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

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

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

February 21, 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 that lead up to the 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 covariates that occur with the experimental setting. A series of functions enable the user to clean up the data by filtering genomic features and samples, to adjust data 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 that the user may exploit to assess the effects of data management by looking at diagnostic plots, such as clustering and heatmaps, RLE boxplots, MDS or correlation plots.

Package

DaMiRseq 0.99.0

Contents

1 Introduction

RNA-Seq is a powerful high-throughput assay that uses next-generation sequencing (NGS) technologies to discover, profile and quantify RNAs. The whole collection of RNAs defines the transcriptome, whose plasticity, allows the researcher to capture important biological information since it is sensitive to any changes occurring in response to environmental challenges, to a different healthy/disease state or to a specific genetic/epigenetic context. The high-dimensional nature of the NGS technique 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 selecting, preprocessing and transforming data so that only relevant features may be 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 package can be used with any technology that produces read counts of genomic features. 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 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. However, 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 other clear advantages 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 than used to draw a “meta-learner” by combining the outputs of 6 different classifiers: Random Forest (RF), Naïve Bayes (NB), 3-Nearest Neighbours (3kNN), Logistic Regression (LR), Linear Discriminant Analysis (LDA) and Support Vectors Machines (SVM); this method belongs to the “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, 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 preparation

DaMiRseq package needs two tab-delimited files:

- **Raw counts Data** - This file has to be in the classical form of a $n \times m$ expression table 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 trasformation procedures that work on 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 Covariates Information** - This file contains the information related to classes/conditions (mandatory) and to known covariates (optional), such as demographic or clinical data, biological conditions 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 covariates). The number of rows must correspond to the number of columns in 'Raw Counts Data' table, as well as the identifiers.

In this vignette we will 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 included 544 *postmortem* donors and the mRNA sequencing of 53 tissues, 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, provided a data frame with classes and covariates included.

2.2 Import Data

DaMiRseq package uses data extracted from *SummarizedExperiment* class object. This class is usually employed to store either expression data produced by sequencing and any 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 covariate data table. The function tests if expression data are in the form of raw counts, i.e. positive numbers, if 'class' variable is included in the data frame and if "NAs" are present in either the counts and the covariates table. In this case *DaMiRseq* package needs two files as input data: 1) a raw counts table and 2) a class and (if present) covariates information table. In this vignette, we will use the dataset described in Section 2.1; we can import the count and covariate 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 the prefiltered expression data made of 21363 genes and 40 samples (20 ACC and 20 FC):

```
set.seed(12)
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 data may be retrieved by the `assay()` accessor function and covariates information may be accessed by `colData()` function [1]. The *"colData(SE)"* data frame, containing the covariates information, includes also the ***'class'*** column as it is required by the instructions (mandatory).
Notes. We have already used the `set.seed()` function that allows the user to make the results of the whole pipeline reproducible.

[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 genes and, then, perform a normalization. Furthermore, we can also decide to exclude from our dataset samples that show a low correlation among biological replicates and, thus, may be suspected to bear some technical artifact. The `DaMiR.normalization` function helps to solve the first issues, while `DaMiR.sampleFilt` allows the removal of inconsistent samples.

2.3.1 Filtering by Expression

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

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

## 2297 Features have been filtered out by espression. 19066 Features remained.
## Performing Normalization by 'vst'
```

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 `varianceStabilizingTransformation` 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 two distinct CV on sample sets that belong, respectively, to the first and the second 'class'. Genes with both 'class' CV greater than `th.cv` will be 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, for example, for 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)

## 2297 Features have been filtered out by espression. 19066 Features remained.
## 14 'Hypervariant' Features have been filtered out. 19052 Features remained.
## Performing Normalization by 'vst'

print(data_norm)

## class: SummarizedExperiment
## dim: 19052 40
## metadata(0):
## assays(1): ''
## rownames(19052): 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 8.204621 9.355828 8.764190 8.459111 9.145884 8.890036
## ENSG00000237683 7.466063 6.607132 6.452431 6.484135 6.618185 8.254552
## ENSG00000268903 5.535636 5.366677 5.078253 5.719672 5.303605 6.722529
## ENSG00000228463 7.843989 7.545048 7.676069 6.802804 6.866569 7.564442
## ENSG00000241670 5.449117 5.711407 6.107432 5.651054 5.258857 5.666543
##                     FC_2     FC_3     FC_4     FC_5
## ENSG00000227232 8.382266 8.949717 8.426013 8.803330
## ENSG00000237683 8.021946 7.079558 7.200588 7.480514
## ENSG00000268903 5.760979 5.608107 6.103205 6.017778
## ENSG00000228463 7.353488 7.725484 7.350439 7.522831
## ENSG00000241670 4.700110 4.831544 5.295067 5.775880
```

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.

2.3.4 Sample Filtering

This step introduces a sample quality checkpoint. Global gene expression should, in fact, exhibit a high correlation among biological replicates; conversely, low correlated samples may be suspected to bear some technical artifacts (e.g. poor RNA quality or library preparation), despite pass sequencing quality controls. If not assessed, these samples may, thus, negatively affect all downstream analysis. `DaMiR.sampleFilt` looks at 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] 19052    40
```

In this casa, 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 covariates 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. Nonetheless, 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, that allows the identification of the the maximum number of sv to be used for data adjustment, has been introduced in our package. Specifically, it integrates part of the SVA algorithm with custom code. Briefly, we computed eigenvalues of data and then, we calculated the squared of each eigenvalues. The ratio between each "squared eigenvalue" and 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 the two methods proposed in *sva* package.

[2] See *sva* package

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

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

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.

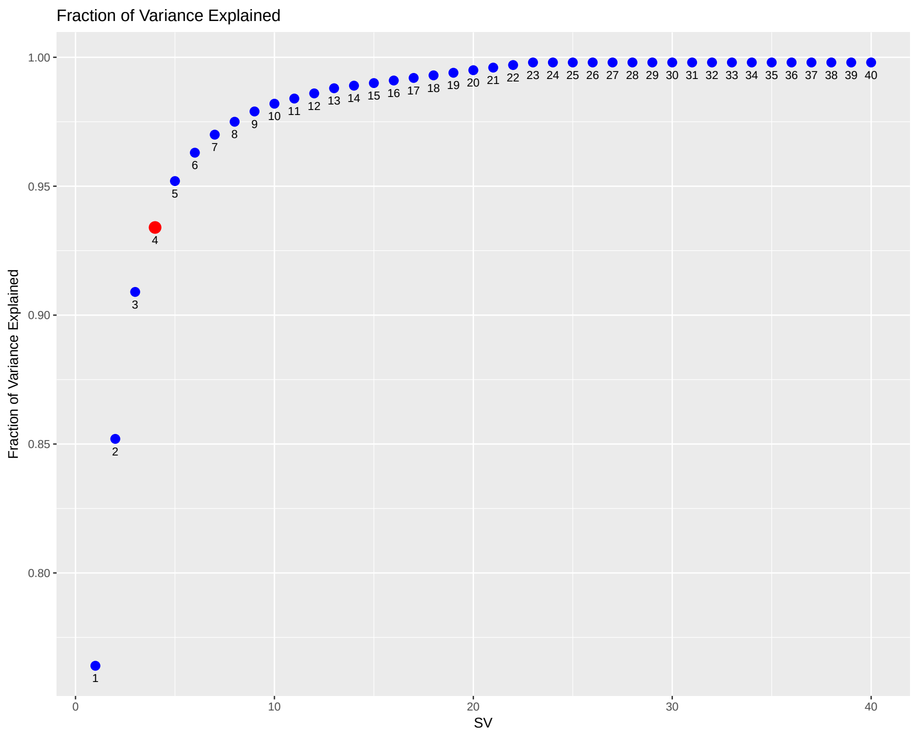


Figure 1: Fraction of Variance Explained
This plot shows the relationship between each sv identified and the corresponding fraction of variance explained. A specific blue dot represents the proportion of variance, explained by a sv together with the prior ones. The red dot marks the upper limit of sv 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, in order to decide if all of these sv (or only a subset of them) may 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 (here 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 covariates. The fourth sv presents, instead, a significant correlation with the "center" covariate. The effect of "center" might be considered a batch effect and we are interested in adjusting the data for a such confounding variable.
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 (i.e. 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.

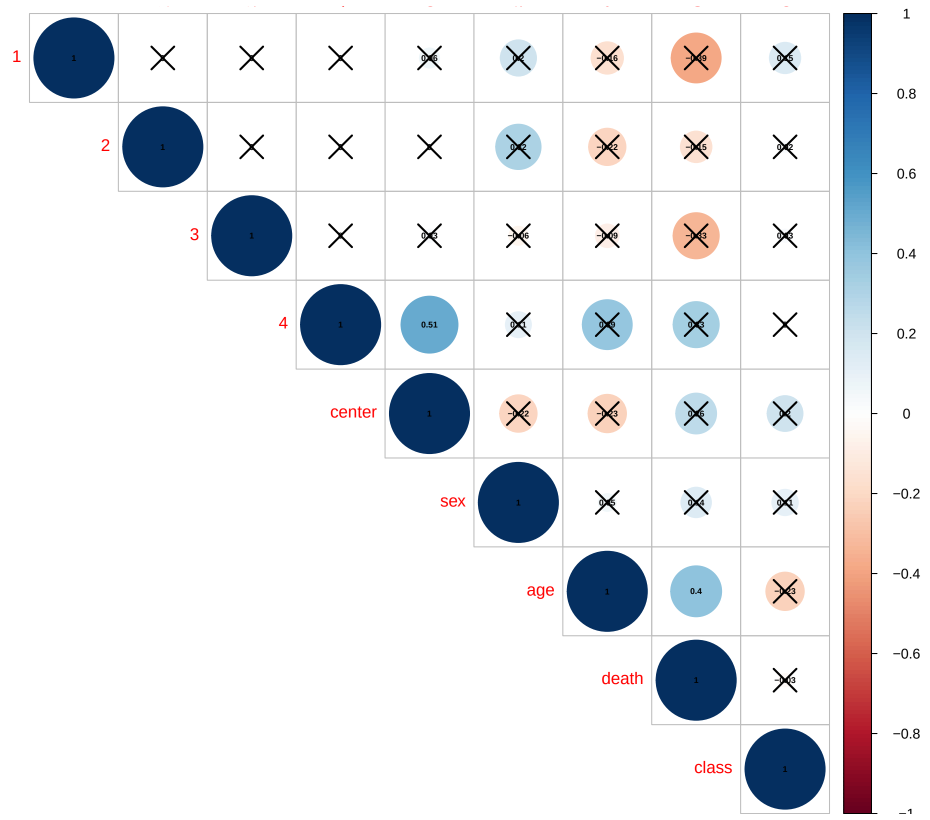


Figure 2: Correlation Plot between sv and known covariates
This plot highligths the correlation between sv and known covariates, using both color gradient and circle size. The color ranges from dark red (correlation = -1) to dark blue (correlation = 1) and the circle size is maximum for a correlation 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" covariates.

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, 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 8.297099 9.518620 8.866718 8.353466 9.021665 8.709330
## ENSG00000237683 7.408854 7.389343 6.604386 6.866093 6.561728 7.815737
## ENSG00000268903 5.679019 5.841366 5.101815 5.698543 5.349141 6.267084
## ENSG00000228463 7.770046 7.484282 7.611028 6.925974 6.912977 7.491365
## ENSG00000241670 5.591719 5.677665 6.069783 5.423661 5.304437 5.595310
##                     FC_2     FC_3     FC_4     FC_5
## ENSG00000227232 8.496465 8.942017 8.544238 9.127007
## ENSG00000237683 7.913538 7.039206 7.031835 7.096342
## ENSG00000268903 5.795415 5.606092 5.746577 5.504510
## ENSG00000228463 7.246634 7.601860 7.516979 7.719087
## ENSG00000241670 4.803654 4.887677 5.081107 5.543058
```

Now, 'data_adjust' object contains a numeric matrix of log-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 cecking the effects of each action, such as filtering, normalization and data cleaning. In this context, the function `DaMiR.Allplot` helps to identify how different arguments or specific tasks, such as filtering or normalization, could 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 covariates;

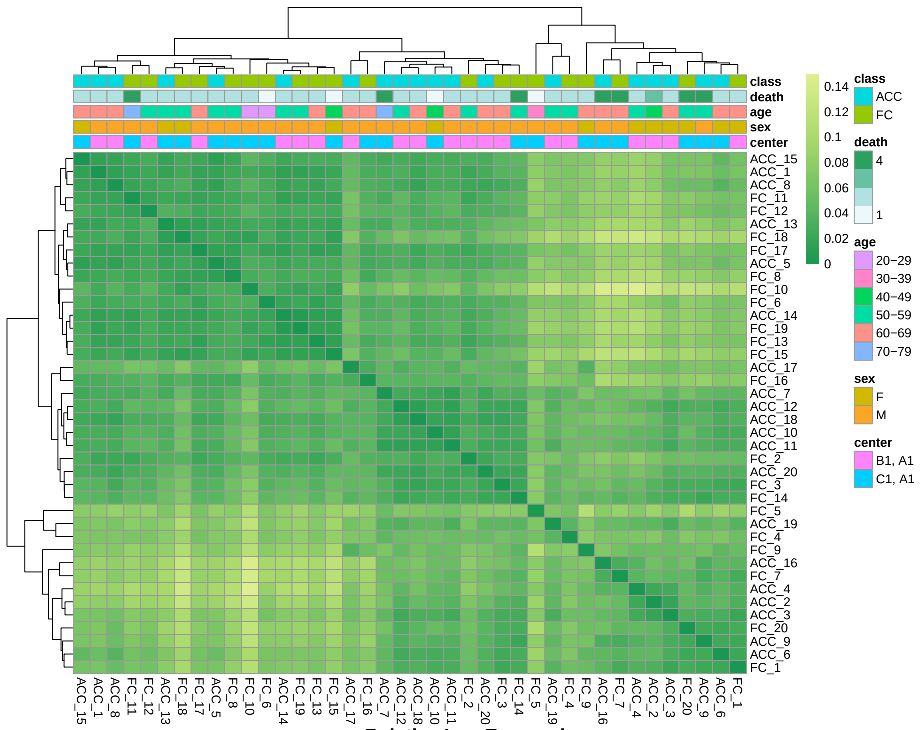
MultiDimensional Scaling (MDS) plots - MDS plot, drawn by *ggplot2* package[14], provides a visual representation of pattern of proximities (i.e. similarities or distances) among a set of samples, and allows the identification of natural clusters. For the 'class' and for each covariate a MDS plot is 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 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 covariate contains missing data (i.e. "NA" values), the function cannot draw the plot that bear covariate 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 (i.e. center, sex, age, death and class), although it is possible to use only a subset of them to be plotted.



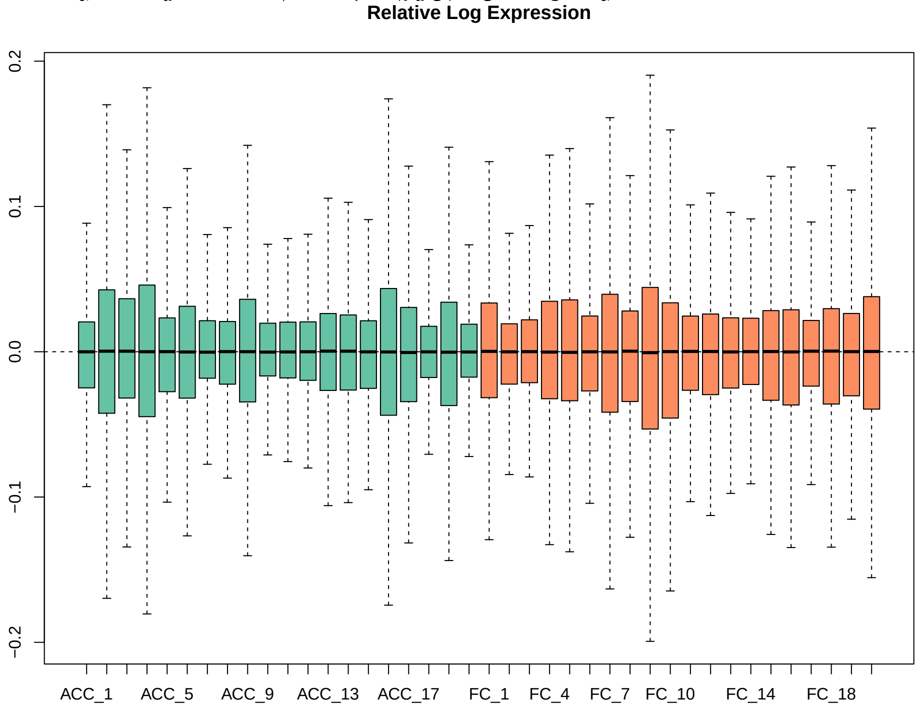


Figure 3: **Heatmap and RLE**
Heatmap (upper panel): colors in heatmap highlight the distance matrix, obtained by a correlation metric: color gradient ranges from *dark green*, meaning 'minimum distance' (i.e. correlation = 1), to *light green green*. On the top of heatmap, horizontal bars represent class and covariates. Each variable is differently colored (see legend). On the top and on the left side of the heatmap the dendrogram is 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 is shown. Here, since all medians are very close to zero, it seems that all the samples are well-normalized and do not present specific quality problems.

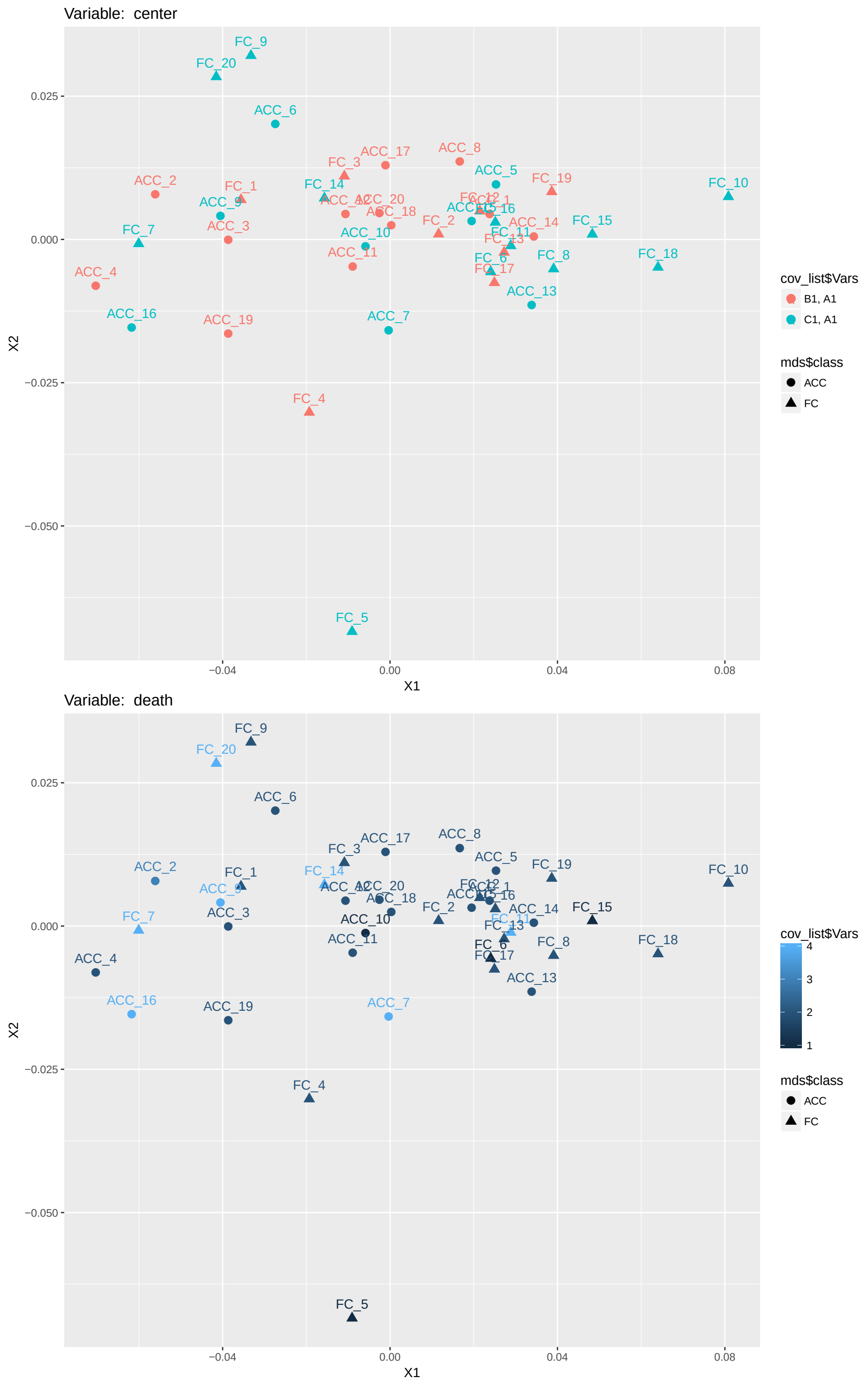


Figure 4: MultiDimentional Scaling plot
An unsupervised MDS plot is drawn. Samples are colored according to the 'Hardy death scale' (upper panel) and the 'center' covariate (lower panel).

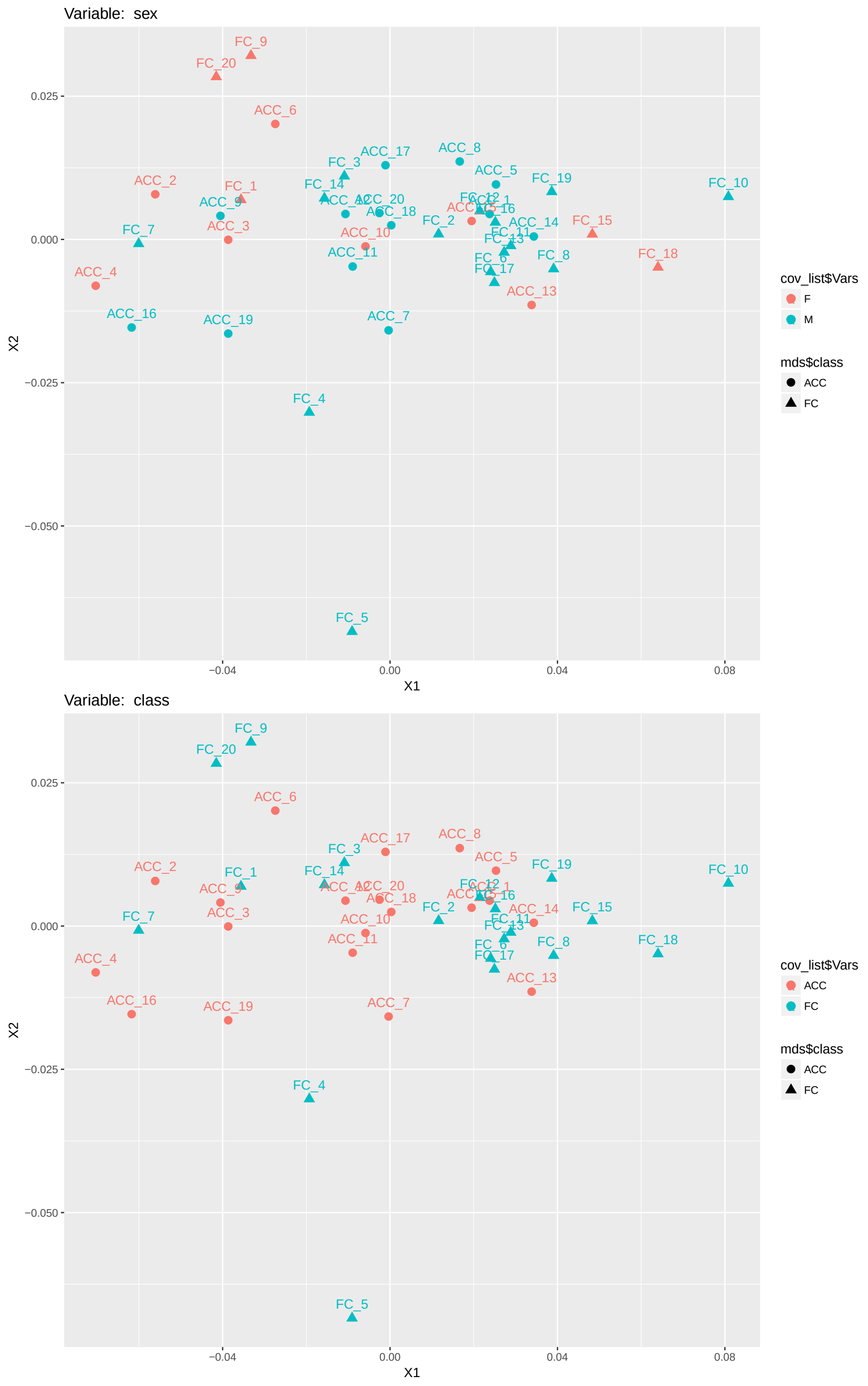
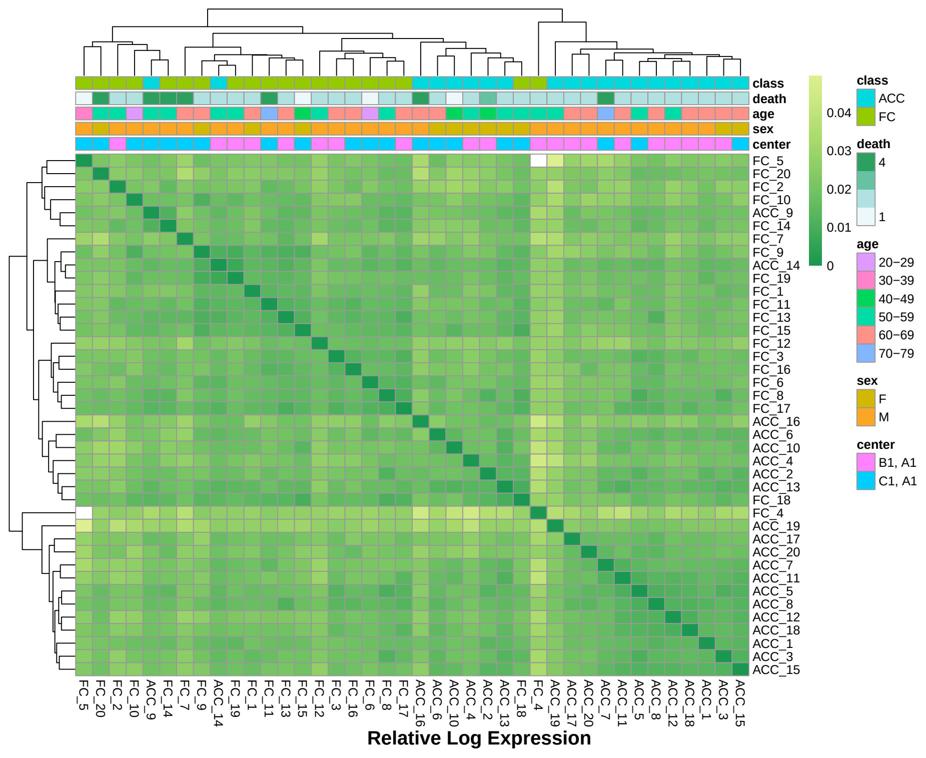


Figure 5: **MultiDimentional Scaling plot**
An unsupervised MDS plot is drawn. Samples are colored according to 'sex' covariate (upper panel) and 'class' (lower panel).

After removing noise effets from our expression data, as discussed in Section 2.4, we may appreciate the result of sv adjustiment on the data: 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.

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



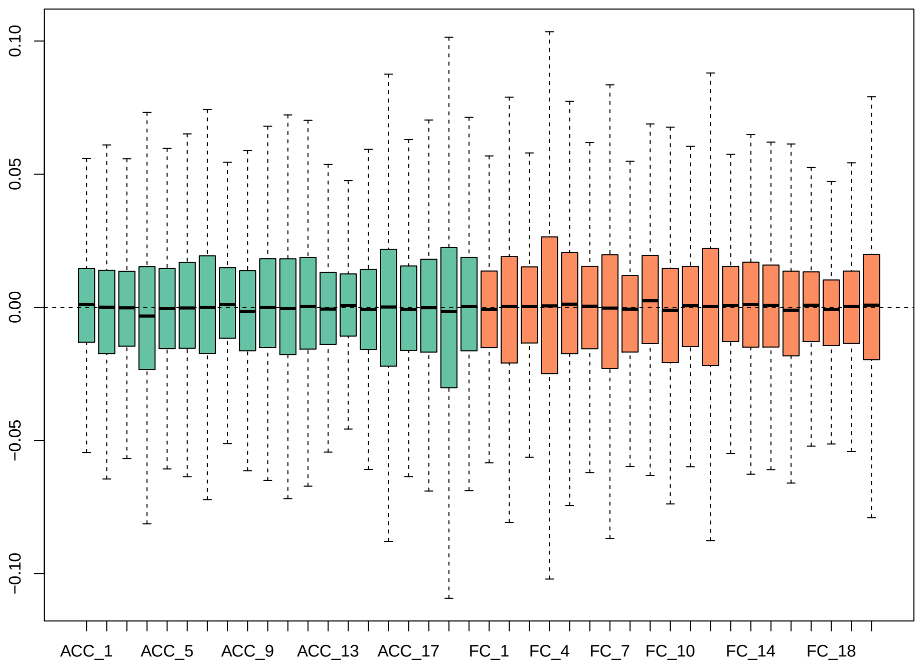


Figure 6: Heatmap and RLE
Heatmap (upper panel): colors in heatmap highlight the distance matrix, obtained by a correlation metric: color gradient ranges from *dark green*, meaning 'minimum distance' (i.e. correlation = 1), to *light green green*. On the top of heatmap, horizontal bars represent class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

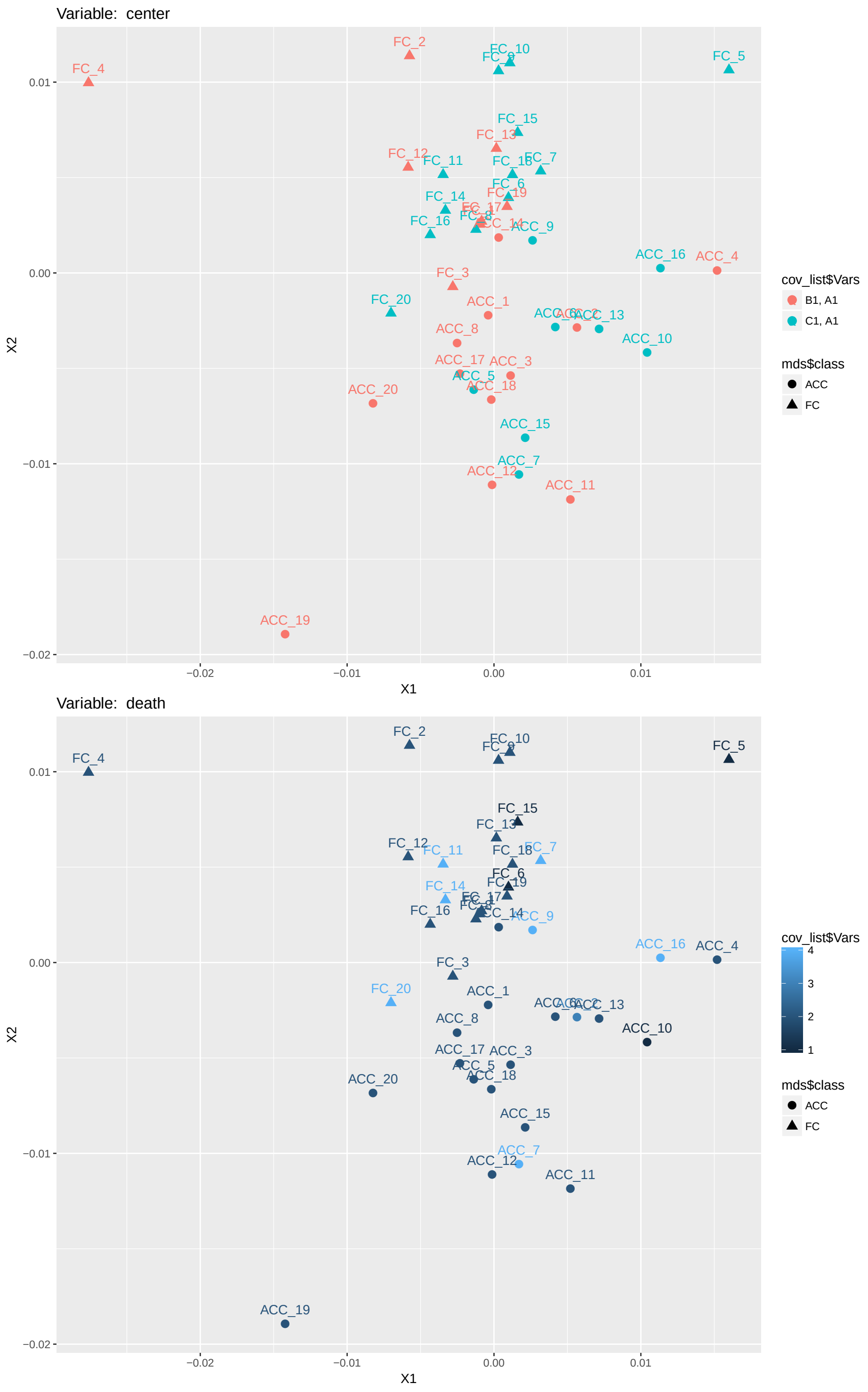


Figure 7: MultiDimentional Scaling plot
An unsupervised MDS plot is drawn. Samples are colored according to the 'Hardy death scale' (upper panel) and the 'center' covariate (lower panel).

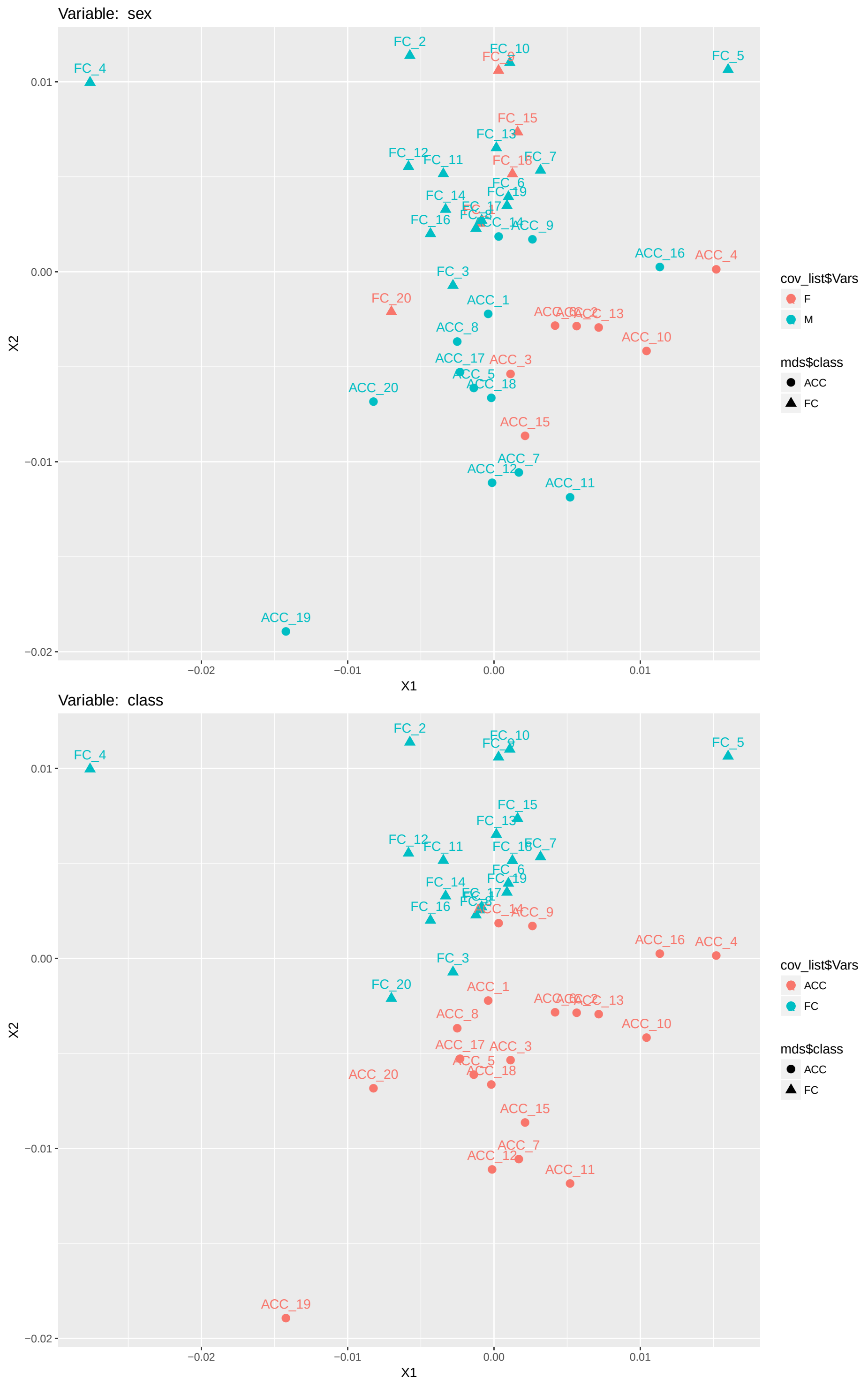


Figure 8: MultiDimentional Scaling plot
An unsupervised MDS plot is drawn. Samples are colored according to 'sex' covariate (upper panel) and 'class' (lower panel).

2.6 Feature Selection

The previous step(s) returned an 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 principle that occurs with 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) which correlate with "class". The correlation 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. Actually, the function `DaMiR.transpose` transposes 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.

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

## 18766 Genes have been discarded for classification 286 Genes remained.
```

The data_reduced object returns an expression matrix with important features. As we can see, the initial number of 19052 features has been reduced to 286.

2.6.2 Removing highly correlated features

Some of the returned important 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)
```

```
## 79 Highly correlated features have been discarded for classification.
##  207 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 further reduction step filters out 79 highly correlated genes over the 286 returned by the `DaMiR.FSelect`. The figure below shows the MDS plot drawn by the use of the expression matrix of the remaining 207 genes.

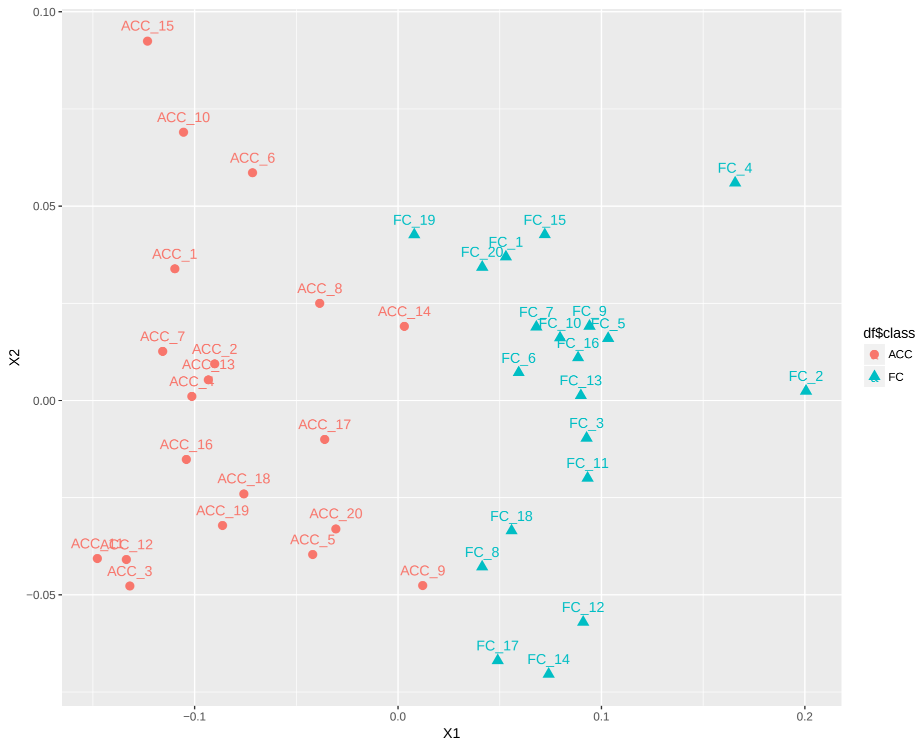


Figure 9: MultiDimentional Scaling plot
A MDS plot is drawn, considering only most informative genes, obtained after feature selection: color code is referred to 'class'.

2.6.3 Ranking and selecting important features

The above functions produced a reduced matrix of variables. The number of reduced variables might be, however, 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 in order to rank features by their importance. The method implements a multivariate filter technique (i.e. *RReliefF*) that assessess the relevance of features by looking at the properties of the data (for details see the `relief` function of the *FSelector* package) [18, 19]. The function produced a data frame with two columns, which stores features ranked by importance scores: a *RReliefF* score and *scaled.RReliefF* value; the latter is computed implementing 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 inputted. 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. Reasonably, 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 37 seconds (i.e. about 1 minutes).

  head(df.importance)

##                   RReliefF scaled.RReliefF
## ENSG00000140015 0.2859418        3.637692
## ENSG00000258754 0.2507846        3.041446
## ENSG00000198963 0.2125161        2.392434
## ENSG00000140955 0.2088250        2.329835
## ENSG00000077327 0.2018292        2.211190
## ENSG00000137699 0.1976589        2.140464
```

After the importance score is calculated, a subset of features can be selected and used as predictors for the classification. 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] "ENSG00000140015" "ENSG00000258754" "ENSG00000198963" "ENSG00000140955"
## [5] "ENSG00000077327"

  # 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.

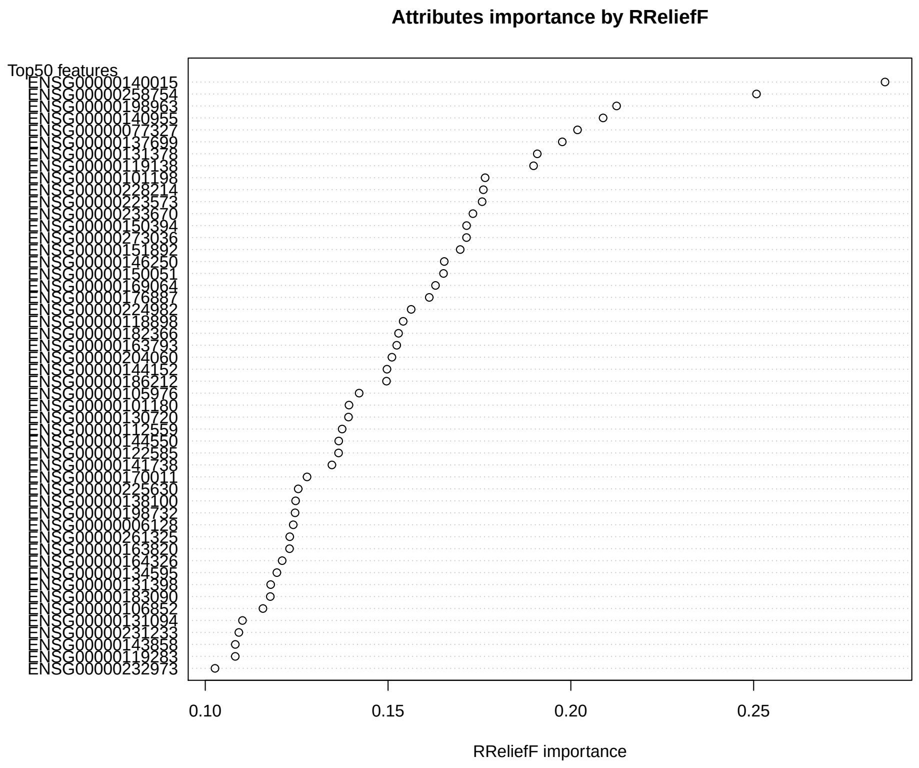


Figure 10: **Feature Importance Plot**
The dotchart shows the list of top 50 genes, sorted by RReliefF importance score. This plot may be used to select the most important predictors that are then used for classification.

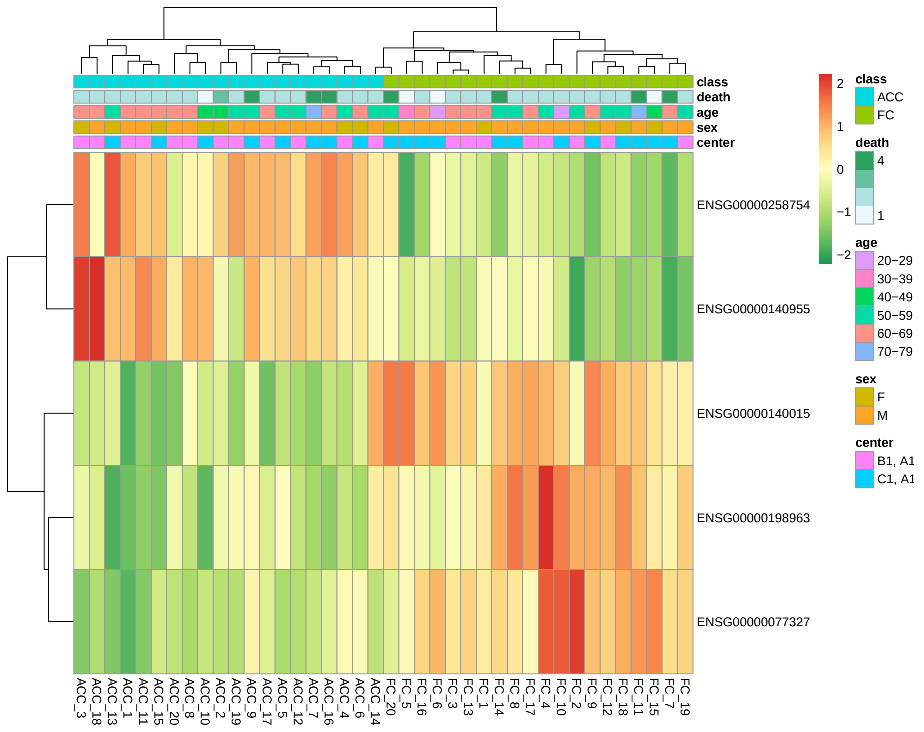


Figure 11: Clustergram
The clustergram is generated by using the expression values of the 5 predictors selected by `DaMiR.FBest` function. As for the heatmap generated by `DaMiR.Allplot`, 'class' and covariates are drawn as horizontal and color coded bars.

2.7 Classification

All the steps executed so far, allowed to normalize, clean and reduce the original expression matrix; the objective was to get 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 binary classification problems.
A meta-learner is built, combining the outputs of 6 different classifiers through a "Stacking" strategy. Currently, there is no a 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 classifier, based on its own accuracy, and then use these weights, together with predictions, to 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 and LR classifiers, whereas TS2 is used to test their accuracy and to calculate weights (w) by formula:

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

where i is a specific classifiers and N is the total number of them (here, $N = 6$). Using this approach:

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

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 binary (0 or 1) predictions (Pr) of each classifier; for each sample *k*, the prediction is computed by:

$$\begin{aligned} Pr_{(k,Ensemble)} = w_{RF} * Pr_{(k,RF)} + w_{NB} * Pr_{(k,NB)} + w_{SVM} * Pr_{(k,SVM)} + \\ + w_{3kNN} * Pr_{(k,3kNN)} + w_{LDA} * Pr_{(k,LDA)} + w_{LR} * Pr_{(k,LR)} \end{aligned} \tag{3}$$

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

The procedure, described above, is implemented in the `DaMiR.EnsembleLearning` function, where `fSample.tr`, `fSample.tr.w` and `iter` arguments allow to algorithm tuning.
In this case, before implementing the classification procedure, we created a "class" vector. To speed up the execution time of the function, we set `iter = 30` (default is 100) even though 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)
```

```
## Ensemble classification is running.  30  iterations were chosen:
## Accuracy:
##  Ensemble RF SVM NB LDA LR 3kNN
##  Mean: 97 96.17 97.33 97 94.83 91.5 95
##  St.Dev. 2.49 3.13 2.54 3.11 6.36 7.78 4.73
```

The function returns a list containing: a matrix of accuracies of each classifier in each iteration, a matrix of weights used for each classifier in each iteration and a list of all models generated in each iteration. These objects can be accessed using the $ accessor.

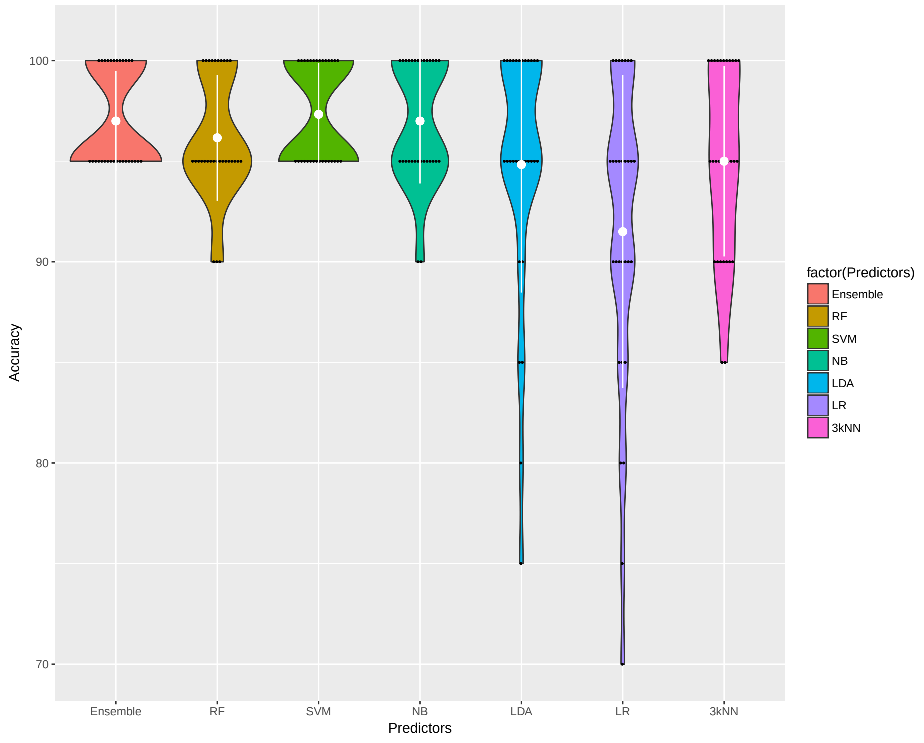


Figure 12: Accuracies Comparison
The violin plot highlights the classification accuracy of each classifier, computed at each iteration; a black dot represents a specific accuracy value while the shape of each "violin" is drawn by a Gaussian kernel density estimation. Averaged accuracies and standard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

As shown in Figure 12, almost all the weak classifiers show a very high (greater than 95% in average) classification accuracy (RF: 96.17 ± 3.13, SVM: 97.33 ± 2.54, NB: 97 ± 3.11, LDA: 94.83 ± 6.36, LR: 91.5 ± 7.78, 3kNN: 95 ± 4.73). As discussed in Section 1, the meta-learner is more influenced by better weak classifiers than inferior ones, which ensures "Ensemble" to reach a classification accuracy equal to 97 ± 2.49.

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. The 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 Standard workflow and variations

In this section, we highlight and discuss how variations to the *DaMiRseq* pipeline could affect the final classification results. Results will be discussed in Section 3.5.
Note. For simplicity, here we do not produce all plots, except for the violin plot generated by `DaMiR.EnsembleLearning`, used for performances comparison. However, the usage of `DaMiR.Allplot`, `DaMiR.corrplot`, `DaMiR.Clustplot` and `DaMiR.MDSplot` **is crucial** to check the effect of each process.

3.1 Without cleaning the data

Adjusting the data step, described in Section 2.4, could be skipped if we assume that data are not affected by any batches (known or unknown), or we do not want to take them into account. In this case, VST or rlog normalized data will be used for the following feature selection step without any other adjustment. However, we sincerely suggest to always check the presence of any confounding factors since their effects could dramatically affect the results. We set the same seed used in Section 2 (`set.seed(12)`), which ensures the right comparisons between results.

```
## 18988 Genes have been discarded for classification 64 Genes remained.
## 6 Highly correlated features have been discarded for classification.
##  58 Features remained.
## Please wait. This operation will take about 20 seconds (i.e. about 0 minutes).
##                  RReliefF scaled.RReliefF
## ENSG00000164326 0.2877387        3.607444
## ENSG00000105976 0.2496426        3.102761
## ENSG00000130720 0.1880990        2.287454
## ENSG00000101198 0.1643420        1.972730
## ENSG00000188011 0.1253070        1.455610
## ENSG00000224982 0.1176725        1.354471
## 5 Predictors have been selected for classification
## [1] "ENSG00000164326" "ENSG00000105976" "ENSG00000130720" "ENSG00000101198"
## [5] "ENSG00000188011"
## Ensemble classification is running.  30  iterations were chosen:
## Accuracy:
##  Ensemble RF SVM NB LDA LR 3kNN
##  Mean: 88.5 88.5 89.17 85.5 85.33 83.17 84.5
##  St.Dev. 5.28 4.38 6.71 6.61 8.09 8.86 9.77
```

3.2 Changing the number of predictors

The number of predictors obviously affects the classification accuracy: a small number of features could not be sufficient to ensure a good classification power; on the other hand, many predictors allow better classification accuracies if they are truly associated with the response, but at the cost of a possible reduction of the generalization ability. In this case, we decide to use 10 predictors instead of the 5 employed in the standard workflow.

```
## 10 Predictors have been selected for classification
##  [1] "ENSG00000140015" "ENSG00000258754" "ENSG00000198963" "ENSG00000140955"
##  [5] "ENSG00000077327" "ENSG00000137699" "ENSG00000131378" "ENSG00000119138"
##  [9] "ENSG00000101198" "ENSG00000228214"
## Ensemble classification is running.  30  iterations were chosen:
## Accuracy:
##  Ensemble RF SVM NB LDA LR 3kNN
##  Mean: 97 96.5 97.33 95.5 91.5 76.17 97.33
##  St.Dev. 2.82 3.26 2.86 4.02 10.43 17.05 3.41
```

3.3 Adding heterogeneous features for classification

Classification algorithms are ables to work with heterogeneous features, i.e. coming from different sources, such as other -omic data techniques, environment, demography etc. Both factorial and continuous variables may be used.
In this example, we used the data of the “sex” and “age” covariates stored in the second and third column of the “df” data frame.

```
## Ensemble classification is running.  30  iterations were chosen:
## Accuracy:
##  Ensemble RF SVM NB LDA LR 3kNN
##  Mean: 96 96.67 96.33 95 85.67 81.67 88.83
##  St.Dev. 3.32 2.4 3.7 5.25 9.98 13.22 8.17
```

3.4 Standard workflow, changing the seed

The aim is to check the robustness of “Ensemble”, changing training and test sets used before.
Note. *DaMiRseq* has also a helper function, called `DaMiR.goldenDice`, which automatically generates a number, combining date and time information; the idea is to be as random as possible for training and test subsets generation, during classification step. Here, we changed the seed (`set.seed(12345)`) before the classification procedure that will slightly affect the accuracy results.

```
## Ensemble classification is running.  30  iterations were chosen:
## Accuracy:
##  Ensemble RF SVM NB LDA LR 3kNN
##  Mean: 97.17 95.83 97.17 95.5 90.33 87.5 95.5
##  St.Dev. 2.52 3.24 2.52 4.02 7.76 8.07 3.79
```

3.5 Performances comparison

In this paragraph, we discuss the effect of each variation of the standard workflow, comparing performances of each "Ensemble" meta-learner.

Figure 13 condenses results of *DaMiRseq*'s standard workflow variations, proposed in this Section. Results will be discussed separately, in comparison with "Standard Workflow",taken as the reference, described in Section 2 and depicted in Figure 12.

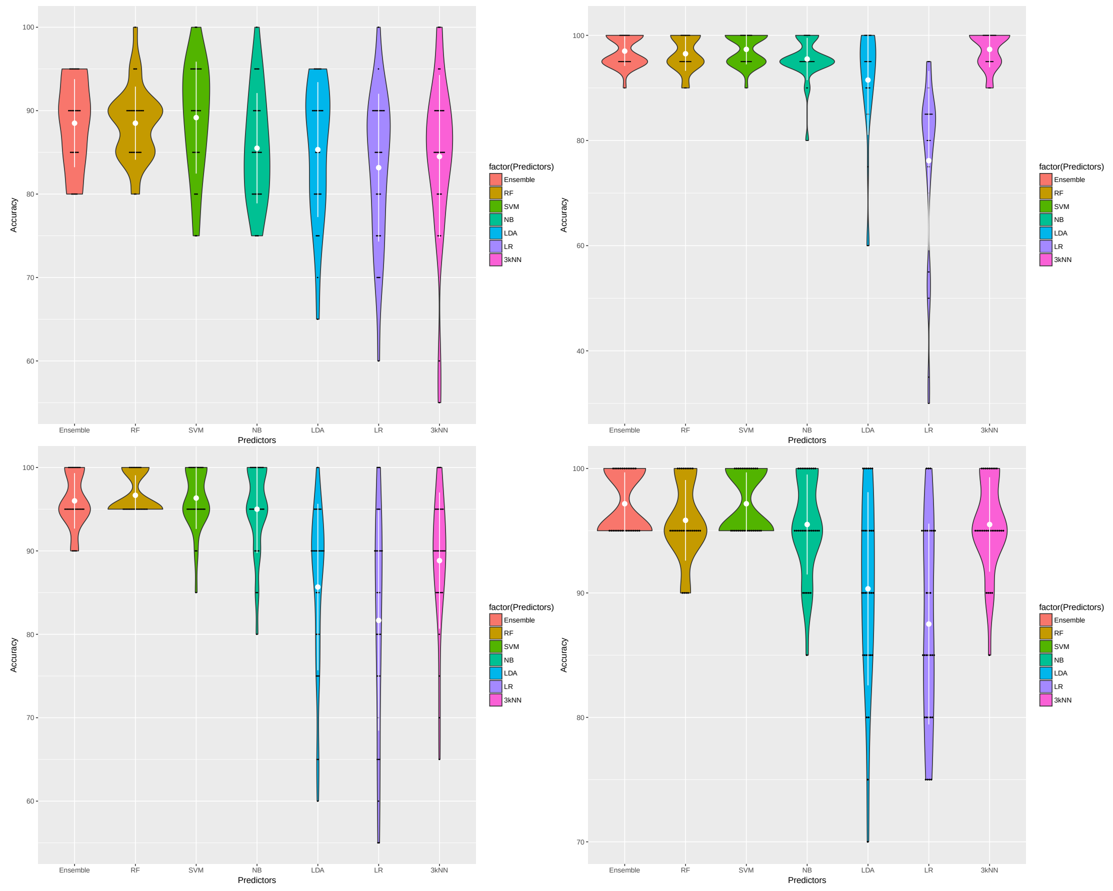


Figure 13: Performance comparison
Each violin plot shows the effect of a specific modification to *DaMiRseq* standard workflow, described in Section 2; in particular, we can see at the top left part of this figure, results of Section 3.4 workflow (*Without cleaning the data*); at the top right, results of Section 3.1's workflow (*Changing the number of predictors*); at the bottom left, results of Section 3.2 workflow (*Adding heterogeneous features for classification*); and, at the bottom right, results of Section 3.3 workflow (*Standard Workflow, changing seed*)
.

'Standard Workflow' vs 'Without cleaning the data'. 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. In this example, overall accuracies drastically decrease below 90% for all classifiers. See Figure 12 and upper right panel in Figure 13.

'Standard Workflow' vs 'Changing the number of predictors'. It is always good practice to achieve a trade-off between model complexity, generalization capability and classification power. Increasing the number of predictors changes the performances, even if in this case the classification accuracy does not increase or decrease uniformly among the classifiers. See Figure 12 and lower left panel in Figure 13.

'Standard Workflow' vs 'Adding heterogeneous features for classification'. We included the "sex" and "age" variables together with predictor genes to generate new classification models. Demographic variables may play an important role in some classification settings and may, in fact, impact prediction. However, in this case, these two variables do not impact deeply on prediction of the classifiers, with the exception of LDA and 3kNN that, conversely, showed a marked accuracy decrease. However, the poor performance of these two classifier (that are, in fact, less suitable to fit categorical variables) did not affected the overall performance of the "Ensemble" learner. See Figure 12 and bottom lower panel in Figure 13.

'Standard Workflow' vs 'Standard Workflow, changing the seed'. It is well known that, changing training and test sets, a specific classifier may present different performances. However, the stability and the robustness of a classifier affects the generalization capability. An advantage of adopting a Staking strategy is to be as stable and robust as possible. In this case the accuracy of "Ensemble" meta-learner is quite similar despite the use of different sampling, even though the performances of weak classifiers display some differences. This means that the choice of a single classifier, even when it shows the higher prediction accuracy, do not always ensure obtaining the best accuracy on new unseen data. The "Ensemble", instead, is always close to the best prediction accuracy. See Figure 12 and upper left panel in Figure 13.

4 Session Info

- R version 3.3.2 (2016-10-31), `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`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, DaMiRseq 0.99.0, GenomeInfoDb 1.10.1, GenomicRanges 1.26.1, ggplot2 2.2.0, IRanges 2.8.1, knitr 1.15.1, S4Vectors 0.12.1, SummarizedExperiment 1.4.0
- Loaded via a namespace (and not attached): acepack 1.4.1, annotate 1.52.0, AnnotationDbi 1.36.0, aroma.light 3.4.0, assertthat 0.1, backports 1.0.4, base64 2.0, bdsmatrix 1.3-2, BiocParallel 1.8.1, BiocStyle 2.3.28, biomaRt 2.30.0, Biostrings 2.42.1, bitops 1.0-6, car 2.1-4, caret 6.0-73, class 7.3-14, cluster 2.0.5, codetools 0.2-15, colorspace 1.3-2, corrplot 0.77, data.table 1.10.0, DBI 0.5-1, DESeq 1.26.0, DESeq2 1.14.1, digest 0.6.10, e1071 1.6-7, EDASeq 2.8.0, entropy 1.2.1, evaluate 0.10, FactoMineR 1.34, flashClust 1.01-2, foreach 1.4.3, foreign 0.8-67, Formula 1.2-1, FSelector 0.21, genalg 0.2.0, genefilter 1.56.0, geneplotter 1.52.0, GenomicAlignments 1.10.0, GenomicFeatures 1.26.2, grid 3.3.2, gridExtra 2.2.1, gtable 0.2.0, highr 0.6, Hmisc 4.0-1, htmlTable 1.7, htmltools 0.3.5, hwriter 1.3.2, igraph 1.0.1, iterators 1.0.8, kknn 1.3.1, labeling 0.3, lattice 0.20-34, latticeExtra 0.6-28, lazyeval 0.2.0, leaps 2.9, limma 3.30.7, lme4 1.1-12, locfit 1.5-9.1,

lubridate 1.6.0, magrittr 1.5, MASS 7.3-45, Matrix 1.2-7.1, MatrixModels 0.4-1, matrixStats 0.51.0, memoise 1.0.0, mgcv 1.8-16, minqa 1.2.4, ModelMetrics 1.1.0, munsell 0.4.3, mvtnorm 1.0-5, nlme 3.1-128, nloptr 1.0.4, nnet 7.3-12, openssl 0.9.5, pbkrtest 0.4-6, pheatmap 1.0.8, pls 2.6-0, plsVarSel 0.9.1, plyr 1.8.4, prettyunits 1.0.2, progress 1.1.2, quantreg 5.29, R.methodsS3 1.7.1, R.oo 1.21.0, R.utils 2.5.0, R6 2.2.0, randomForest 4.6-12, RColorBrewer 1.1-2, Rcpp 0.12.8, RCurl 1.95-4.8, reshape2 1.4.2, rJava 0.9-8, rmarkdown 1.2, rpart 4.1-10, rprojroot 1.1, Rsamtools 1.26.1, RSQLite 1.1-1, rtracklayer 1.34.1, RWeka 0.4-29, RWekajars 3.9.0-1, scales 0.4.1, scatterplot3d 0.3-37, ShortRead 1.32.0, SparseM 1.74, splines 3.3.2, stringi 1.1.2, stringr 1.1.0, survival 2.40-1, sva 3.22.0, tibble 1.2, tools 3.3.2, XML 3.98-1.5, xtable 1.8-2, XVector 0.14.0, yaml 2.1.14, zlibbioc 1.20.0

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.