, SparseArray 1.12.3, splines 4.6.2, statmod 1.5.2, stringi 1.8.9, stringr 1.6.0, survival 3.8-12, sva 3.60.0, sys 3.4.3, sysfonts 0.8.9, tibble 3.3.1, tidyselect 1.2.1, timechange 0.4.0, timeDate 4052.112, tinytex 0.61, tools 4.6.2, vctrs 0.7.3, withr 3.0.3, xfun 0.62, XML 3.99-0.25, xtable 1.8-8, XVector 0.52.0, yaml 2.3.12

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 Herve Pages. *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] Mattia Chiesa, Gualtiero I Colombo, and Luca Piacentini. Damirseq—an r/bioconductor package for data mining of rna-seq data: normalization, feature selection and classification. *Bioinformatics*, 34(8):1416–1418, 2018.

[13] 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.

[14] 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.

[15] Luca Piacentini, José Pablo Werba, Elisa Bono, Claudio Saccu, Elena Tremoli, Rita Spirito, and Gualtiero Ivanoe Colombo. Genome-wide expression profiling unveils autoimmune response signatures in the perivascular adipose tissue of abdominal aortic aneurysm. *Arteriosclerosis, thrombosis, and vascular biology*, 39(2):237–249, 2019.

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

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

[18] 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.

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

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

[21] 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.

[22] Daniel M Cornforth, Justine L Dees, Carolyn B Ibberson, Holly K Huse, Inger H Mathiesen, Klaus Kirketerp-Møller, Randy D Wolcott, Kendra P Rumbaugh, Thomas Bjarnsholt, and Marvin Whiteley. Pseudomonas aeruginosa transcriptome during human infection. *Proceedings of the National Academy of Sciences*, 115(22):E5125–E5134, 2018.

[23] 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.