# Protein Microarray Data Analysis using the *PAA* Package

Michael Turewicz

November 23, 2015

## Contents

# 1 Introduction

## 1.1 General information

Protein Array Analyzer (*PAA*) 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 pre- processing (background correction, batch filtering, normalization) univariate feature pre-selection is performed (e.g., using the "minimum M statistic" approach - hereinafter referred to as "mMs", [1]). Subsequently, a multivariate feature selection is conducted to discover biomarker candidates. Therefore, 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"` (default) duplicate spots will 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 a exemplary dummy data set that comes with PAA and has been created from the real data described below.

```
> 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)
> save(dummy.elist, file=paste(gpr, "/DummyData.RData",
+    sep=""), compress="xz")
```

*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.* [2] (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: exceptionally not via `loadGPR()`).

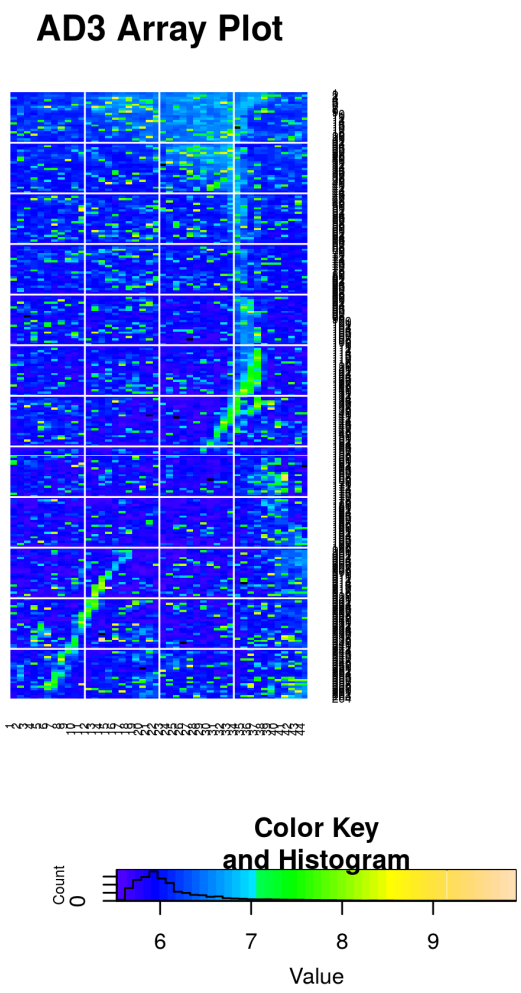
```
> cwd <- system.file(package="PAA")
> dir.create(paste(cwd, "/demo/demo_output", sep=""))
> output.path <- paste(cwd, "/demo/demo_output",  sep="")
> load(paste(cwd, "/extdata/Alzheimer.RData", sep=""))
```

# 3 Pre-processing

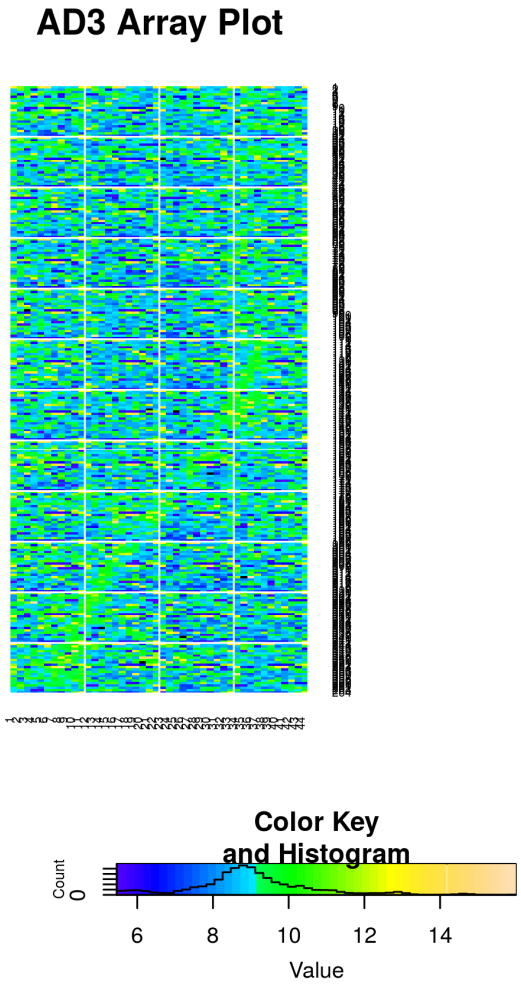
Before pre-processing the microarrays should be inspected visually for any strong spatial biases. In *PAA* this can be done for *ProtoArrays* using the function `plotArray()` which plots the imported *ProtoArray* data in the original arrangement mimicing the original scan image. Thus, `plotArray()` is a visualization tool that can be used to visualize *ProtoArrays* 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 pre-processing 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,
+   protoarray.aggregation="min", colpal="topo.colors")
```




```
> plotArray(elist=elist, idx=3, data.type="fg", log=FALSE, normalized=FALSE,
+   protoarray.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.

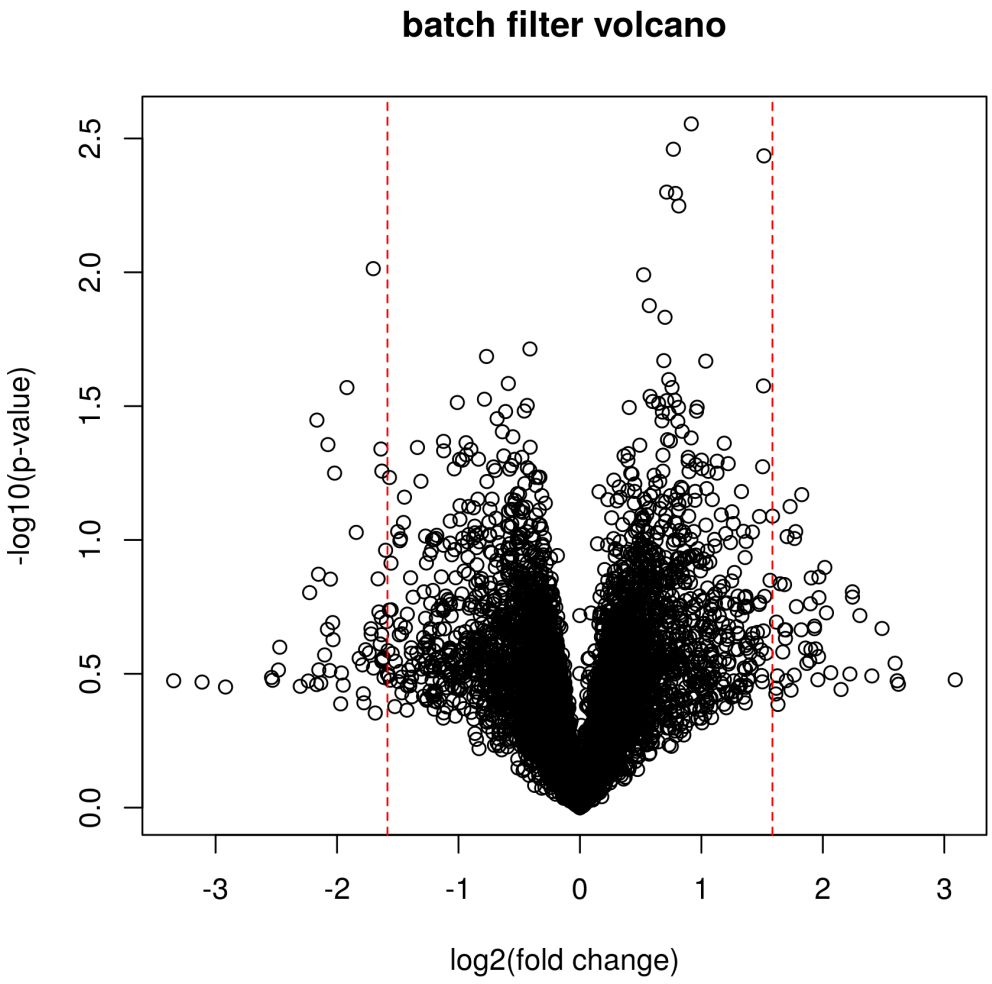
For background correction *limma*'s function `backgroundCorrect()` can be used:

```
> 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 pre-processing. 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 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 <- batchFilter(elist=elist, lot1=lot1, lot2=lot2, p.thresh=0.001,
```

```
+ fold.thresh=3)
```

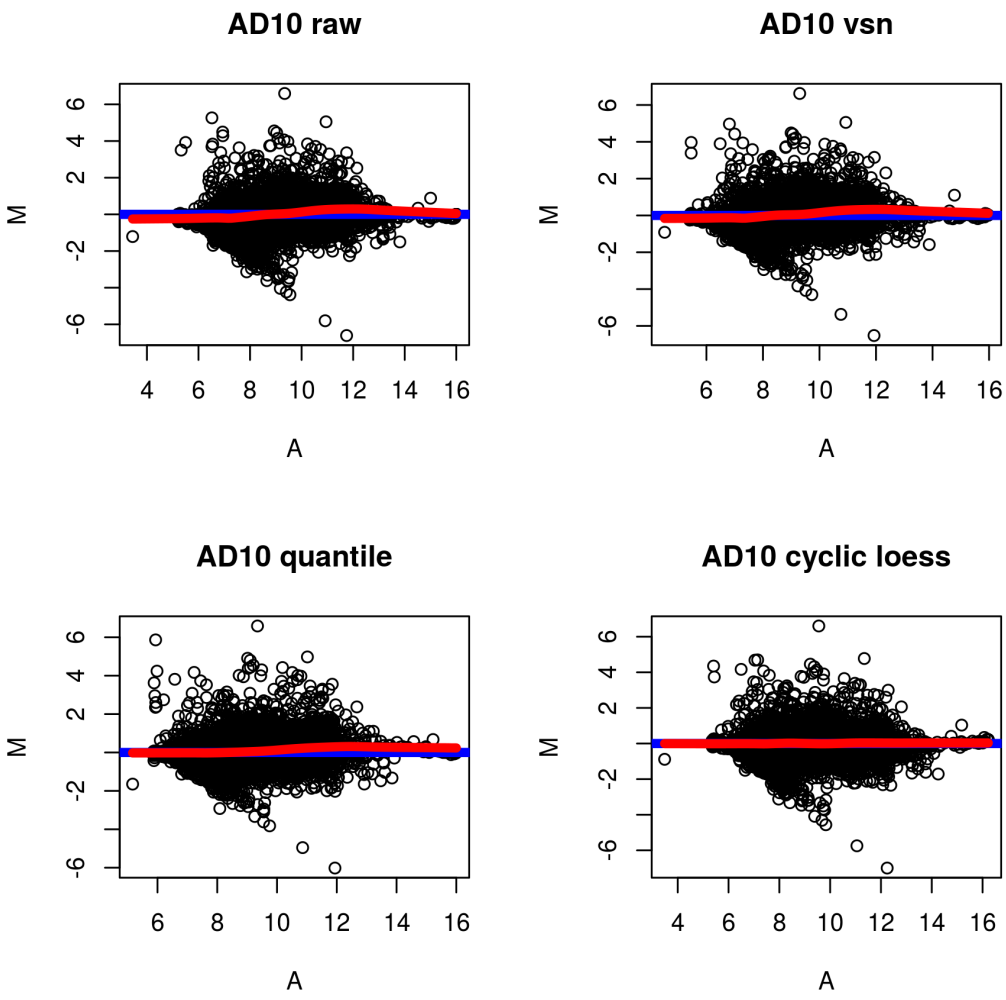



Another important step in pre-processing is normalization. To assist in choosing an appropriate normalization method for a given data set, *PAA* provides two functions: `plotNormMethods()` and `plotMAPlots()`. `plotNormMethods()` draws boxplots (one boxplot per sample) of raw data and data after all kinds of normalization provided by *PAA*. For each normalization approach sample-wise boxplots are 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.* [3]) 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 [4]. To use `batchAdjust()` the targets file information of the *EList* object must contain the columns *"Batch"* and *"Group"*.

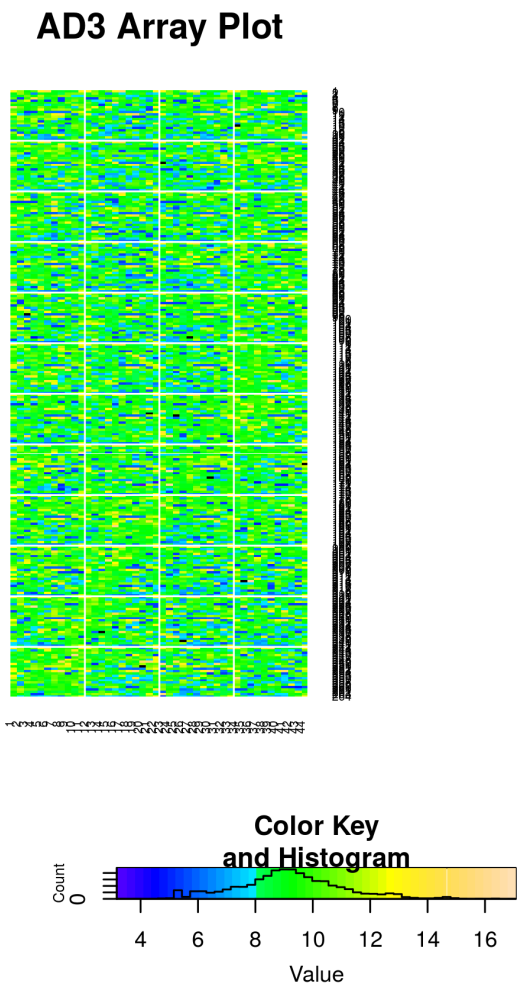
```
> elist <- batchAdjust(elist=elist, log=TRUE)
```

```
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 pre-processing, the corrected data can be inspected via `plotArray()` again. E.g., now the plot of the ProtoArray

plotted above shows that the revealed spatial bias is less obvious after pre-processing.

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



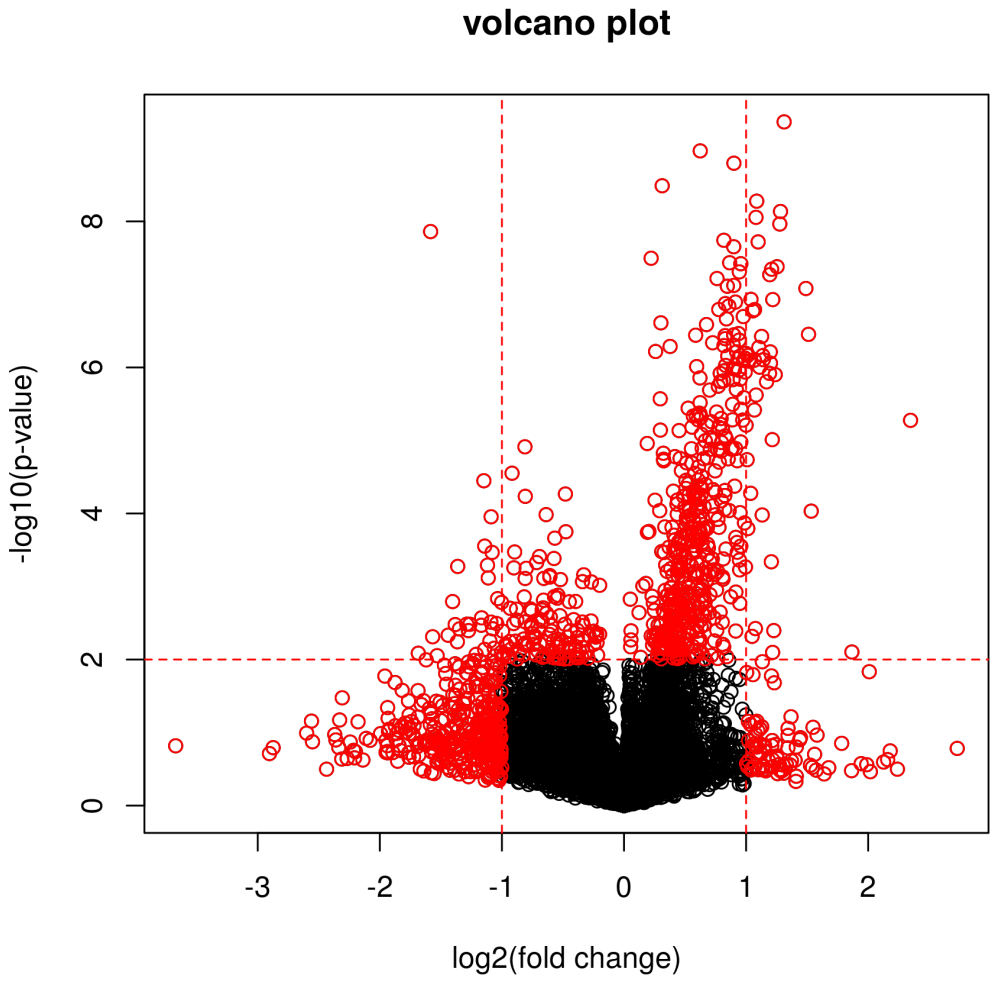

Since for further analysis also data in original scale will be needed, a copy of the *EList* object containing unlogged data should be created.

```
> 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 t-test p-values or mMs as well as 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. Therefore, 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",
+ p.thresh=0.01, fold.thresh=2)
```

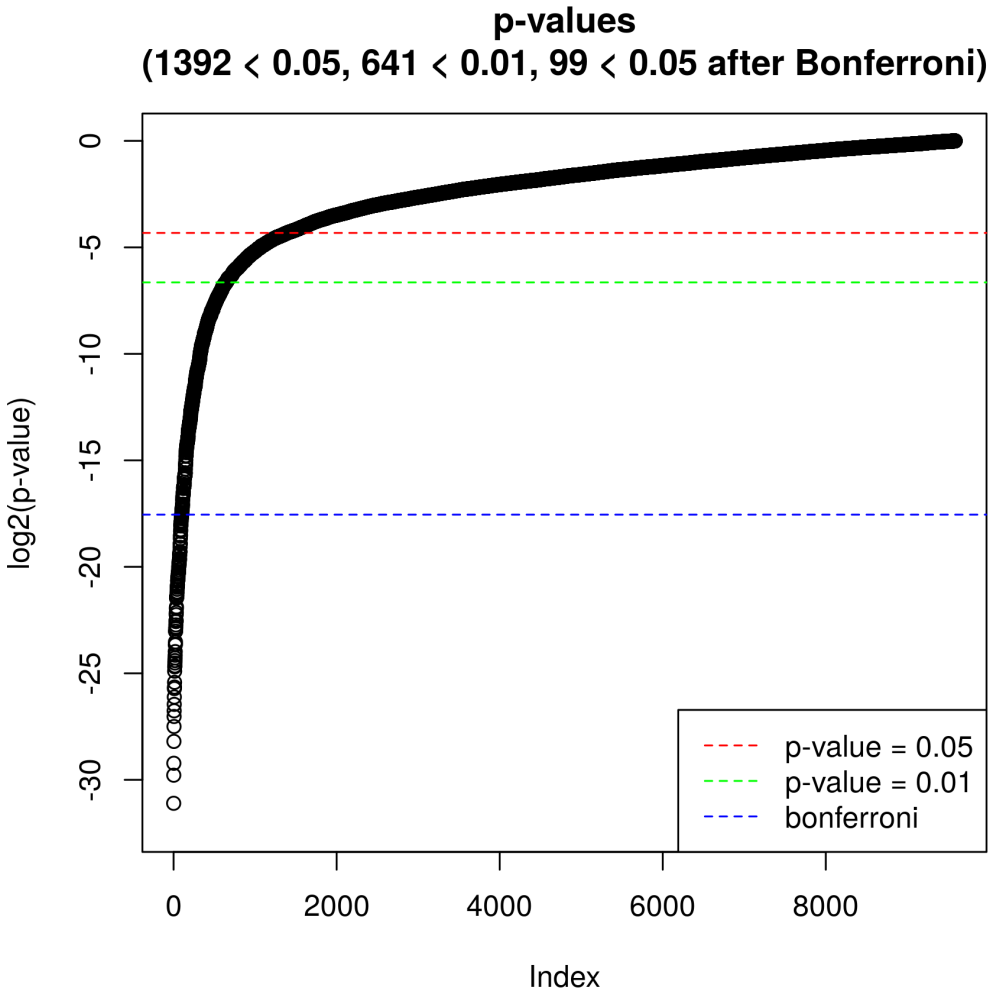



Here, an example with `method="mMs"` is given:

```
> mMs.matrix1 <- mMs.matrix2 <- mMsMatrix(x=20, y=20)
> volcanoPlot(elist=elist.unlog, group1=c1, group2=c2, 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.unlog, group1=c1, group2=c2, method="tTest")
```

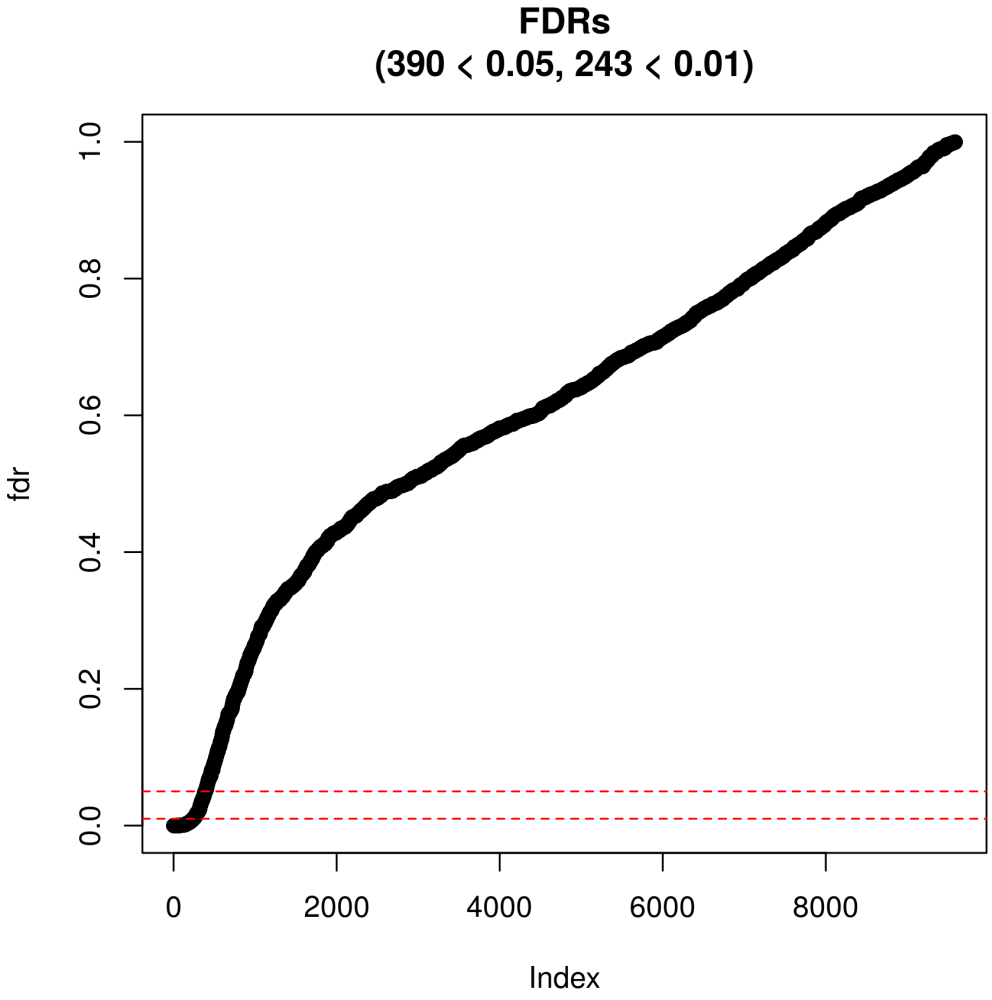



Here, an example with `method="mMs"` is given:

```
> mMs.matrix1 <- mMs.matrix2 <- mMsMatrix(x=20, y=20)
> pvaluePlot(elist=elist.unlog, group1=c1, group2=c2, 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.unlog, group1=c1, group2=c2, method="tTest", adjust=TRUE)
```




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

```
> mMs.matrix1 <- mMs.matrix2 <- mMsMatrix(x=20, y=20)
> pvaluePlot(elist=elist.unlog, group1=c1, group2=c2, 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 0.501988087542915 0.0241860325286354 0.330335726331277
3  1 2 15   0.320875589014851 0.634210502468438                 1                1
4  1 2 17   0.178493270666023  0.52723150928707  0.150422391245528 0.835015538626552
5  1 2 19   0.271589613948768  0.59793213561654  0.243589743589744 0.835015538626552
6  1 2 21  0.0707666042203721 0.394970369965038 0.0457380457380457 0.484098604098604
7   1 3 1  0.0284911744794212 0.268607060873835                 1                1
8   1 3 3 0.00921968894879151 0.142280968989526               0.5 0.910368401063426
9   1 3 5 0.00592168460061219 0.106226964810385  0.053014553014553 0.484098604098604
10  1 3 7   0.806173147194643 0.916805614435089  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 2548.69868040326
3   1.10224097047813 450.550786587285 408.758881818595 413.716205403229 417.428604869084
4  0.595491172830504 1518.91515397688 2550.69297964089 1215.20762270891  1689.1444916955
5  0.454051309774644  2530.1344545838 5572.35360875749 1824.40003073847 1864.76465290003
6  0.759043204421859 2636.25422481143 3473.12802414112 2248.99036532219 2924.27002197574
7   1.26294406395723 484.784697258339 383.852864979108 446.558759285452 349.263285007633
8   1.48059072464481 692.098536568907 467.447570114245 556.156389770551 455.411881000121
9   1.36023092627671 1991.44021666422   1464.045683857 1873.36646648453 1436.18751389589
10 0.908397966805546 818.482441505167 901.017474074084 730.232223314622 469.463840773123
              sd.AD           sd.NDC
1  1643.99677640782   564.4302706304
2  2970.84080119717 18364.0058765935
3  165.734824607582 82.0150145643133
4  1062.44868297829 3151.76500089472
5  2444.44412207379 11793.1087294661
6  1276.31835442119 1553.96096388882
7  154.993619177929 122.909766431917
8  339.123207021637 93.5019966245783
9  718.441427077343 323.488045744026
10 433.002023903253 1422.22599366312
```

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 pre-selection

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 pre-selection"* 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", 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 pre-selection can be performed (usage via `preselection.method`). As univariate pre-selection methods mMs (`"mMs"`), Student's t-test (`"tTest"`) and mRMR (`"mrmr"`) are supported. Alternatively, no pre-selection can be chosen (`"none"`). This approach is similar to the method proposed in *Baek et al.* [5].

**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* [6]), 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 and is described in *Abeel et al.* [7].

`selectFeatures()` takes an `EListRaw` or `EList` object, group-specific sample numbers, group labels and parameters choosing and setting up a univariate feature pre-selection method as well as 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",selection.method="rf.rfe",subruns=2,candidate.number=1000,
+     method="frequency")
```

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

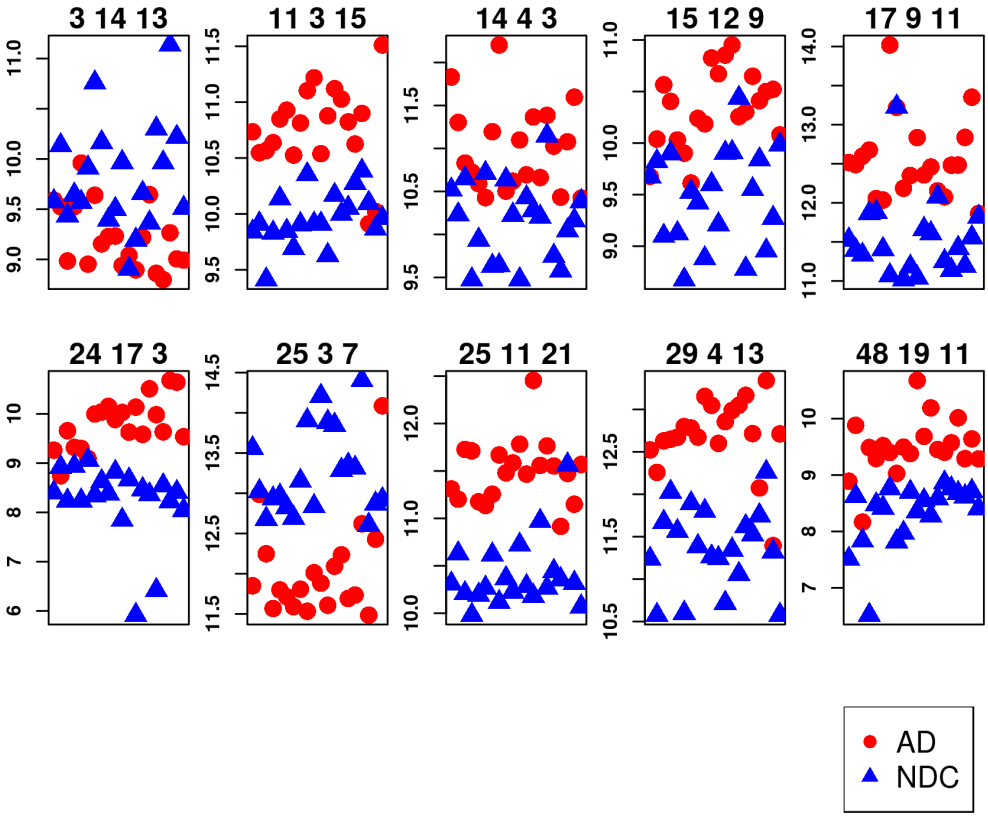
Because runtimes would take too long for this vignette *PAA* comes with pre-computated `selectFeatures.results` objects stored in '.RData' files. These objects can be loaded as follows:

```
> # 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

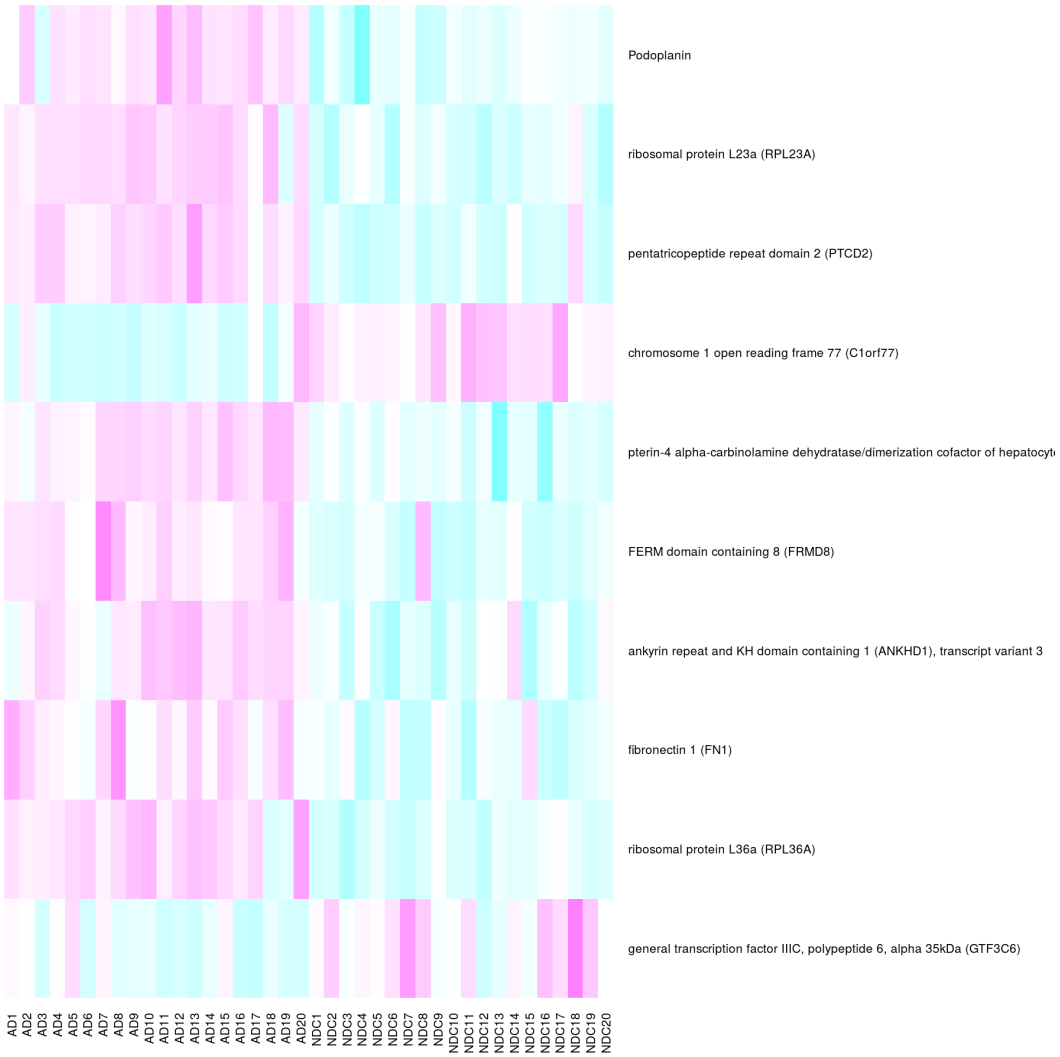
After the selection of a feature panel, these features should be validated by manual inspection and evaluation for further research. To aid results inspection, *PAA* provides several functions. The function `plotFeatures()` plots the intensities of all features (represented by BRC-IDs) that have been 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 in 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 uniprot 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)
```




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').

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

| | BRC | AD1 | AD2 | AD3 | AD4 |
|---|---|---|---|---|---|
| 1 | 3 14 13 | 771.433440157475 | 737.068543182098 | 506.509024617034 | 736.588851000494 |
| 2 | 11 3 15 | 1704.11488224753 | 1496.65528183652 | 1517.25551432498 | 1593.07099727861 |
| 3 | 14 4 3 | 3645.54841240841 | 2526.6168621342 | 1821.66550063235 | 1691.67557516903 |
| 4 | 15 12 9 | 818.461533980501 | 1052.00001705278 | 1515.41303856353 | 1355.18754782922 |
| 5 | 17 9 11 | 5853.3296132514 | 5749.39707713476 | 6201.69099908049 | 6547.8009327327 |
| 6 | 24 17 3 | 614.427688242766 | 428.773390294605 | 808.178343313086 | 638.692844649007 |
| 7 | 25 3 7 | 3692.00181196141 | 8107.28294808855 | 4863.2298094671 | 3034.55994415583 |
| 8 | 25 11 21 | 2540.90741558512 | 2356.0447976407 | 3388.29729602303 | 3353.9003414911 |
| 9 | 29 4 13 | 5886.536213952 | 4897.27146044234 | 6338.32649855009 | 6423.35661951507 |
| 10 | 48 19 11 | 474.498937861052 | 943.550167460079 | 287.27425910161 | 718.066645073774 |

| | AD5 | AD6 | AD7 | AD8 | AD9 |
|---|---|---|---|---|---|
| 1 | 995.953196816827 | 495.556187999054 | 797.291831845909 | 570.27097906642 | 599.658424542806 |
| 2 | 1845.74002333541 | 1948.74725704858 | 1474.82659493596 | 1799.92812279791 | 2200.5837969357 |
| 3 | 1541.4980807955 | 1378.19303071998 | 2344.28233212592 | 4718.52573314618 | 1451.16540951054 |

```
4  1045.65878862808 956.618003092471 783.051188927561 1209.45324357601  1165.6702531658
5  4253.92007266518 4199.33245714912 16615.5838810269  9552.1022536634 4661.14941806543
6  630.131117379421 547.404006663499 1022.31269475795 1052.05465948835 1139.01947331139
7  3562.14387572132 3359.25151114482 3081.26920034005 3584.23934006449 2965.24403870638
8  2313.50890173121 2249.69441173037 2443.60003229657 3256.44772793452 2856.18931199357
9  6527.46112288469 7123.30942029013 7041.87220218448 6531.57635614536 9109.59734324713
10 627.005778580238  734.72768160691 675.962981565189  522.11795841714 721.783383669617
                AD10             AD11             AD12             AD13             AD14
1   601.61374490539 490.646297749198 526.562459230263 476.202714282601 595.908446740426
2  2382.32674141506 1488.04707151358 1882.53426964658 2226.14085257105 2087.15591730029
3  1573.14801575871 2193.52319524127 1656.90216562732 2639.45630906393 1620.43972141502
4  1815.24244144851 1631.98834508922 1851.07522460675 1980.93948465889 1223.80001459399
5  5220.99275618885 7308.29349932005 5244.26858654111 5628.40229135569 4556.03527897316
6  946.230248346506 1045.08920541636 792.019550730672 1126.04229910868 767.655166998841
7  4129.50138698448 3774.02043662051 3121.82055142515 4367.46882822357 4820.89532950348
8  3072.06801981497   3522.706248661 2830.61009823552 5613.49707855303 3017.52725160791
9  8466.97464288742 6207.32953054157 7413.92324894751 8127.03826735894 8479.48909101352
10 666.072166782071 1637.70381756676 823.561959466415 1168.77896600105 700.500559561621
                AD15             AD16             AD17             AD18             AD19
1  800.762961411123 465.094844578007 445.207793478522 615.269196751757 513.794174455715
2   1812.4734896282  1579.9897938486 1910.13596078838 963.204464145622 1039.56722703483
3  2675.70767816961 2086.54165587402 1383.74794708655 2161.70859900922 3097.43992992444
4  1261.86846682952  1604.6507431494 1361.71947755349 1449.23812040171 1470.13378995336
5  4316.70678856553 5720.89472533376 5728.10060944017 7317.28485835265 10456.0759202355
6  1458.02539474764  1012.2816060924 795.484757992508 1639.86772626606  1598.9827428381
7  3308.27621209006 3408.31206669194 6301.11500033229 2867.16665878206 5514.46894147623
8  3479.37457669234  3004.3988795354 1932.60122159644 2843.65026626264 2270.98257648292
9  9189.19488788393 6725.36483581996  4312.8026934387  10383.248560208 2694.38987693426
10 677.177187392846 761.373267213567  1034.7509030767 626.423425652316 798.246305693239
                AD20             NDC1             NDC2             NDC3             NDC4
1  509.465336985822  764.03963065196 1123.96058976114 693.528823039918 799.441404968981
2  2920.18462185127 915.637590277119 963.775473473381 681.261552556144 910.853761517289
3  1373.00177127862  1466.6185534305 1201.50931505021 1605.86142777358 711.236591520915
4  1081.85231306212 817.470628339066 904.872626137077 548.137800902123 956.146494368586
5  3725.32360318959 2951.07948417374 2692.77844181851 2581.13450084882  3733.2264317801
6  743.221771652252   337.7186683345 483.322546000382 301.125669591757 490.183249560809
7  17403.0639517425 12052.8374235595 8328.71569741779 6565.68128186054 7820.32532325889
8  3036.79006143744 1269.47160954923 1583.18497790231 1183.43892640827  1010.0646849764
9  6712.43148200038 2413.12605828541 1525.97853111371 3263.39959267821 4172.84362566436
10 624.782210065881 182.295560156436 395.064365250821  228.18649659473 91.5705107087877
                NDC5             NDC6             NDC7             NDC8             NDC9
1  759.776623181678 962.916214573148 1733.86920351672 1147.94060838566 672.772808953259
2  1126.15524889113 921.019121382185 828.259446105578 959.876577904814 1307.39432533322
3  980.725384642246 1677.75573543924 793.808207526627 797.219622725495 1594.94839974005
4  558.147474737821 408.867897234498 738.299108244182 685.261863738142 471.432884591676
5  3776.90201491392 2715.13778750748 2150.20825861469 9602.21392427571 2072.68509413821
6  301.947434088494 534.917476299248 325.670250527034 395.694498724042 333.421597010831
7  8010.06949392926 7263.97429482264 6623.14680993597 9103.45485471742 15346.1426689505
8  1170.24785888503 1231.86541332021 1571.13913180881 1109.19382331243 1320.66832709391
9  3025.53385619706 1552.31727701722 3797.72742810443 2675.01132406589 3567.42918673871
10 356.082351062428 342.419012324196 433.833805230495 226.463945020404 250.973191093232
               NDC10            NDC11            NDC12            NDC13            NDC14
1  724.162647196417 1001.28705686367 478.929948819404 585.835579721394 809.352736702492
2  962.915795437027 962.755624324433 792.830629136654 1150.43186525266 1022.86203105949
```

```
3  1198.12849426954 711.651604930737 1378.43688330882 1244.47760101974 1178.98996424869
4  776.550195469829 592.116823353938 958.769969338747 962.825344005406 1385.80583194318
5  2321.60381731547 2110.06898430524  3245.1113983374 3121.50585469776 4328.19955684069
6  451.007734997018 229.958238114366 405.347056390551 60.3389420405944 353.903428438434
7  7321.80836561542 18936.6907846862 15152.3315873422 14764.4527060412 10013.2907378493
8  1198.03636977277 1688.16655748438 1246.92040081615 1161.51333026481 2006.71616640872
9  2458.90563107894 2423.92198962608 1685.40686957677 2604.02295069198 2131.81994710554
10 415.262894883498 326.707237188373 374.407955183011 310.738299976302   382.7139538853
               NDC15            NDC16            NDC17            NDC18            NDC19
1  659.263177504851 1261.79825082435 997.809726651286 2249.49020487926 1191.21535312782
2  1062.35911151592  1239.7749569386 1337.00488957594 1095.93306259692 933.945959870117
3  2272.33646592075 860.911959012863 764.552316277617 1056.43031305259 1146.79872232844
4  439.862622783571 750.310216501293 916.813965922586 496.485083095296 618.368266966113
5   2438.8619602911 2254.96149654015 2740.74372189604 2343.23103478011 3013.41709592446
6  328.846237238928 85.6032474266259 373.602655575968 298.889309887519 340.279297203394
7  10459.2377467132 10185.6344502934 21662.2153856729 6234.67987741103  7484.5925757424
8  1230.74191006278 1392.05483748243 1311.21516047789 3032.16037497358 1276.96111522988
9  3149.45015415533 2947.81436316397 3446.34834802618 4908.87673098921 2563.74502597501
10 465.668088868463 432.798190503331 409.598843975691 393.066754268871 420.218891517836
               NDC20
1  729.347137099923
2  996.619534817372
3  1337.30346375448
4   1014.5286162758
5  3612.07550934687
6  264.345154327535
7   7765.6353086002
8  1074.54696516573
9  1527.93431315706
10 339.414030657283
```

## References


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

[2] 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.

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

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

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

[6] 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.

[7] 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.