Protein Microarray Data Analysis using the *PAA* Package

Michael Turewicz

October 14, 2015

Contents

1 Introduction **2**
1.1 General information 2
1.2 Installation 2

2 Loading PAA and importing data **3**

3 Pre-processing **4**

4 Differential analysis **7**

5 Feature pre-selection **11**

6 Feature selection **12**

7 Results inspection **14**

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 *comment: (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/. *comment: 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).

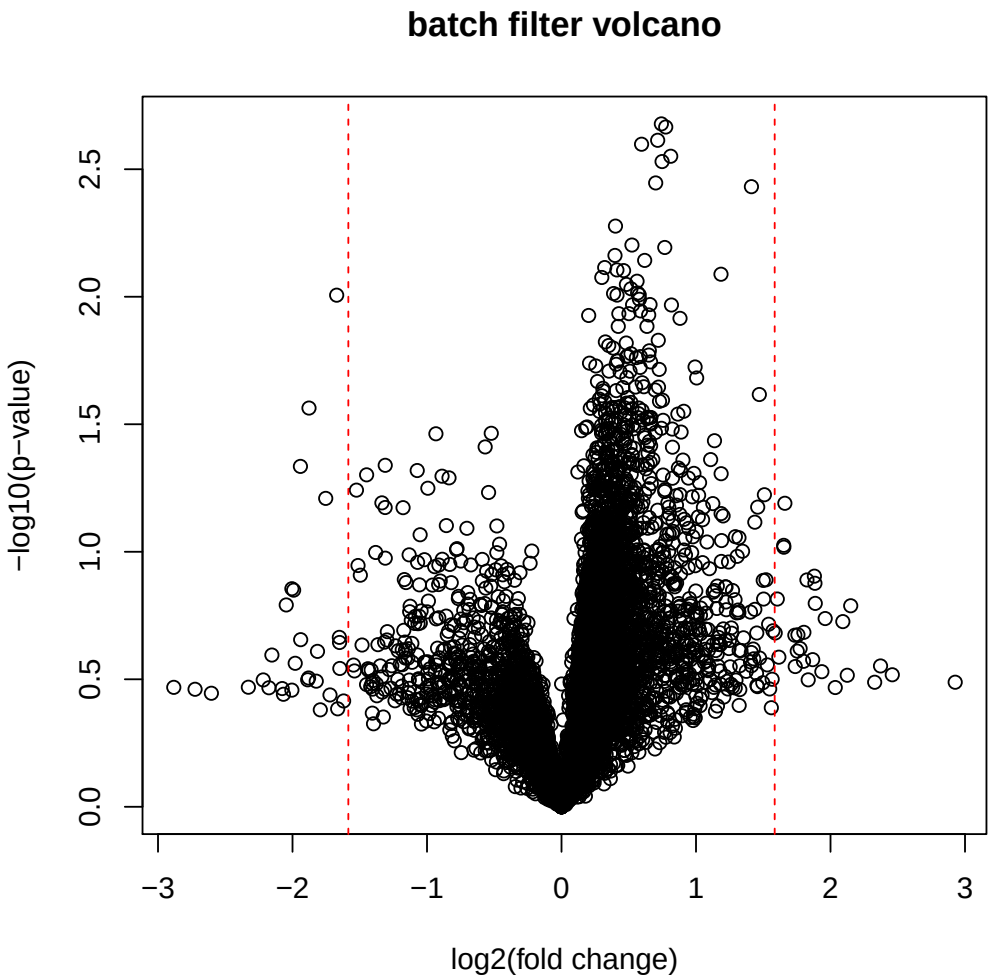
In the following code chunk, the *PAA* installation path (where exemplary data is located) is localized, the new folder 'demo_output' (where all output of the following analysis will be saved) is created, and the exemplary data set is loaded (note: exceptionally not via `loadGPR()`).

```
> 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

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)
```



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

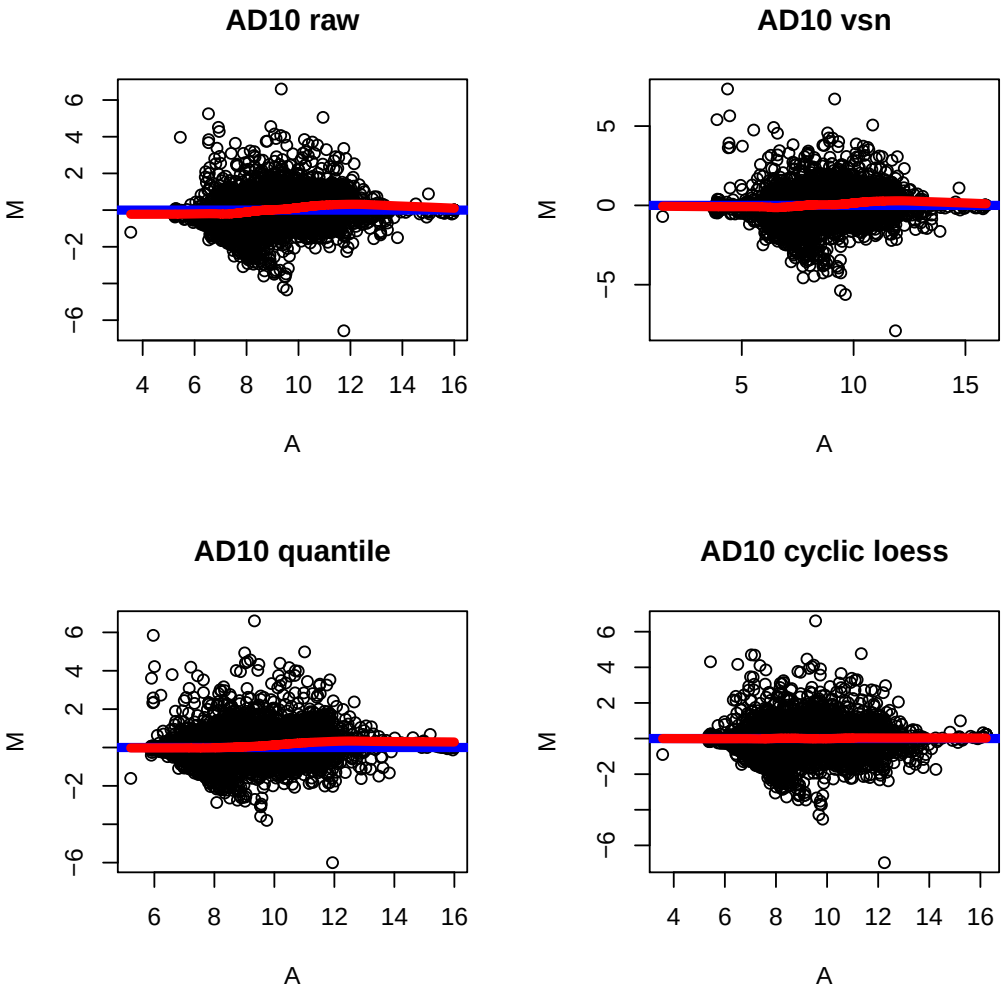
```
> library(limma)
> elist <- backgroundCorrect(elist, method="normexp",
+  normexp.method="saddle")
```

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 `"cycliclоess"`, `"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
```

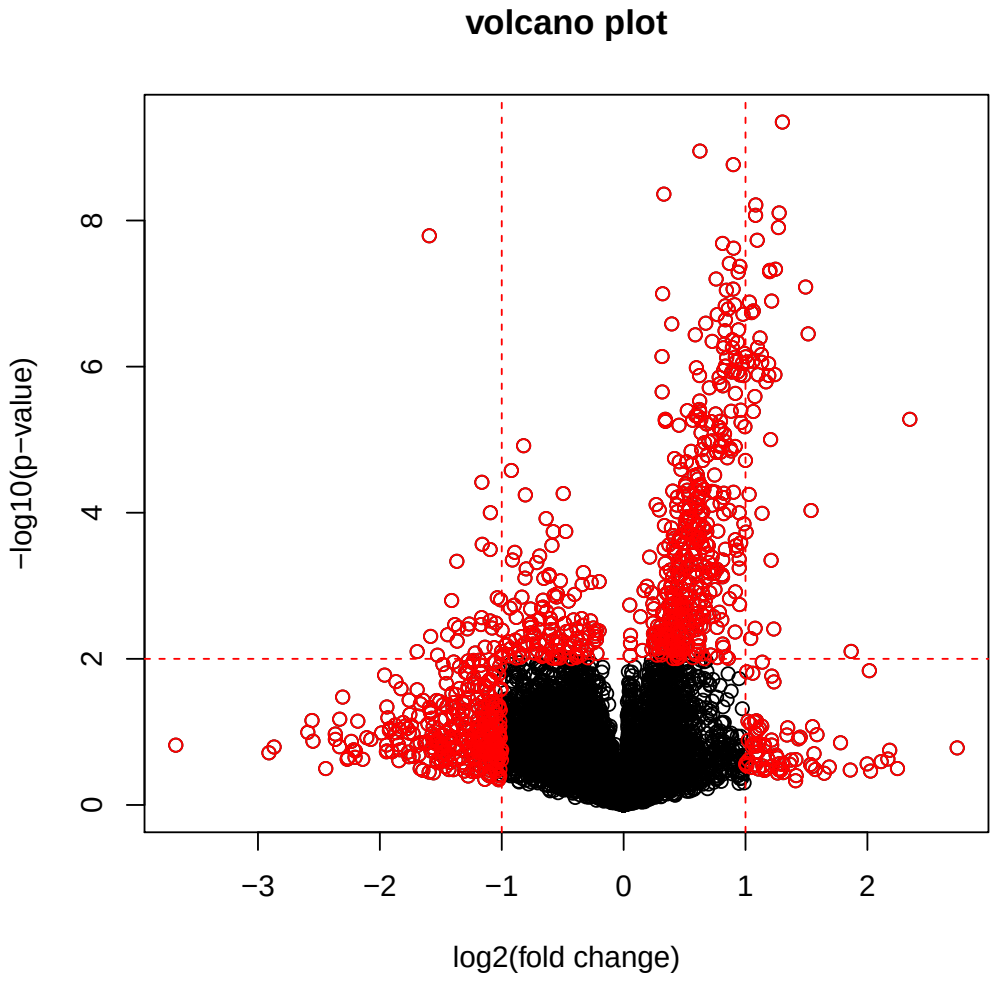
Since for further analysis also data in original scale will be needed, a copy of the *EList* object containing unlogged data should be created.

```
> 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)
```

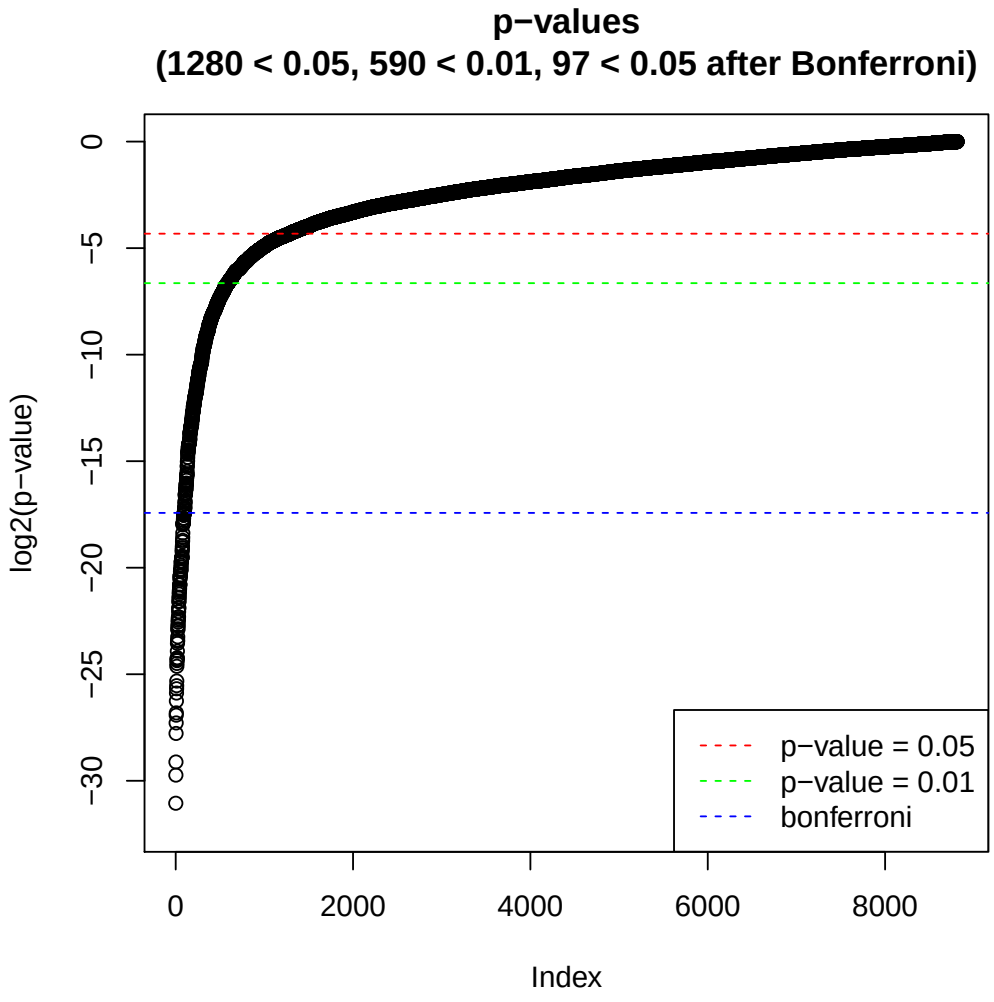


Here, an example with `method="mMs"` is given:

```
> 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. *comment: 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")
```

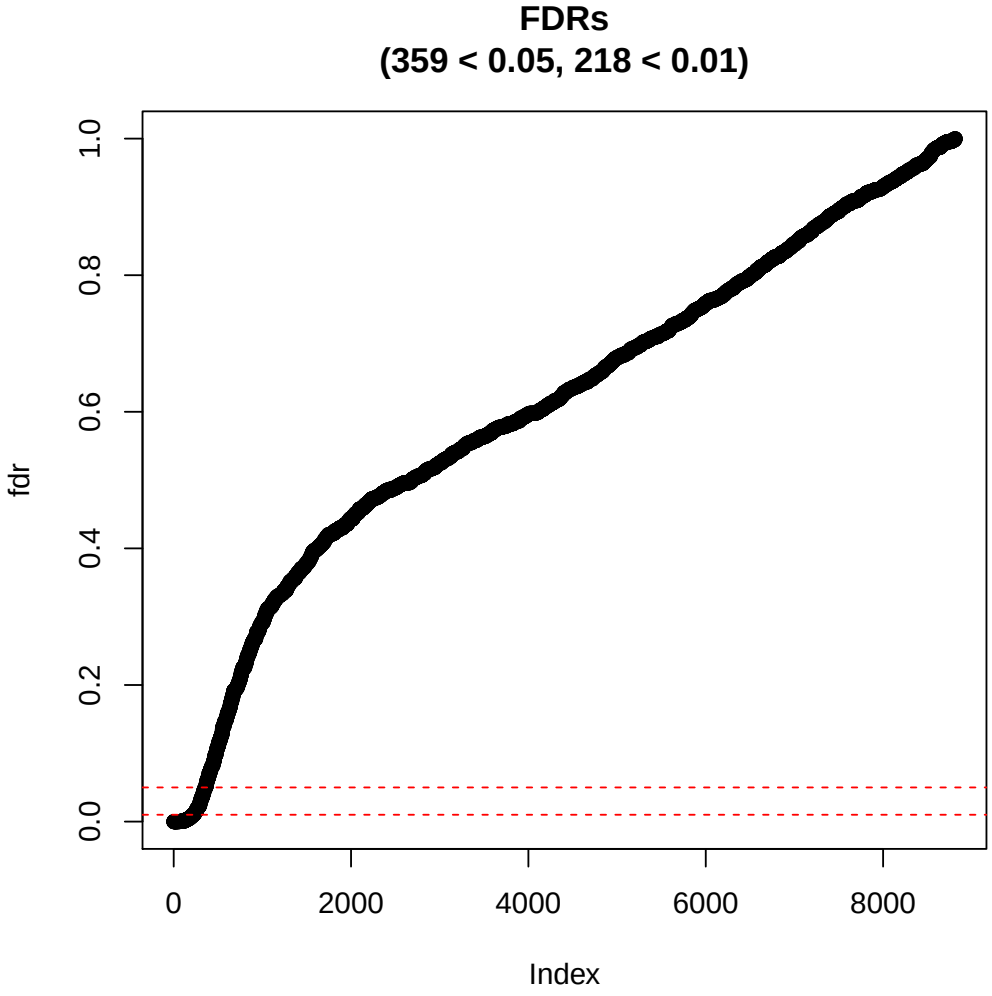


Here, an example with `method="mMs"` is given:

```
> 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    0.35310616699664 0.653515561033814  0.243589743589744 0.839889049396093
2  1 2 13   0.151047676620869 0.497495334555317 0.0241860325286354 0.339954552910469
3  1 2 15   0.322661412277615 0.633943943562643                  1                 1
4  1 2 17   0.179355969483044  0.52542198857641  0.150422391245528 0.836914478564923
5  1 2 19   0.270872133026703 0.596500776702732  0.243589743589744 0.839889049396093
6  1 2 21  0.0705685797508115 0.394619856182679 0.0457380457380457 0.494409794409794
7   1 3 1  0.0287047750392359 0.272016325183641                  1                 1
8   1 3 3 0.00964332079377624 0.146761073824877                0.5  0.91592184577011
9   1 3 5 0.00585967882460957 0.105606031659068  0.053014553014553 0.494409794409794
10  1 3 7   0.810874235449665 0.917242284304697  0.302494802494803  0.91592184577011
         fold.change          mean.AD         mean.NDC        median.AD       median.NDC
1   1.36424980900744 1385.39667514052 1015.50072867407 836.967141422825 856.924119132527
2  0.259812534239783 2174.63594761717 8370.01938332267 1301.78619646052 2539.13568716017
3   1.10187764526044 449.341637655752 407.796309861194 413.155296296935 417.153735113663
4  0.596681100872995 1514.67315107298 2538.49694394022 1209.86000885239 1677.15842610077
5  0.452784319410235 2514.10669279988 5552.54805659917 1823.88915456646 1845.85809953754
6  0.756982579985153 2623.03091375804 3465.11397106322 2243.35546297194 2912.01139165561
7    1.2628409159071  483.43447618784 382.815024520004 445.041330625547 348.039093254718
8   1.48323540665116 691.595735474642 466.275098594175 554.338819804953 453.925085866405
9   1.36256262849398  1987.7391153558 1458.82403772723 1862.02911541228 1431.38640822382
10 0.910535317015787 818.500619086081 898.922429245971 728.594780869617 468.248941989063
              sd.AD           sd.NDC
1  1652.18383300672 564.613228332769
2  2944.53946690674 18321.9607074207
3  165.514809954714 81.5402394496131
4  1059.83503973736  3132.9210970924
5  2423.24879814855 11762.9616782294
6  1287.13398591756 1559.47972714555
7  154.770438502943  122.65819388996
8  342.739574049016 93.0408286530952
9  719.952644954054 322.419391545683
10 438.081229970117 1419.40167042176
```

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")
```

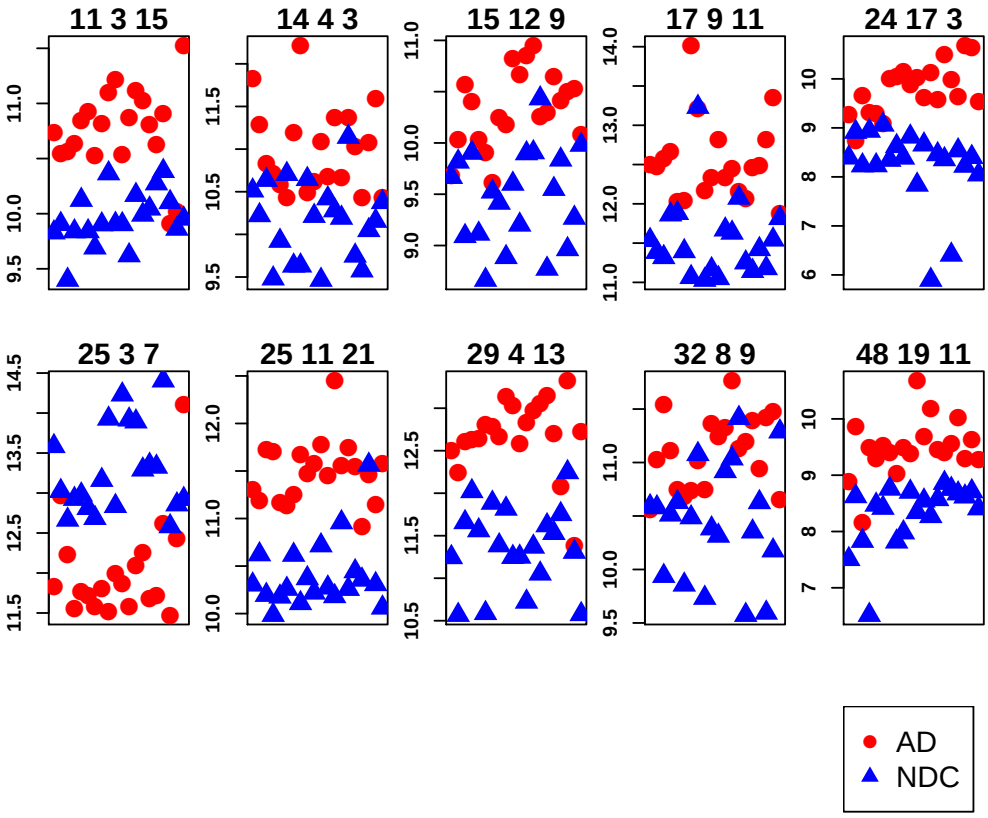
Because runtimes would take too long for this vignette *PAA* comes with pre-computated `selectFeatures.results` objects stored in '.RData' files. These objects can be loaded as follows:

```
> # 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

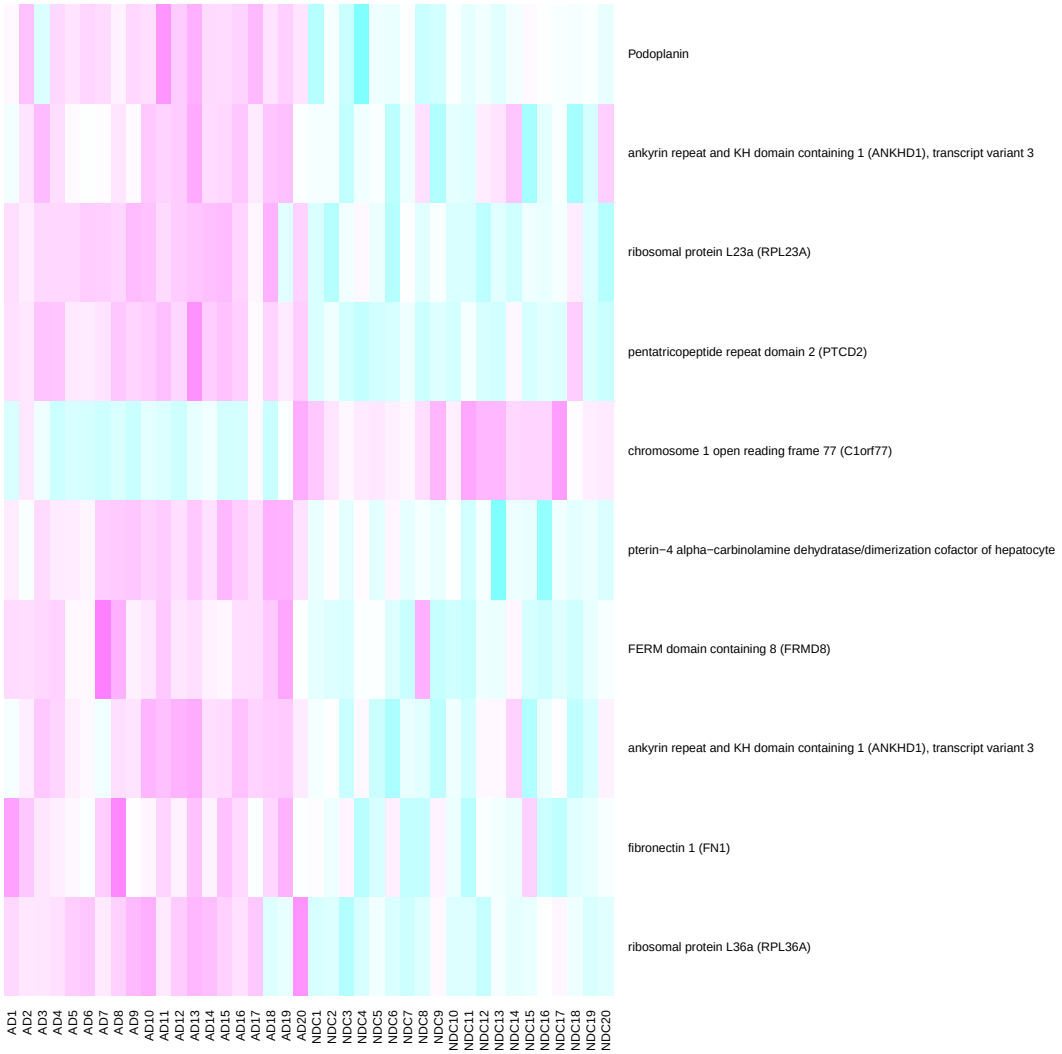
After the selection of a feature panel, these features should be validated by manual inspection and evaluation for further research. To aid results inspection, *PAA* provides several functions. The function `plotFeatures()` plots the intensities of all features (represented by BRC-IDs) that have been 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    11 3 15 1703.31837653865 1491.17023063965 1516.13683196889 1586.55323368053
2     14 4 3 3638.24183490315 2507.73791136244 1819.30785847308 1682.58958141381
3    15 12 9 821.091703810558 1050.40509812001 1516.12540728299 1351.48975396829
4    17 9 11 5787.75253046834 5671.34450215111 6126.47135063689 6486.17248013197
5    24 17 3 614.584148494639 429.887812284176 807.832729963429 637.320267615249
6     25 3 7 3635.53112882019 7990.41506031097 4796.06511893848 2997.11069054398
7   25 11 21  2531.7234406839  2338.0008855226 3374.42979464505  3335.5406174437
8    29 4 13 5818.84609184953  4835.6334896692 6260.78284330116 6362.76748930227
9     32 8 9 1508.37205423432 2087.86339225907  2978.2370633432 2213.84066817173
10 48 19 11 473.791048911754 934.424698346471 284.642146467594 717.019151760854
                AD5               AD6               AD7               AD8               AD9
1  1844.18473811195 1950.90757685586 1475.17205681948 1807.60560643943 2190.69780358162
2  1538.08167843364 1379.61110468666 2344.11280366484 4750.94013203375 1444.04641638651
3  1045.43023291406  957.86822096897 783.982260159628 1215.92596638622  1161.4676248529
```

```
4  4184.32057346599 4211.68367775125 16548.9454002678 9496.00340144831 4610.01164207187
5  628.713502128171 546.262623360453 1026.46069123642 1062.49526284506 1137.40108048132
6   3496.1236925539 3364.91208249702 3073.07492238391 3568.68829203055 2920.57842939983
7  2298.49366324812 2257.18667755705  2442.2015022878 3255.80024434939 2844.78353204362
8  6413.35466967837 7148.35699913156  7029.6785233379 6483.53515610306  9010.0506599242
9   1721.9208410507 1645.25275970219 1703.23229862491 2069.88235278295 1721.89110376598
10 624.855884458338 733.902457373689 677.491158879676 522.971461690363 720.366619428177
               AD10             AD11             AD12             AD13             AD14
1  2379.64797428159 1482.54752420343 1872.37884895698 2226.50561379619 2091.23667210811
2  1569.33972549855 2185.86487832947 1644.48695648081 2639.22584184982 1624.59842869766
3  1814.25014382062 1629.77663743834 1843.73723277642    1977.31021333 1224.86003133795
4  5160.59383843425 7224.05720531052 5171.41866728093 5581.33689854595 4573.58516928971
5  945.340867319505 1045.67374345179  790.70676358878  1120.7608235132 766.132711380178
6  4072.94914243742 3732.71254564852  3067.5497781666 4362.43051201276 4870.43126139058
7  3060.19956837113 3503.81728665358 2804.67540953222 5596.43197782138  3024.0396334285
8  8371.63051137636 6136.39944545229 7313.65738539658 8060.25564884267 8512.37512655366
9  2640.52576093206 2422.07715425284 2567.74886372465 3480.71808952069 2241.34765462906
10 665.113232367235 1639.29551732958  823.25224977212 1163.02333193623 698.671936926017
               AD15             AD16             AD17             AD18             AD19
1  1796.81816368656 1579.97385830164 1917.19655977339 960.936531580009 1039.50167046324
2  2652.54851031753 2084.63869961729 1386.05222754699 2153.90581797748 3094.71985037169
3  1256.71055501353 1606.42129243192 1361.88369342244 1448.36385882365 1474.49508102757
4  4279.52130097711 5660.75632274034 5726.46013860462 7230.53130393488 10423.2392011002
5  1452.81151801677 1012.32365352872 796.266913559128 1639.73697779925 1600.34919564201
6  3280.12411491222 3378.87836977646 6281.24057929661 2843.95187919414 5541.49227947119
7  3448.72851459466 2991.50818247592 1934.22744248055 2827.28687043181 2260.66124065267
8  9106.25565956143 6656.22834904811 4308.19296760313 10253.0880243242 2682.87441843273
9  2344.26964721439 2680.98712353003 1974.62737669058 2745.07148551982 2850.07005266812
10 674.536469180021 760.323282923353 1037.54546373286 625.698066763681 797.391558657625
               AD20             NDC1             NDC2             NDC3             NDC4
1  2946.26557640838 909.207097987646 958.624894828959 674.880833606733 915.346904155452
2   1385.5875102269 1460.87396435347 1195.22929635565 1592.23215304582 714.553733299817
3  1083.85602771078 812.812311592161 901.354447111923 545.056729374537 956.891073918903
4  3750.64180669888 2970.86490703243 2674.16650629038 2555.09079656132 3723.07778886927
5  741.843912201544 335.694433490563 482.671841077609 300.460219033344 490.210340661021
6   17556.802532596 12279.8736380657 8324.44757171929 6490.01769803064 7772.89649792268
7  3062.92704696704  1263.7235205225 1575.69209873403 1172.09542695646 1012.84874165721
8  6764.62285389594 2427.91454913588 1516.15454675997 3229.84582877413 4158.60768298914
9  1610.79852290994 1541.07059077718  1530.4816003523 980.811468448716 1459.84668675691
10 623.744582070775 181.909289886979 394.474836721317 227.608312832381 91.4725523131246
               NDC5             NDC6             NDC7             NDC8             NDC9
1  1115.69523268786 915.946869512326 827.020837844048 956.348455394368 1318.33978172361
2  970.378250208912 1670.49509640502 792.997427452208 793.728370327222 1603.19863222592
3    553.8099550715 405.497211445671 736.191738711636 682.645922791657  472.86221060931
4  3771.71316342442 2692.55319833414 2142.84829720304 9619.81542081895 2091.04326120572
5  300.863384370603 534.774026818072 325.060797592139 394.839570159419 335.230235179288
6  8072.34334004832 7183.43750927496 6588.18960163548 9160.45581544033 15577.6974445481
7  1157.50343092261 1224.94433390768 1569.99177529267 1103.75139556953 1323.75548640432
8  3022.78840942773 1539.42071964506 3783.01639008689 2673.98741987484 3603.94257955327
9  1583.30791348507 927.187030253167 1434.18530018986 2160.88424764584   851.0002804503
10  355.28762077986 341.832481585919  432.71107826099 225.641397709878 252.526635120438
              NDC10            NDC11            NDC12            NDC13            NDC14
1  959.257675508435 957.341279192094 787.402181166036 1150.79364648404 1016.42641477274
2  1187.96590541036 706.692205529492 1370.40232149845 1243.07345010214 1170.32554780569
```

```
3  776.192477434883 588.784979655287 953.743512519063 960.221157414342 1382.23548520413
4  2295.75138775831 2122.76795082979 3259.62149960379 3169.24027711727 4314.41839772679
5  451.435669447388 228.634624800981 403.812439673337 59.5569520858149 351.956469660082
6  7308.47205936607 19207.4234983569 15458.5167054435 15203.3848298103 10024.4428206163
7  1191.75424388911 1682.11100112997 1236.53931702241 1161.09914105544  1990.3685201984
8  2432.19009327438 2437.12343325146 1688.64401310563 2644.49259951045 2123.53333017901
9  1334.67761578034 1272.75960322345 1934.36783119773 2100.07227236988 2730.55406576972
10 416.616616174824 324.540000314341 373.809158031369 308.673263943977 380.498275944134
               NDC15            NDC16            NDC17            NDC18            NDC19
1  1056.06522218764 1237.54153082996 1337.59901762048 1100.11555153139 930.419133911021
2   2258.8858930509 860.187885373515  762.68537728479  1058.6093541547 1141.24350308935
3  437.018638219734 749.466727339497 913.993986001539 498.887355504144 616.722582782358
4  2438.05365611175 2261.13989048685 2741.75082743599 2317.18062320151 2982.57212044726
5  327.815145969138  84.979246315028 371.670837931869 298.828079582219 338.311320052558
6  10444.2095879426 10270.8473128381 21686.5427957231 6132.15385113103 7411.09249626089
7  1224.14926990847 1392.46674319834 1309.60812159325  3027.3087940244 1267.72844131155
8  3148.43464428128  2956.7093947542 3449.42656455086 4854.73440141574 2536.12652668801
9  761.872603948816 1308.30420331565 1582.03420922645 773.020960808349 1157.06189145691
10 464.593326883323 431.436983803771 408.176977133056 392.918747989245 419.221011898196
               NDC20
1  991.269096390677
2  1329.21537957385
3  1010.03153620702
4  3606.42662869172
5  264.312700186009
6  7768.29355917887
7  1068.26009930721
8  1525.83910786186
9  2501.30845021578
10 339.568707526541
```

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.