Protein Microarray Data Analysis using the *PAA* Package

Michael Turewicz

April 24, 2017

Contents

1 Introduction

1.1 General information

Protein Array Analyzer (*PAA*, *Turewicz et al.* [1]) is a package for protein microarray data analysis (esp., *ProtoArray* data). It imports single color (protein) microarray data that has been saved in 'gpr' file format. After preprocessing (background correction, batch filtering, normalization) univariate feature preselection is performed (e.g., using the "minimum M statistic" approach - hereinafter referred to as "mMs", [2]). Subsequently, a multivariate feature selection is conducted to discover biomarker candidates. For this purpose, either a frequency-based backwards elimination approach or ensemble feature selection can be used. *PAA* provides a complete toolbox of analysis tools including several different plots for results examination and evaluation.
In this vignette the general workflow of *PAA* will be outlined by analyzing an exemplary data set that accompanies this package.

1.2 Installation

The recommended way to install *PAA* is to type the commands described below in the *R* console (note: an active internet connection is needed):

```
> # only if you install a Bioconductor package for the first time
> source("http://www.bioconductor.org/biocLite.R")
> # else
> library("BiocInstaller")
> biocLite("PAA", dependencies=TRUE)
```

This will install *PAA* including all dependencies.

Furthermore, *PAA* has an external dependency that is needed to provide full functionality. This external dependency is the free *C++* software package *"Random Jungle"* that can be downloaded from http://www.randomjungle.de/. Note: *PAA* will be usable without *Random Jungle*. However, it needs this package for random jungle recursive feature elimination (*RJ-RFE*) provided by the function `selectFeatures()`. Please follow the instructions for your OS in the README file to install *Random Jungle* properly on your machine.

2 Loading PAA and importing data

After launching *R*, the first step of the exemplary analysis is to load *PAA*.

```
> library(PAA)
```

New microarray data should be imported using the function `loadGPR()` which is mainly a wrapper to *limma*'s function `read.maimages()` featuring optional duplicate aggregation for *ProtoArray* data. *PAA* supports the import of files in 'gpr' file format. The imported data is stored in an expression list object (*EList*, respectively, *EListRaw*, see Bioconductor package *limma*). Paths to a targets file and to a folder containing 'gpr' files (all 'gpr' files in this folder that are listed in the targets file will be read) are mandatory arguments. The folder that can be obtained by the command `system.file("extdata", package = "PAA")` contains an exemplary targets file that can be used as a template. Below, the first 3 rows of this targets file are shown.

```
> targets <- read.table(file=list.files(system.file("extdata", package="PAA"),
+  pattern = "^targets", full.names = TRUE), header=TRUE)
> print(targets[1:3,])
```

```
  ArrayID                    FileName Group  Batch       Date Array SerumID
1     AD1 GSM734833_PA41992_-_AD1.gpr    AD Batch1 10.11.2010 41992     AD1
2     AD2 GSM734834_PA41994_-_AD2.gpr    AD Batch2 10.11.2010 41994     AD2
3     AD3  GSM734835_PA42006_-AD3.gpr    AD Batch1 12.11.2010 42006     AD3
```

The columns "ArrayID", "FileName", and "Group" are mandatory. "Batch" is mandatory for microarray data that has been processed in batches. The remaining three columns as well as custom columns containing further information (e.g., clinical data) are optional.

If `array.type` is set to `"ProtoArray"` duplicate spots can be aggregated. After importing, the object can be saved in a '.RData' file for further sessions. In the following code chunk, `loadGPR()` is demonstrated using an exemplary dummy data set that comes with PAA and has been created from the real data set analyzed in this vignette.

```
> gpr <- system.file("extdata", package="PAA")
> targets <- list.files(system.file("extdata", package="PAA"),
+    pattern = "dummy_targets", full.names=TRUE)
> dummy.elist <- loadGPR(gpr.path=gpr, targets.path=targets,
+    array.type="ProtoArray")
> save(dummy.elist, file=paste(gpr, "/DummyData.RData",
+    sep=""), compress="xz")
```

In many gpr files the mandatory column "Description" is not available and the needed information is stored in another column. In order to process such gpr files `loadGPR()` constructs a makeshift "Description" column. If `array.type` is set to `"ProtoArray"` this is performed automatically. Otherwise, `loadGPR()` provides the arguments `description`, `description.features` and `description.discard` to pass the information needed for the construction of the makeshift "Description" column. In the following code chunk these arguments are demonstrated.

```
> targets2 <- list.files(system.file("extdata", package="PAA"),
+  pattern = "dummy_no_descr_targets", full.names=TRUE)
> elist2 <- loadGPR(gpr.path=gpr, targets.path=targets2, array.type="other",
+  description="Name", description.features="^Hs~", description.discard="Empty")
```

PAA comes with an exemplary protein microarray data set. This 20 Alzheimer's disease serum samples vs. 20 controls data is a subset of a publicly available *ProtoArray* data set. It can be downloaded from the repository *"Gene Expression Omnibus"* (*GEO*, http://www.ncbi.nlm.nih.gov/geo/, record "GSE29676"). It has been contributed by *Nagele E et al.* [3] (note: Because a data set stored in 'gpr' files would be too large to accompany this package the exemplary data is stored as an '.RData' file).

In the following code chunk, the *PAA* installation path (where exemplary data is located) is localized, the new folder 'demo_output' (where all output of the following analysis will be saved) is created, and the exemplary data set is loaded (note: here, exceptionally not via `loadGPR()`).

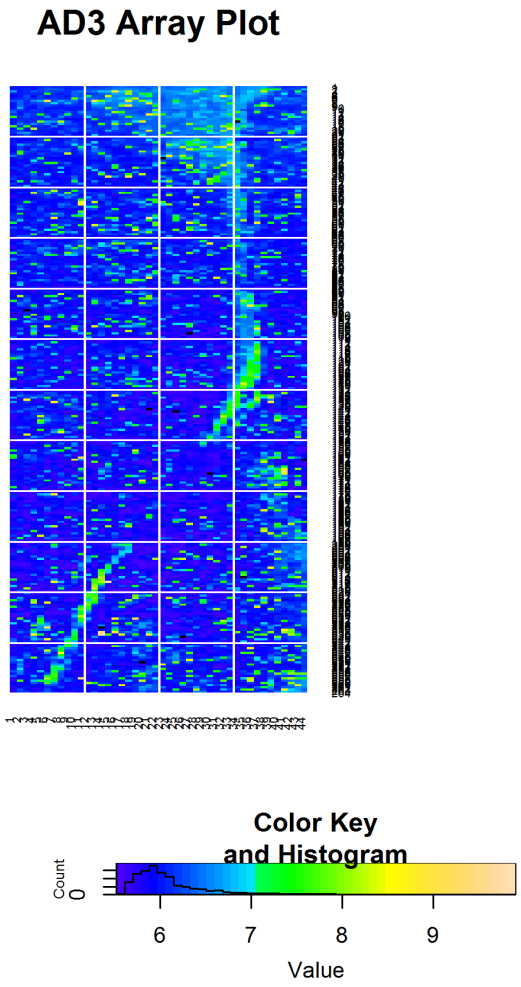
```
> cwd <- system.file(package="PAA")
> dir.create(paste(cwd, "/demo/demo_output", sep=""))
> output.path <- paste(cwd, "/demo/demo_output",  sep="")
> load(paste(cwd, "/extdata/Alzheimer.RData", sep=""))
```

3 Preprocessing

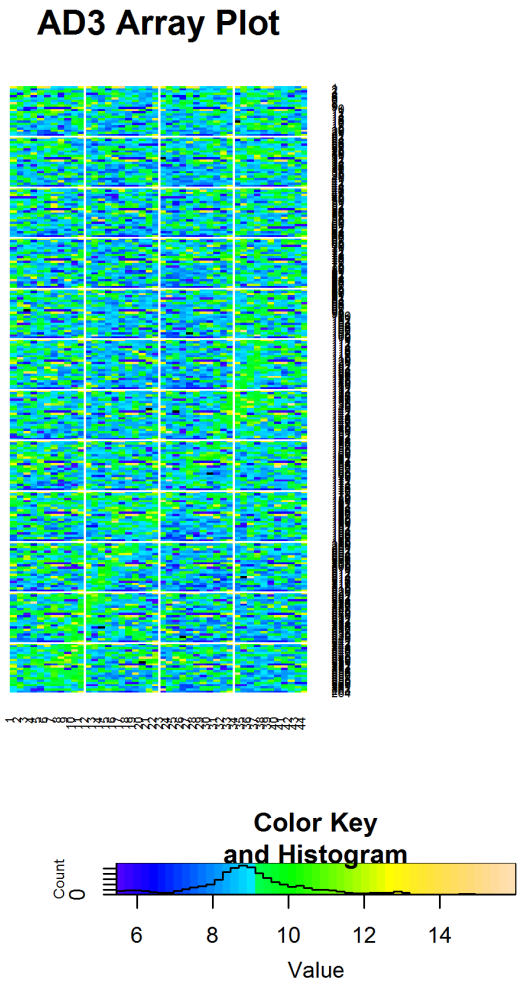
Before preprocessing the microarrays should be inspected visually for any strong spatial biases. In *PAA* this can be done using the function `plotArray()` which plots the imported microarray data in the original arrangement mimicking the original scan image. Thus, `plotArray()` is a visualization tool that can be used to visualize arrays for which the original scan image is not available. Visual inspection of the spatial expression pattern can then identify possible local tendencies and strong spatial biases. Moreover, the array can be inspected at all stages of the preprocessing workflow in order to check the impact of the particular methods that have been applied.

Consequently, as a first step, always the plots of the foreground signals (`data.type="fg"`) should be compared with the plots of the background signals (`data.type="bg"`).

```
> plotArray(elist=elist, idx=3, data.type="bg", log=FALSE, normalized=FALSE,
+   aggregation="min", colpal="topo.colors")
```




```
> plotArray(elist=elist, idx=3, data.type="fg", log=FALSE, normalized=FALSE,
+   aggregation="min", colpal="topo.colors")
```



In the exemplary plots shown above there is a relatively strong artifact in the background signal that has a slight impact on the foreground signal. In both figures an irregular stripe runs from the upper right part of the array to the bottom left corner. Although in the background plot this artifact is more visible than in the foreground plot the corresponding spatial bias is obvious.

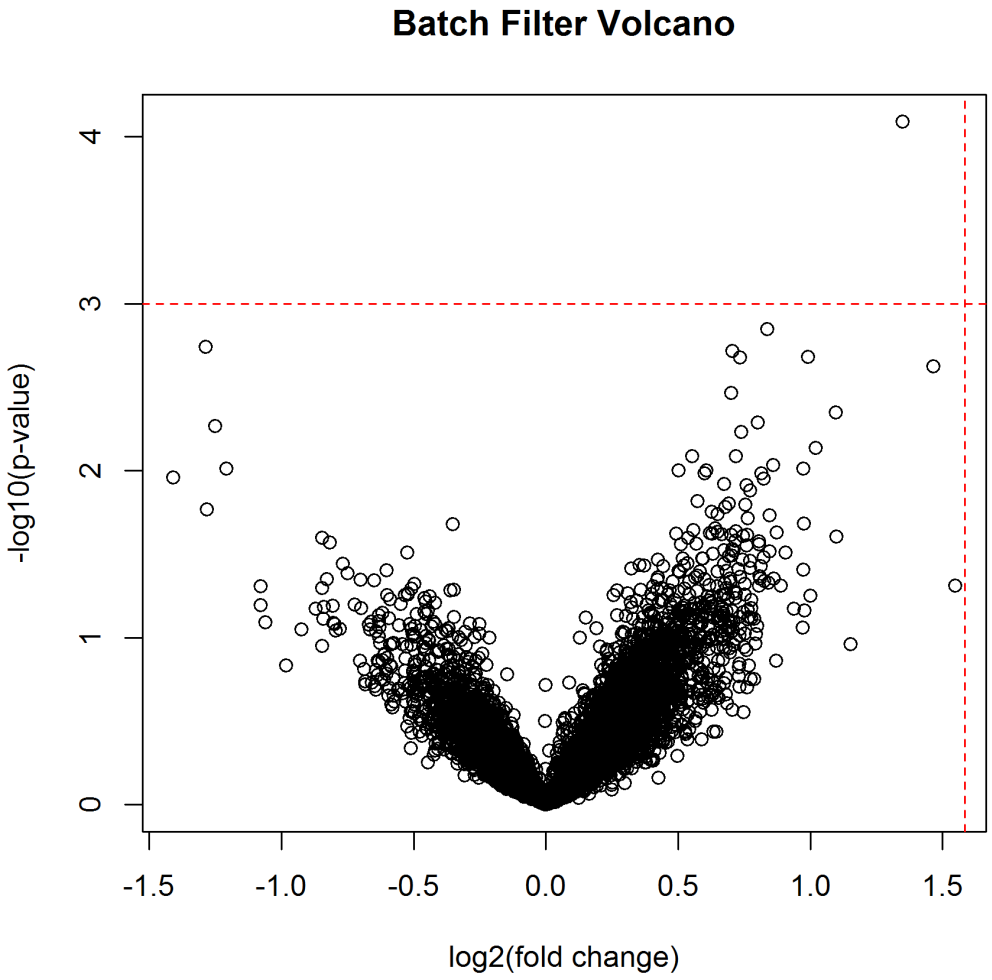
For background correction *limma*'s function `backgroundCorrect()` can be used:

```
> library(limma)
> elist <- backgroundCorrect(elist, method="normexp",
+  normexp.method="saddle")
```

If the microarrays were manufactured or processed in lots/batches, data analysis will suffer from batch effects resulting in wrong results. Hence, the elimination of batch effects is a crucial step of data preprocessing. A simple method to remove the most obvious batch effects is to find features that are extremely differential in different batches. In *PAA* this can be done for two batches using the function `batchFilter()`. This function takes an *EList* or *EListRaw* object and the batch-specific column name vectors `lot1` and `lot2` to find differential features regarding the batches/lots. For this purpose, thresholds for p-values (Student's t-test) and fold changes can be defined. To visualize the differential features a volcano plot is drawn. Finally, the differential features are removed and the remaining data is returned.

```
> lot1 <- elist$targets[elist$targets$Batch=='Batch1','ArrayID']
> lot2 <- elist$targets[elist$targets$Batch=='Batch2','ArrayID']
> elist.bF <- batchFilter(elist=elist, lot1=lot1, lot2=lot2, log=FALSE,
```

```
+  p.thresh=0.001, fold.thresh=3)
```



For multi-batch scenarios (i.e., where the number of distinct batches is larger than two) *PAA* provides the function `batchFilter.anova()`. This function calculates p-values via an one-way analysis of variance (ANOVA) in order to identify features which are differential regarding at least two batches. Apart from that this function works analogously to `batchFilter()` which has been designed for the scenario of two batches.

```
> elist.bF.a <- batchFilter.anova(elist=elist, log=FALSE, p.thresh=0.001,
+  fold.thresh=3)
> elist <- elist.bF
```

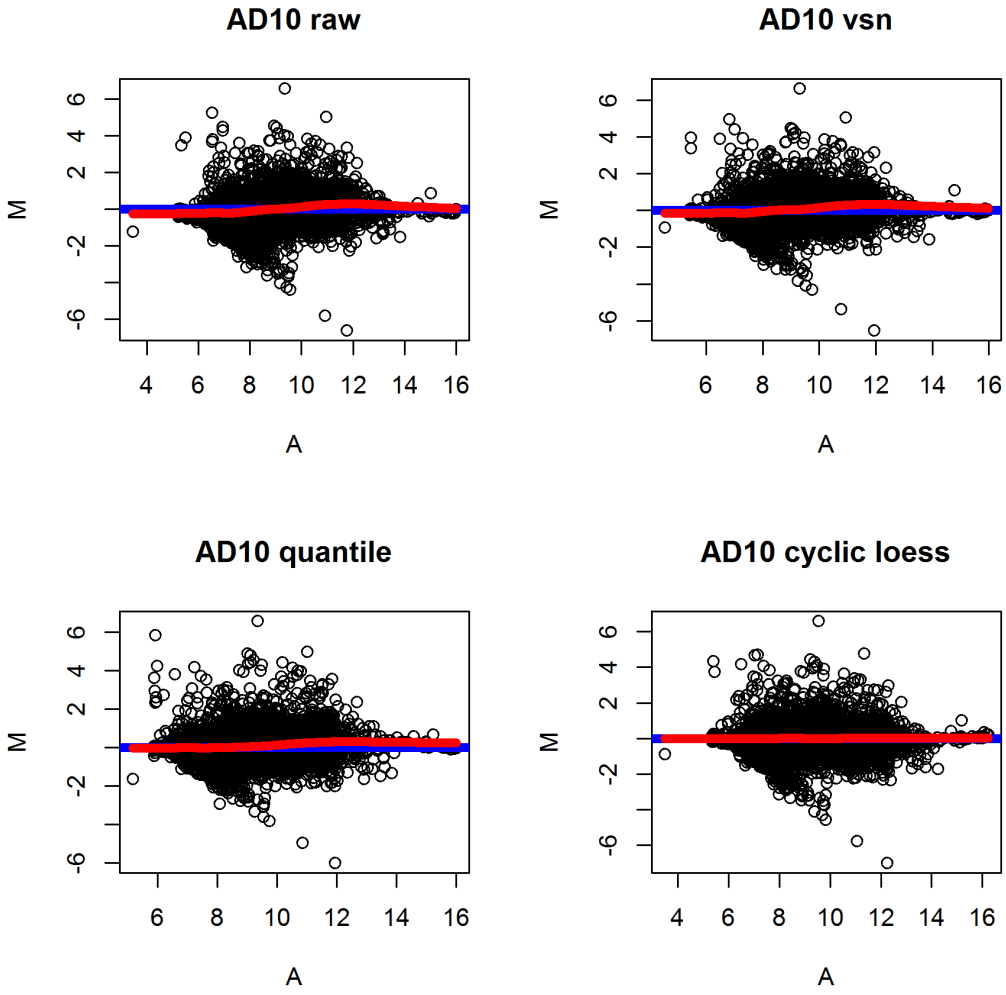
Another important step in preprocessing is normalization. To assist in choosing an appropriate normalization method for a given data set, *PAA* provides two functions: `plotNormMethods()` and `plotMAPlots()`. `plotNormMethods()` draws sample-wise boxplots of raw data and data after all kinds of normalization provided by *PAA*. For each normalization approach a plot containing all sample-wise boxplots is created. All boxplots will be saved as a high-quality 'tiff' file, if an output path is specified.

```
> plotNormMethods(elist=elist)
```

`plotMAPlots()` draws MA plots of raw data and data after applying all kinds of normalization methods provided by *PAA*. If `idx="all"` and an output path is defined (default), for each microarray one 'tiff' file containing MA plots will be created. If `idx` is an integer indicating the column index of a particular sample, MA plots only for this sample will be

created.

```
> plotMAPlots(elist=elist, idx=10)
```



After choosing a normalization method, the function `normalizeArrays()` can be used in order to normalize the data. `normalizeArrays()` takes an *EListRaw* object, normalizes the data, and returns an *EList* object containing normalized data in log2 scale. As normalization methods `"cyclicloess"`, `"quantile"` or `"vsn"` can be chosen. Furthermore, for *ProtoArrays* robust linear normalization (`"rlm"`, see *Sboner A. et al.* [4]) is provided.

```
> elist <- normalizeArrays(elist=elist, method="cyclicloess",
+ cyclicloess.method="fast")
```

In addition to `batchFilter()`, the function `batchAdjust()` can be used after normalization via `normalizeArrays()` to adjust the data for batch effects. This is a wrapper to *sva*'s function `ComBat()` for batch adjustment using the empirical Bayes approach [5]. To use `batchAdjust()` the targets file information of the *EList* object must contain the columns *"Batch"* and *"Group"*.

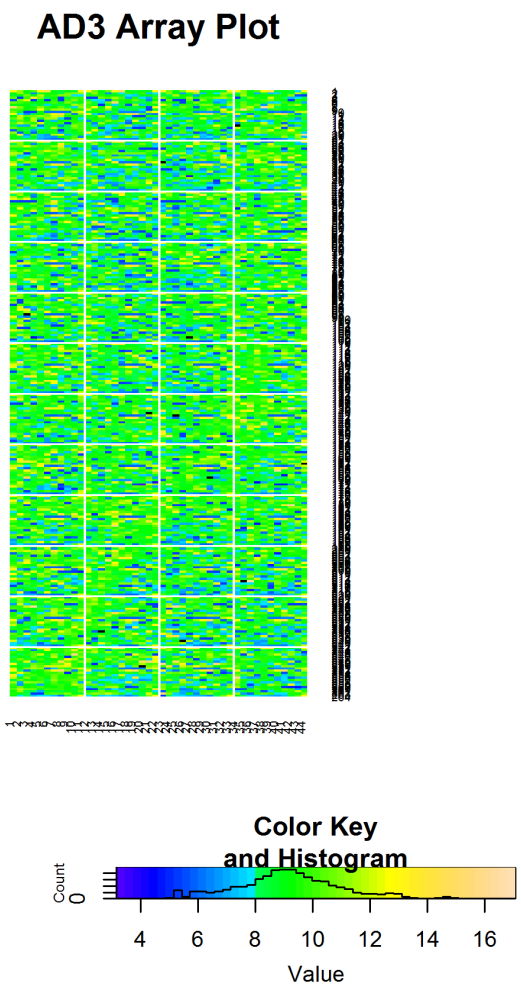
```
> elist <- batchAdjust(elist=elist, log=TRUE)
```

```
Found 2 batches
Adjusting for 1 covariate(s) or covariate level(s)
Standardizing Data across genes
Fitting L/S model and finding priors
```

```
Finding parametric adjustments
Adjusting the Data
```

After preprocessing, the corrected data can be inspected via `plotArray()` again. E.g., now the plot of the ProtoArray already plotted shows that the revealed spatial bias is less obvious after preprocessing.

```
> plotArray(elist=elist, idx=3, data.type="fg", log=TRUE, normalized=TRUE,
+   aggregation="min", colpal="topo.colors")
```



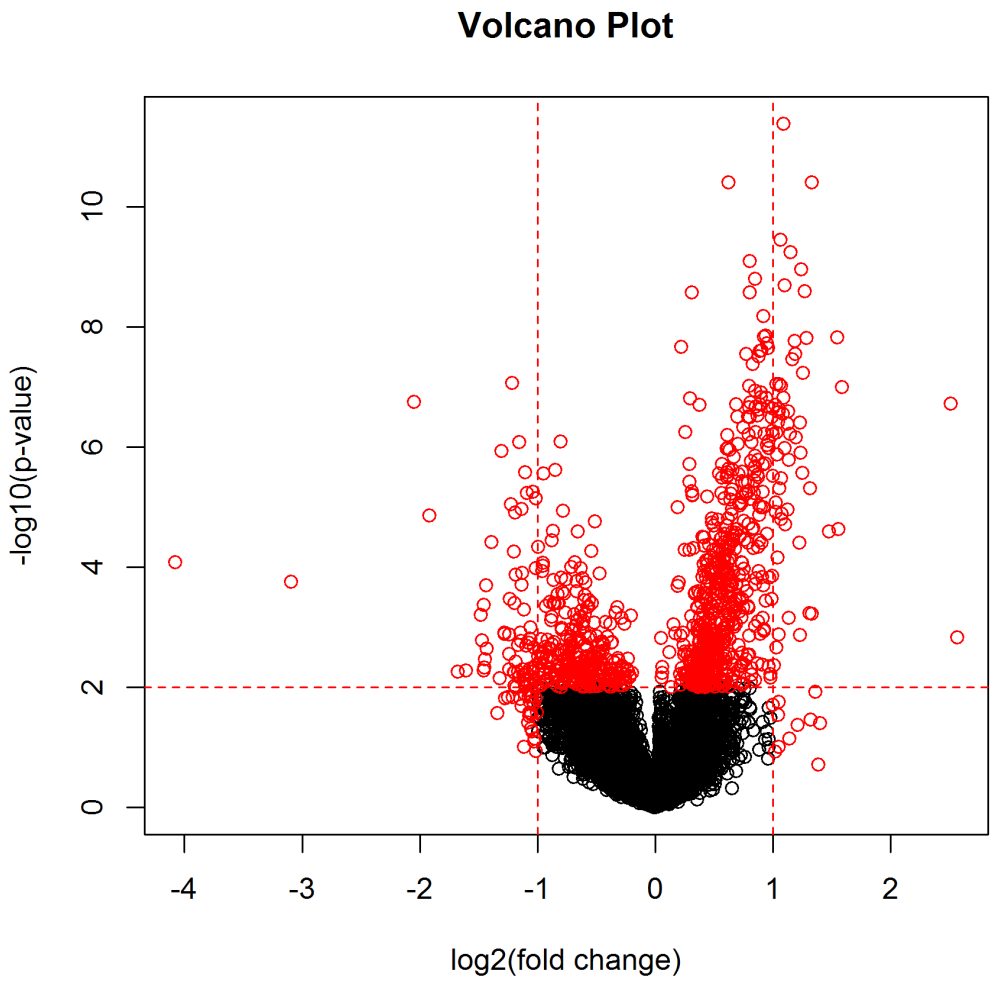
Because for further analysis also data in original scale will be needed, a copy of the *EList* object containing unlogged data should be created.

```
> elist.unlog <- elist
> elist.unlog$E <- 2^(elist$E)
```

4 Differential analysis

The goal of univariate differential analysis is to detect relevant differential features. Therefore, statistical measures such as p-values or fold changes are considered. *PAA* provides plotting functions in order to depict the number and the quality of the differential features in the data set. Accordingly, the function `volcanoPlot()` draws a volcano plot to visualize differential features. For this purpose, thresholds for p-values and fold changes can be defined. Furthermore, the p-value computation method (`"mMs"` or `"tTest"`) can be set. When an output path is defined (via `output.path`) the plot will be saved as a 'tiff' file. In the next code chunk, an example with `method="tTest"` is given.

```
> c1 <- paste(rep("AD",20), 1:20, sep="")
> c2 <- paste(rep("NDC",20), 1:20, sep="")
> #volcanoPlot(elist=elist.unlog, group1=c1, group2=c2, method="tTest",
> volcanoPlot(elist=elist, group1=c1, group2=c2, log=TRUE, method="tTest",
+ p.thresh=0.01, fold.thresh=2)
```

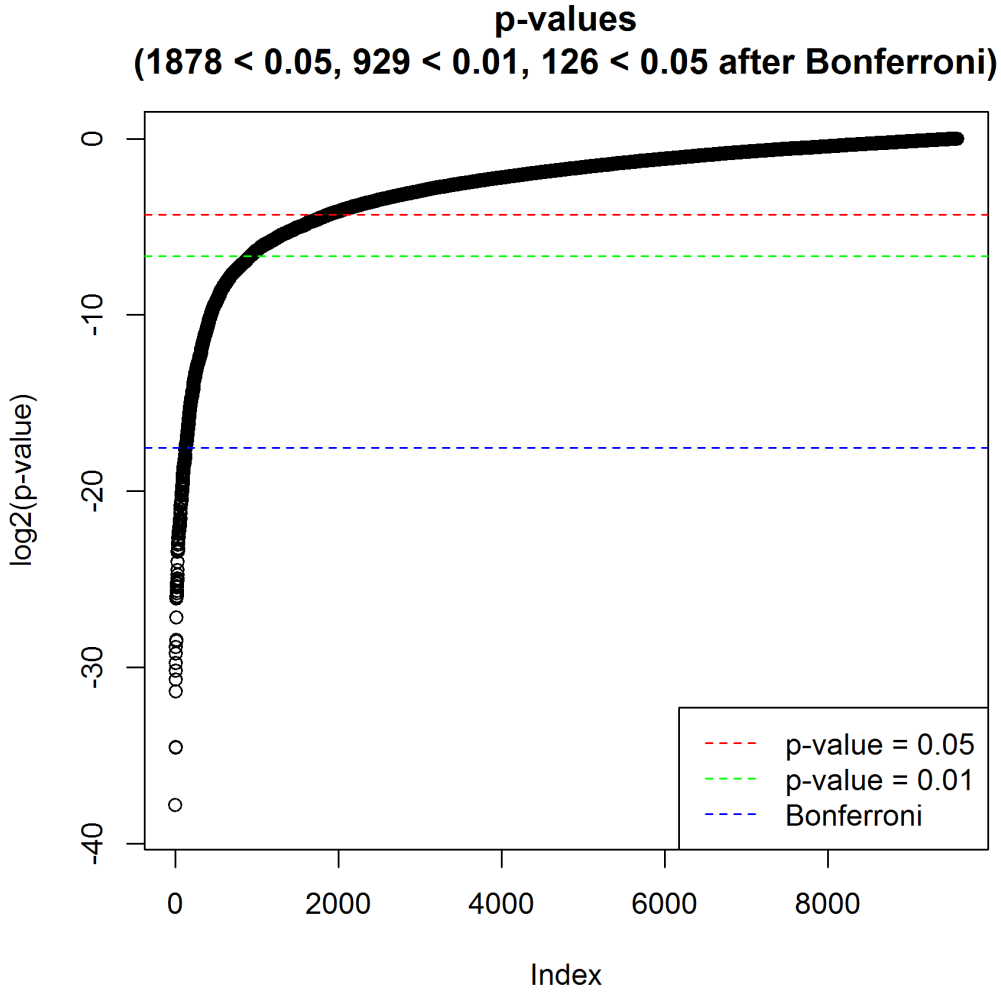


Here, an example with `method="mMs"` is given:

```
> mMs.matrix1 <- mMs.matrix2 <- mMsMatrix(x=20, y=20)
> volcanoPlot(elist=elist.unlog, group1=c1, group2=c2, log=FALSE, method="mMs",
+ p.thresh=0.01, fold.thresh=2, mMs.matrix1=mMs.matrix1,
+ mMs.matrix2=mMs.matrix2, above=1500, between=400)
```

Another plotting function is `pvaluePlot()` which draws a plot of p-values for all features in the data set (sorted in increasing order and in log2 scale). The p-value computation method (`"tTest"` or `"mMs"`) can be set via the argument `method`. Furthermore, when `adjust=TRUE` adjusted p-values (method: Benjamini & Hochberg, 1995, computed via `p.adjust()`) will be used. For a better orientation, horizontal dashed lines indicate which p-values are smaller than 0.05 and 0.01. If `adjust=FALSE`, additionally, the respective Bonferroni significance threshold (to show p-values that would be smaller than 0.05 after a possible Bonferroni correction) for the given data is indicated by a third dashed line. Note: Bonferroni is not used for the adjustment. The dashed line is for better orientation only. When an output path is defined (via `output.path`) the plot will be saved as a 'tiff' file. In the next code chunk, an example with `method="tTest"` is given.

```
> pvaluePlot(elist=elist, group1=c1, group2=c2, log=TRUE, method="tTest")
```



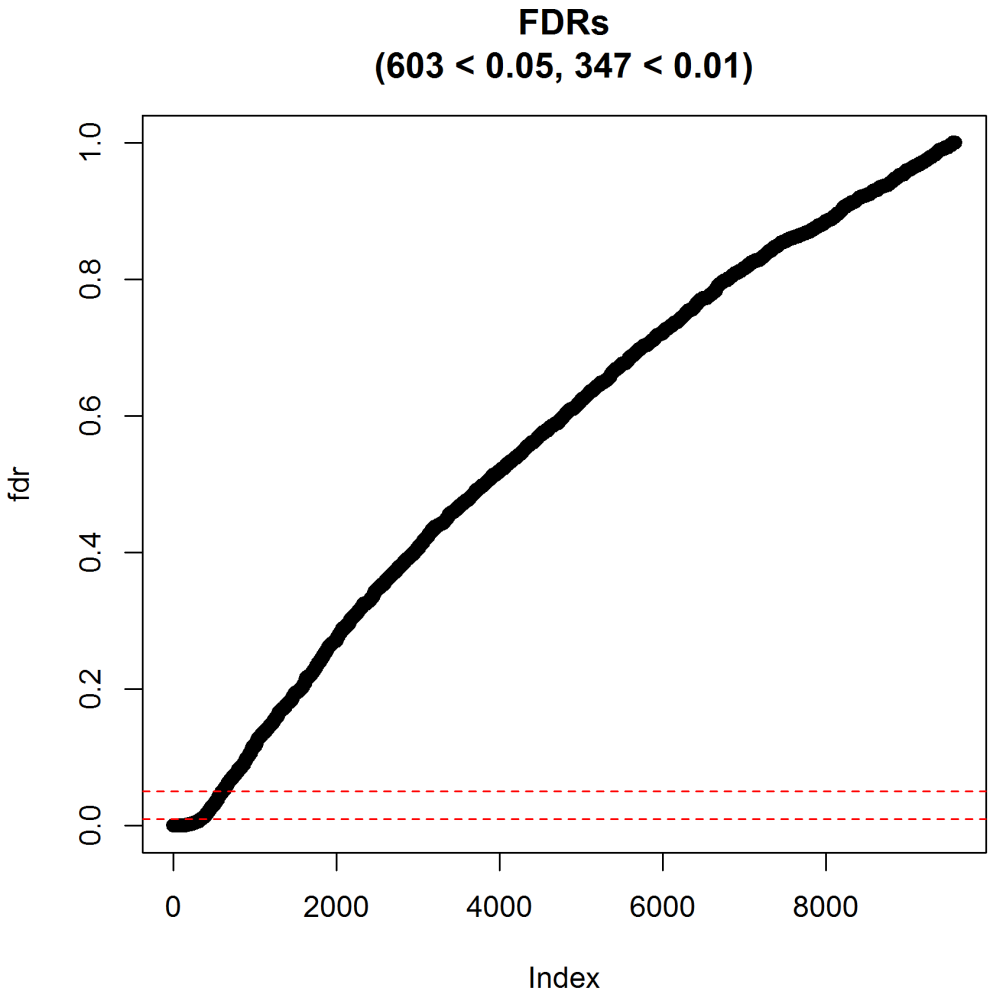
Here, an example with `method="mMs"` is given:

```
> mMs.matrix1 <- mMs.matrix2 <- mMsMatrix(x=20, y=20)
> pvaluePlot(elist=elist.unlog, group1=c1, group2=c2, log=FALSE, method="mMs",
+ mMs.matrix1=mMs.matrix1, mMs.matrix2=mMs.matrix2, above=1500,
+ between=400)
```

Here, an example with `method="tTest"` and `adjust=TRUE` is given:

```
> pvaluePlot(elist=elist, group1=c1, group2=c2, log=TRUE, method="tTest",
```

```
+   adjust=TRUE)
```



Here, an example with `method="mMs"` and `adjust=TRUE` is given:

```
> pvaluePlot(elist=elist.unlog, group1=c1, group2=c2, log=FALSE, method="mMs",
+ mMs.matrix1=mMs.matrix1, mMs.matrix2=mMs.matrix2, above=1500,
+ between=400, adjust=TRUE)
```

Finally, `diffAnalysis()` performs a detailed univariate differential analysis. This function takes an `EList$E`- or `EListRaw$E`- matrix (e.g., `temp <- elist$E`) extended by row names comprising *"BRC"*-IDs of the corresponding features. The BRC-IDs can be created via:
`brc <- paste(elist$genes[,1], elist$genes[,3], elist$genes[,2])`.
Next, the row names can be assigned as follows: `rownames(temp) <- brc`. Furthermore, the corresponding column name vectors, group labels and mMs- parameters are needed to perform the univariate differential analysis. This analysis covers inter alia p-value computation, p-value adjustment (method: Benjamini & Hochberg, 1995), and fold change computation. Since the results table is usually large, a path for saving the results should be defined via `output.path`. Optionally, a vector of row indices (`features`) and additionally (not mandatory for subset analysis) a vector of corresponding feature names (`feature.names`) can be forwarded to perform the analysis for a feature subset.

```
> E <- elist.unlog$E
> rownames(E) <- paste(elist.unlog$genes[,1], elist.unlog$genes[,3],
+     elist.unlog$genes[,2])
```

```
> write.table(x=cbind(rownames(E),E),
+     file=paste(cwd,"/demo/demo_output/data.txt", sep=""), sep="\t", eol="\n",
+     row.names=FALSE, quote=FALSE)
> mMs.matrix1 <- mMs.matrix2 <- mMsMatrix(x=20, y=20)
> diff.analysis.results <- diffAnalysis(input=E, label1=c1, label2=c2,
+     class1="AD", class2="NDC", output.path=output.path,
+     mMs.matrix1=mMs.matrix1, mMs.matrix2=mMs.matrix2, above=1500,
+     between=400)
> print(diff.analysis.results[1:10,])
```

```
      BRC              t.test            FDR.t. min..M.stat...mMs.          FDR.mMs.
1  1 2 11   0.352751286646465  0.65394032025644  0.243589743589744 0.835015538626552
2  1 2 13    0.15129653649193 0.501988087542919 0.0241860325286354 0.330335726331277
3  1 2 15    0.32087558901485 0.634210502468436                  1                1
4  1 2 17   0.178493270666023  0.52723150928707  0.150422391245528 0.835015538626552
5  1 2 19   0.271589613948769 0.597932135616531  0.243589743589744 0.835015538626552
6  1 2 21  0.0707666042203724  0.39497036996504 0.0457380457380457 0.484098604098604
7   1 3 1  0.0284911744794211 0.268607060873834                  1                1
8   1 3 3 0.00921968894879159 0.142280968989526                0.5 0.910368401063426
9   1 3 5 0.00592168460061226 0.106226964810385  0.053014553014553 0.484098604098604
10  1 3 7   0.806173147194642 0.916805614435088  0.302494802494803 0.910368401063426
         fold.change          mean.AD         mean.NDC        median.AD       median.NDC
1   1.36257910953312 1384.85432520349 1016.34783295481 839.980702981773 857.735001801978
2  0.260784177611613 2189.55822304562 8396.05471121233 1305.46661974481 2548.69868040327
3   1.10224097047813 450.550786587285 408.758881818594 413.716205403229 417.428604869083
4  0.595491172830503 1518.91515397688 2550.69297964089 1215.20762270891  1689.1444916955
5  0.454051309774646  2530.1344545838 5572.35360875746 1824.40003073848 1864.76465290004
6  0.759043204421859 2636.25422481143 3473.12802414112 2248.99036532219 2924.27002197574
7   1.26294406395723 484.784697258339 383.852864979108 446.558759285452 349.263285007632
8   1.48059072464481 692.098536568906 467.447570114244 556.156389770551 455.411881000121
9    1.3602309262767 1991.44021666422   1464.045683857 1873.36646648453 1436.18751389589
10 0.908397966805546 818.482441505167 901.017474074085 730.232223314622 469.463840773123
              sd.AD           sd.NDC
1  1643.99677640782   564.4302706304
2  2970.84080119717 18364.0058765935
3  165.734824607582 82.0150145643134
4  1062.44868297829 3151.76500089472
5  2444.44412207378  11793.108729466
6  1276.31835442119 1553.96096388882
7  154.993619177929 122.909766431917
8  339.123207021637 93.5019966245782
9  718.441427077343 323.488045744025
10 433.002023903253 1422.22599366313
```

Subsequently, the most relevant differential features (i.e., features having low p-values and high absolute fold changes) can be extracted as a univariate feature selection. Nevertheless, it is recommended to perform also multivariate feature selection and to consider feature panels obtained from both approaches.

5 Feature preselection

Before multivariate feature selection will be performed, it is recommended to discard features that are obviously not differential. Discarding them will accelerate runtimes without any negative impact on results. In *PAA*, this task is called *"feature preselection"* and it is performed by the function `preselect()`. This function iterates all features of the data set to score them via *mMs*, *Student's t-test*, or *mRMR*. If `discard.features` is `TRUE` (default), all features that are considered as obviously not differential will be collected and returned for discarding. Which features are considered as not differential depends on the parameters `method`, `discard.threshold`, and `fold.thresh`.

- If `method = "mMs"` features having an *mMs* value larger than `discard.threshold` (here: numeric between 0.0 and 1.0) or do not satisfy the minimal absolute fold change `fold.thresh` will be considered as not differential.
- If `method = "tTest"` features having a p-value larger than `discard.threshold` (here: numeric between 0.0 and 1.0) or do not satisfy the minimal absolute fold change `fold.thresh` will be considered as not differential.
- If `method = "mrmr"` *mRMR* scores for all features will be computed as scoring method (using the function `mRMR.classic()` of the *R* package *mRMRe*). Subsequently, features that are not the `discard.threshold` (here: integer indicating a number of features) features having the best *mRMR* scores are considered as not differential.

```
> mMs.matrix1 <- mMs.matrix2 <- mMsMatrix(x=20, y=20)
> pre.sel.results <- preselect(elist=elist.unlog, columns1=c1, columns2=c2,
+     label1="AD", label2="NDC", log=FALSE, discard.threshold=0.5,
+     fold.thresh=1.5, discard.features=TRUE, mMs.above=1500, mMs.between=400,
+     mMs.matrix1=mMs.matrix1, mMs.matrix2=mMs.matrix2,
+     method="mMs")
> elist <- elist[-pre.sel.results$discard,]
```

6 Feature selection

For multivariate feature selection *PAA* provides the function `selectFeatures()`. It performs a multivariate feature selection using *"frequency-based"* feature selection (based on *RF-RFE*, *RJ-RFE* or *SVM-RFE*) or *"ensemble"* feature selection (based on *SVM-RFE*).

Frequency-based feature selection (`method="frequency"`): The whole data is splitted in k cross validation training and test set pairs. For each training set a multivariate feature selection procedure is performed. The resulting k feature subsets are tested using the corresponding test sets (via classification). As a result, `selectFeatures()` returns the average k-fold cross validation classification accuracy as well as the selected feature panel (i.e., the union set of the k particular feature subsets). As multivariate feature selection methods random forest recursive feature elimination (*RF-RFE*), random jungle recursive feature elimination (*RJ-RFE*) and support vector machine recursive feature elimination (*SVM-RFE*) are supported. To reduce running times, optionally, an additional univariate feature preselection can be performed (control via `preselection.method`). As univariate preselection methods mMs (`"mMs"`), Student's t-test (`"tTest"`) and mRMR (`"mrmr"`) are supported. Alternatively, no preselection can be chosen (`"none"`). This approach is similar to the method proposed in *Baek et al.* [6].

Ensemble feature selection (`method="ensemble"`): From the whole data a previously defined number of subsamples is drawn defining pairs of training and test sets. Moreover, for each training set a previously defined number of bootstrap samples is drawn. Then, for each bootstrap sample *SVM-RFE* is performed and a feature ranking is obtained. To obtain a final ranking for a particular training set, all associated bootstrap rankings are aggregated to a single ranking. To score the `cutoff` best features, for each subsample a classification of the test set is performed (using a svm trained with the `cutoff` best features from the training set) and the classification accuracy is determined. Finally, the stability of the subsample-specific panels is assessed (via Kuncheva index, *Kuncheva LI, 2007* [7]), all subsample-specific rankings are aggregated, the top n features (defined by `cutoff`) are selected, the average classiification accuracy is computed, and all these results are returned in a list. This approach has been proposed in *Abeel et al.* [8].

`selectFeatures()` takes an `EListRaw` or `EList` object, group-specific sample numbers, group labels and parameters choosing and configuring a multivariate feature selection method (frequency-based or ensemble feature selection) to select a panel of differential features. When an output path is defined (via `output.path`) results will be saved on the hard disk and when `verbose` is TRUE additional information will be printed to the console. Depending on the selection method, one of two different results lists will be returned:

1. If `method` is `"frequency"`, the results list contains the following elements:
   - accuracy: average k-fold cross validation accuracy.
   - sensitivity: average k-fold cross validation sensitivity.
   - specificity: average k-fold cross validation specificity.
   - features: selected feature panel.
   - all.results: complete cross validation results.
2. If `method` is `"ensemble"`, the results list contains the following elements:
   - accuracy: average accuracy regarding all subsamples.
   - sensitivity: average sensitivity regarding all subsamples.
   - specificity: average specificity regarding all subsamples.
   - features: selected feature panel.
   - all.results: all feature ranking results.
   - stability: stability of the feature panel (i.e., Kuncheva index for the subrun-specific panels).

In the following two code chunks first *"frequency-based"* feature selection and then *"ensemble"* feature selection is demonstrated.

```
> selectFeatures.results <- selectFeatures(elist,n1=20,n2=20,label1="AD",
+     label2="NDC",log=TRUE,selection.method="rf.rfe",subruns=2,
+     candidate.number=1000,method="frequency")
```

```
> selectFeatures.results <- selectFeatures(elist,n1=20,n2=20,label1="AD",
+     label2="NDC",log=TRUE,subsamples=10,bootstraps=10,method="ensemble")
```

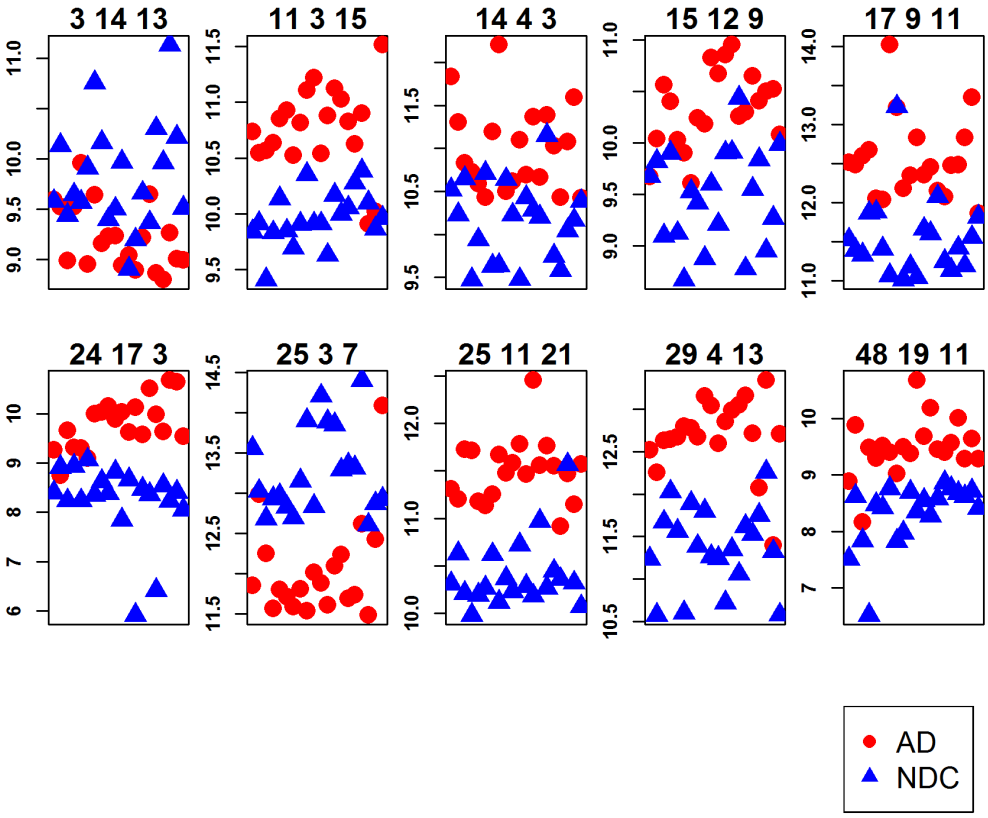
Because runtimes would take too long for this vignette *PAA* comes with precomputed `selectFeatures.results` objects stored in '.RData' files. These objects can be loaded as follows:

```
> # results of frequency-based feature selection:
> load(paste(cwd, "/extdata/selectFeaturesResultsFreq.RData", sep=""))
> # or results of ensemble feature selection:
> load(paste(cwd, "/extdata/selectFeaturesResultsEns.RData", sep=""))
```

7 Results inspection

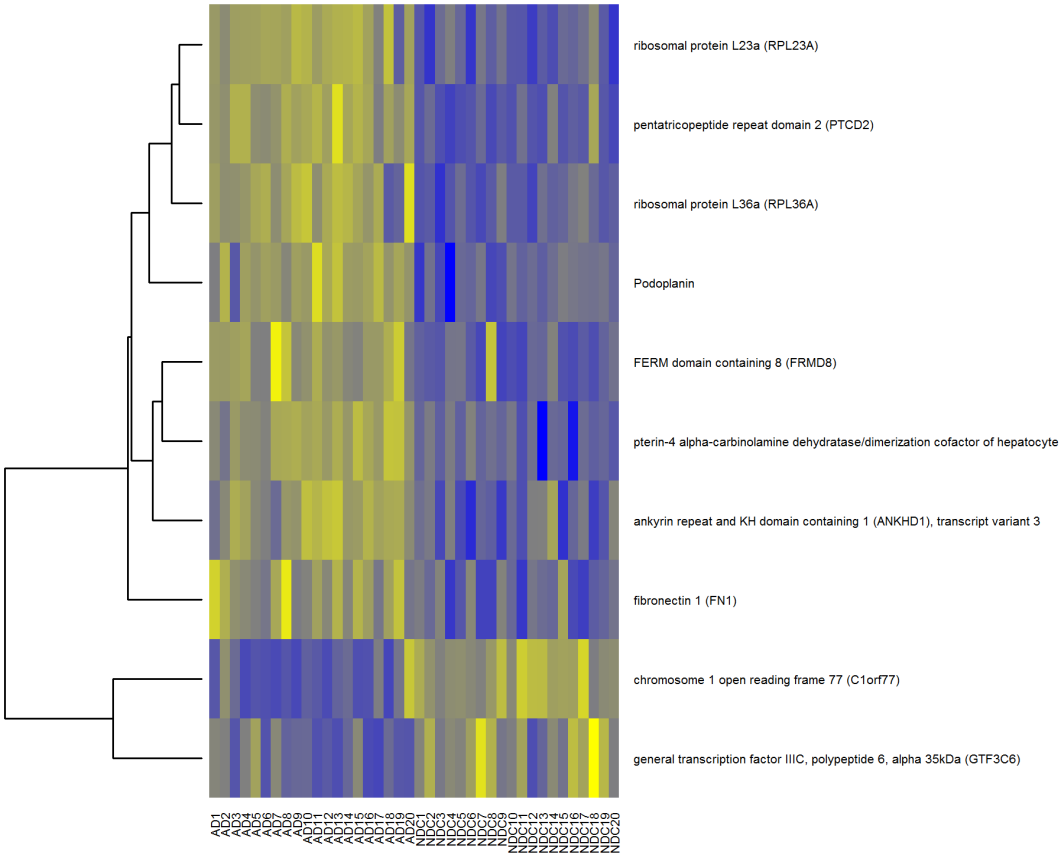
After the selection of a feature panel, these features should be validated by manual inspection and evaluation for further research. To aid results inspection, *PAA* provides several functions. The function `plotFeatures()` plots the intensities of all features (represented by BRC-IDs) that have been selected by `selectFeatures()` (one sub-plot per feature) in group-specific colors. All sub-plots are aggregated in one figure. If `output.path` is not NULL, this figure will be saved as a 'tiff' file in `output.path`.

```
> plotFeatures(features=selectFeatures.results$features, elist=elist, n1=20,
+     n2=20, group1="AD", group2="NDC")
```



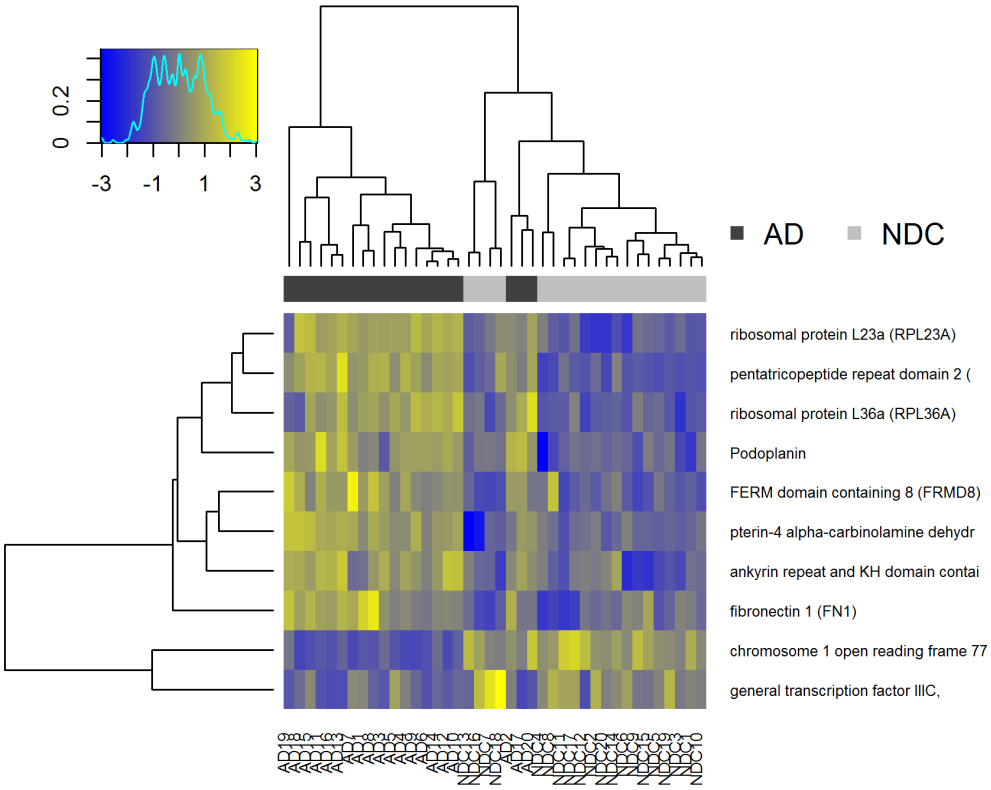
Alternatively, the function `plotFeaturesHeatmap()` plots intensities of all features given in the vector `features` (represented by BRC-IDs) as a heatmap. If `description` is `TRUE` (default: `FALSE`), features will be described via protein names instead of UniProtKB accessions. Again, if `output.path` is not `NULL`, the heatmap will be saved as a 'tiff' file in `output.path`.

```
> plotFeaturesHeatmap(features=selectFeatures.results$features, elist=elist,
+     n1=20, n2=20, description=TRUE)
```



As an alternative to `plotFeaturesHeatmap()` the function `plotFeaturesHeatmap.2()` which is based on the *gplots* function `heatmap.2()` gives similar plots with some additional information. Apart from that both functions `plotFeaturesHeatmap()` and `plotFeaturesHeatmap.2()` are analogous.

```
> plotFeaturesHeatmap.2(features=selectFeatures.results$features, elist=elist,
+     n1=20, n2=20, description=TRUE)
```



Finally, the function `printFeatures()` creates a table containing the selected biomarker candidate panel as well as additional information for results inspection. If `output.path` is defined, this table will be saved in a 'txt' file ('candidates.txt').

```
> elist$E <- round(elist$E,2)
> printFeatures(features=selectFeatures.results$features, elist=elist)[,-2]
```

```
        BRC   AD1   AD2   AD3   AD4   AD5   AD6   AD7   AD8   AD9  AD10  AD11  AD12  AD13
1    3 14 13  9.59  9.53  8.98  9.52  9.96  8.95  9.64  9.16  9.23  9.23  8.94  9.04   8.9
2    11 3 15 10.73 10.55 10.57 10.64 10.85 10.93 10.53 10.81  11.1 11.22 10.54 10.88 11.12
3     14 4 3 11.83  11.3 10.83 10.72 10.59 10.43 11.19  12.2  10.5 10.62  11.1 10.69 11.37
4    15 12 9  9.68 10.04 10.57  10.4 10.03   9.9  9.61 10.24 10.19 10.83 10.67 10.85 10.95
5    17 9 11 12.52 12.49  12.6 12.68 12.05 12.04 14.02 13.22 12.19 12.35 12.84 12.36 12.46
6    24 17 3  9.26  8.74  9.66  9.32   9.3   9.1    10 10.04 10.15  9.89 10.03  9.63 10.14
7     25 3 7 11.85 12.99 12.25 11.57  11.8 11.71 11.59 11.81 11.53 12.01 11.88 11.61 12.09
8   25 11 21 11.31  11.2 11.73 11.71 11.18 11.14 11.25 11.67 11.48 11.58 11.78 11.47 12.45
9    29 4 13 12.52 12.26 12.63 12.65 12.67  12.8 12.78 12.67 13.15 13.05  12.6 12.86 12.99
10 48 19 11  8.89  9.88  8.17  9.49  9.29  9.52   9.4  9.03   9.5  9.38 10.68  9.69 10.19
    AD14  AD15  AD16  AD17  AD18  AD19  AD20  NDC1  NDC2  NDC3  NDC4  NDC5  NDC6  NDC7
1   9.22  9.65  8.86   8.8  9.27  9.01  8.99  9.58 10.13  9.44  9.64  9.57  9.91 10.76
2  11.03 10.82 10.63  10.9  9.91 10.02 11.51  9.84  9.91  9.41  9.83 10.14  9.85  9.69
```

```
3  10.66 11.39 11.03 10.43 11.08  11.6 10.42 10.52 10.23 10.65  9.47  9.94 10.71  9.63
4  10.26  10.3 10.65 10.41  10.5 10.52 10.08  9.68  9.82   9.1   9.9  9.12  8.68  9.53
5  12.15 12.08 12.48 12.48 12.84 13.35 11.86 11.53 11.39 11.33 11.87 11.88 11.41 11.07
6   9.58 10.51  9.98  9.64 10.68 10.64  9.54   8.4  8.92  8.23  8.94  8.24  9.06  8.35
7  12.24 11.69 11.73 12.62 11.49 12.43 14.09 13.56 13.02 12.68 12.93 12.97 12.83 12.69
8  11.56 11.76 11.55 10.92 11.47 11.15 11.57 10.31 10.63 10.21  9.98 10.19 10.27 10.62
9  13.05 13.17 12.72 12.07 13.34  11.4 12.71 11.24 10.58 11.67 12.03 11.56  10.6 11.89
10  9.45   9.4  9.57 10.02  9.29  9.64  9.29  7.51  8.63  7.83  6.52  8.48  8.42  8.76
    NDC8  NDC9 NDC10 NDC11 NDC12 NDC13 NDC14 NDC15 NDC16 NDC17 NDC18 NDC19 NDC20
1  10.16  9.39   9.5  9.97   8.9  9.19  9.66  9.36  10.3  9.96 11.14 10.22  9.51
2   9.91 10.35  9.91  9.91  9.63 10.17    10 10.05 10.28 10.38  10.1  9.87  9.96
3   9.64 10.64 10.23  9.48 10.43 10.28  10.2 11.15  9.75  9.58 10.04 10.16 10.39
4   9.42  8.88   9.6  9.21  9.91  9.91 10.44  8.78  9.55  9.84  8.96  9.27  9.99
5  13.23 11.02 11.18 11.04 11.66 11.61 12.08 11.25 11.14 11.42 11.19 11.56 11.82
6   8.63  8.38  8.82  7.85  8.66  5.92  8.47  8.36  6.42  8.55  8.22  8.41  8.05
7  13.15 13.91 12.84 14.21 13.89 13.85 13.29 13.35 13.31  14.4 12.61 12.87 12.92
8  10.12 10.37 10.23 10.72 10.28 10.18 10.97 10.27 10.44 10.36 11.57 10.32 10.07
9  11.39  11.8 11.26 11.24 10.72 11.35 11.06 11.62 11.53 11.75 12.26 11.32 10.58
10  7.82  7.97   8.7  8.35  8.55  8.28  8.58  8.86  8.76  8.68  8.62  8.71  8.41
```

References

[1] Turewicz M, Ahrens M, May C, Marcus K, Eisenacher M. PAA: an R/Bioconductor package for biomarker discovery with protein microarrays. Bioinformatics (2016) 32 (10): 1577-1579. doi:10.1093/bioinformatics/btw037, PubMed PMID: 26803161.

[2] Love B: The Analysis of Protein Arrays. In: Functional Protein Microarrays in Drug Discovery. CRC Press; 2007: 381-402.

[3] Nagele E, Han M, Demarshall C, Belinka B, Nagele R (2011): Diagnosis of Alzheimer's disease based on disease-specific autoantibody profiles in human sera. PLoS One 6: e23112.

[4] Sboner A. et al., Robust-linear-model normalization to reduce technical variability in functional protein microarrays. J Proteome Res 2009, 8(12):5451-5464.

[5] Johnson WE, Li C, and Rabinovic A (2007) Adjusting batch effects in microarray expression data using empirical Bayes methods. Biostatistics 8:118-27.

[6] Baek S, Tsai CA, Chen JJ.: Development of biomarker classifiers from high- dimensional data. Brief Bioinform. 2009 Sep;10(5):537-46.

[7] Kuncheva, LI: A stability index for feature selection. Proceedings of the IASTED International Conference on Artificial Intelligence and Applications. February 12-14, 2007. Pages: 390-395.

[8] Abeel T, Helleputte T, Van de Peer Y, Dupont P, Saeys Y: Robust biomarker identification for cancer diagnosis with ensemble feature selection methods. Bioinformatics. 2010 Feb 1;26(3):392-8.