Protein Microarray Data Analysis using the *PAA* Package

Michael Turewicz

October 13, 2014

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=""))
```

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).

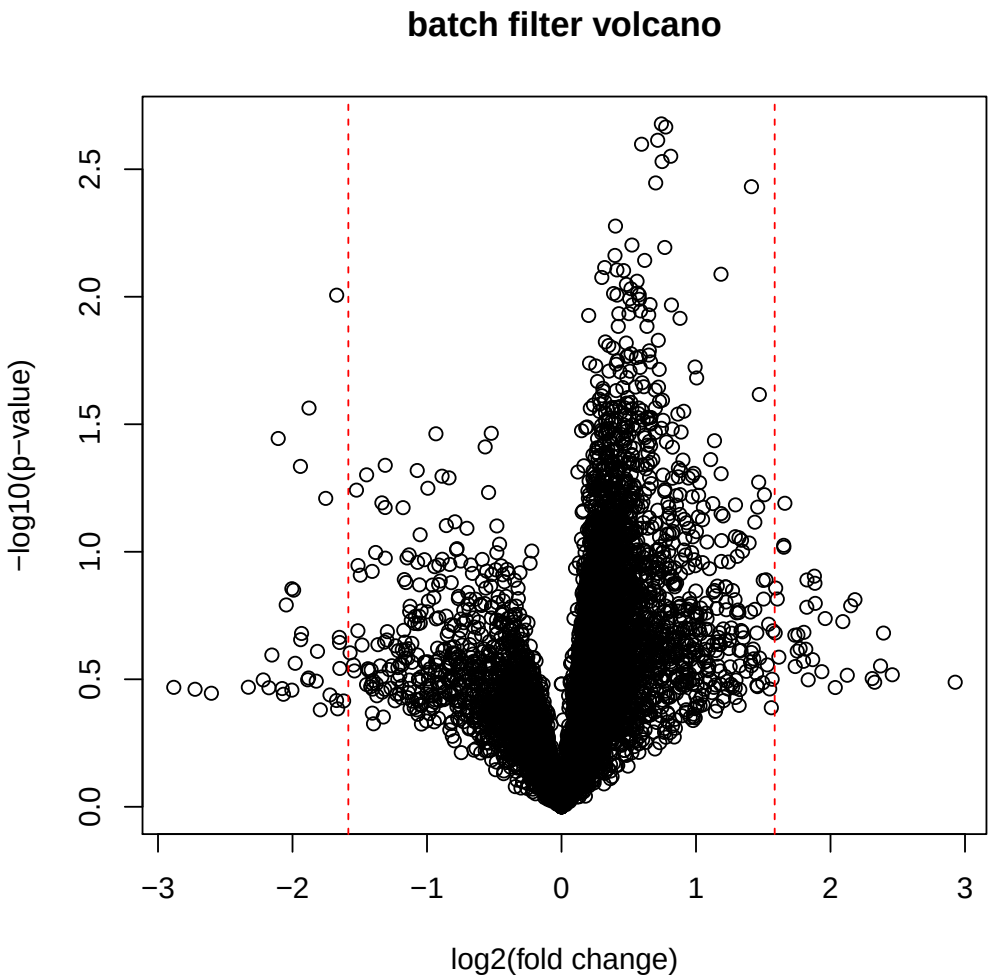
In the following code chunk, the *PAA* installation path (where exemplary data is located) is localized, the new folder 'demo_output' (where all output of the following analysis will be saved) is created, and the exemplary data set is loaded (note: exceptionally not via `loadGPR()`).

```
> 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:

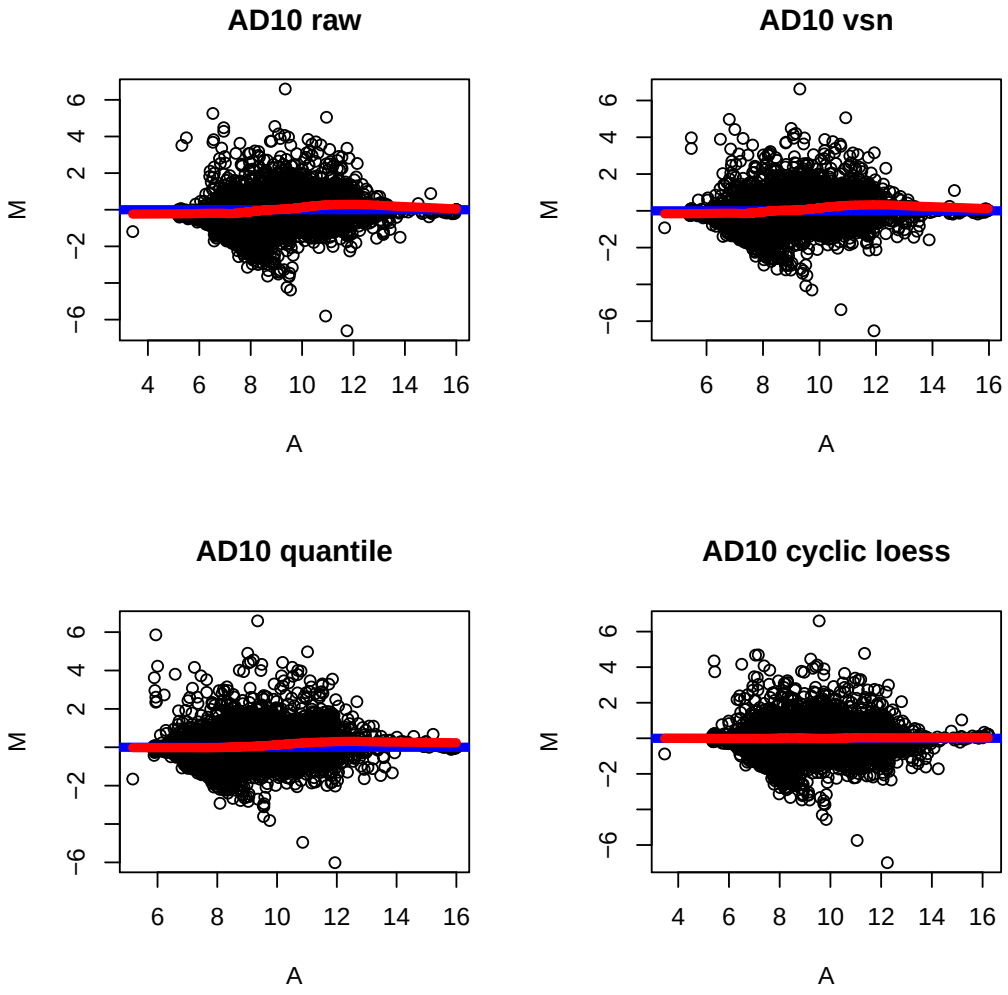
```
> library(limma)
> elist <- backgroundCorrect(elist, method="normexp",
+  normexp.method="saddle")
```

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 `"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
Found 1  categorical covariate(s)
Standardizing Data across genes
Fitting L/S model and finding priors
Finding parametric adjustments
Adjusting the Data
```

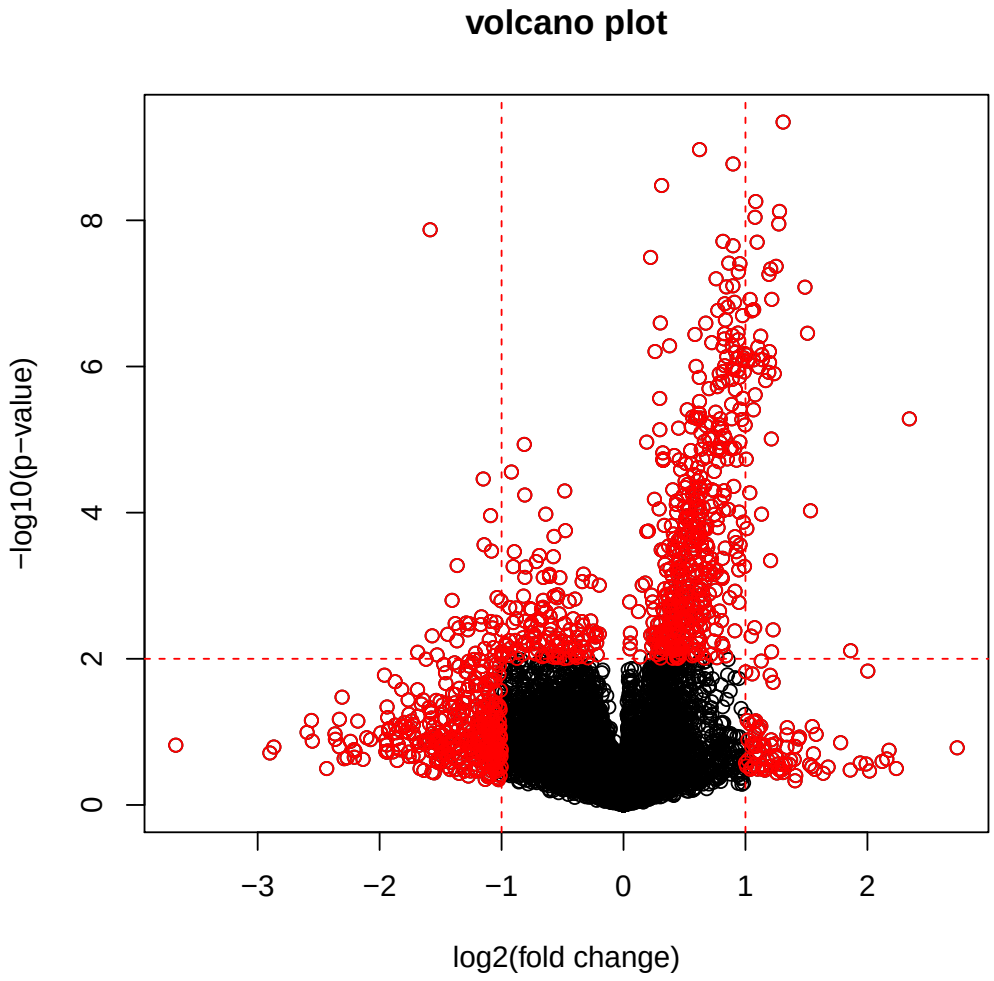
Since for further analysis also data in original scale will be needed, a copy of the *EList* object containing unlogged data should be created.

```
> 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)
```

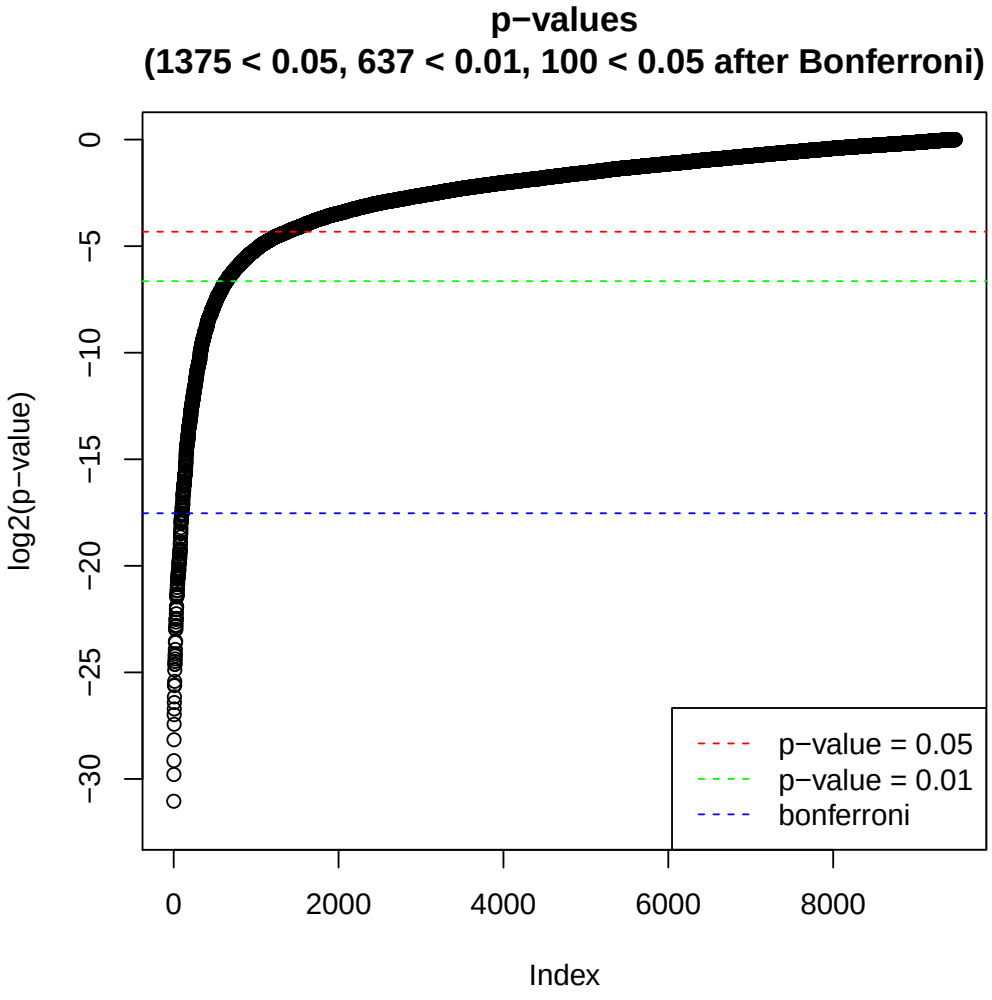


Here, an example with `method="mMs"` is given:

```
> 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")
```

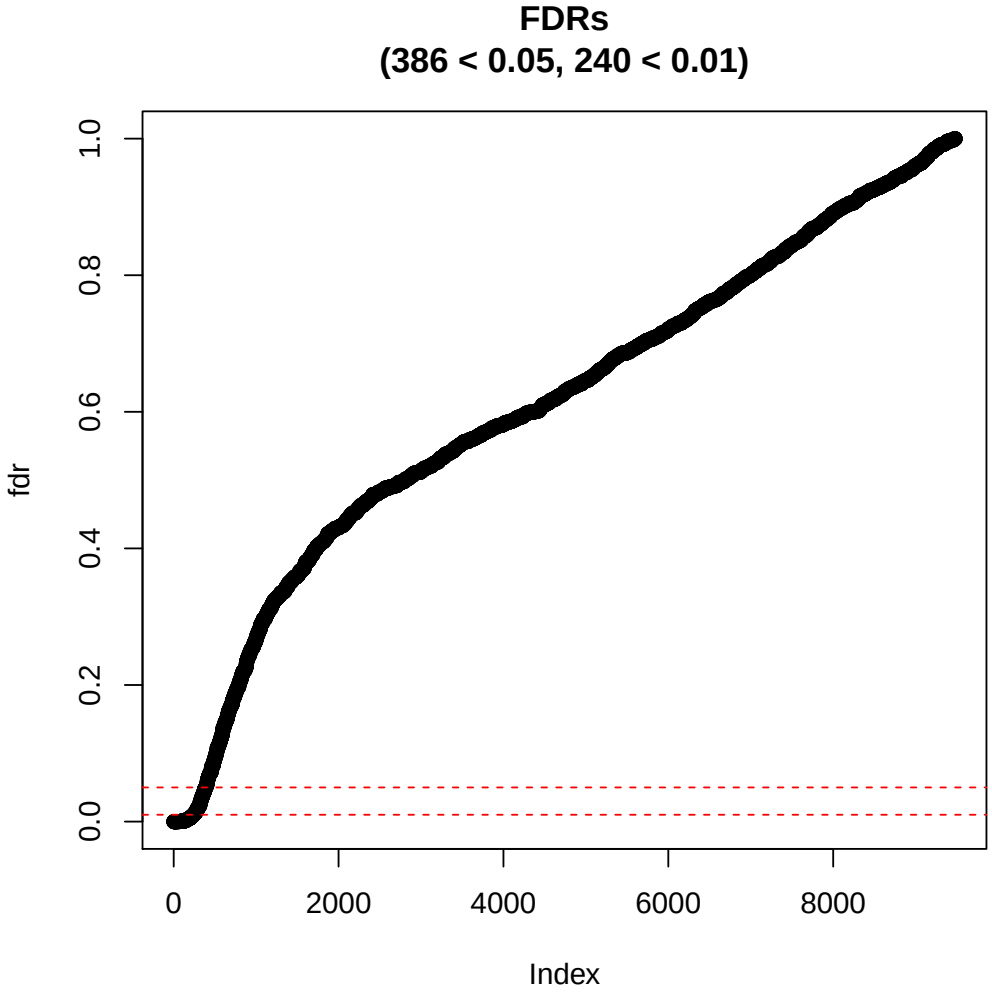


Here, an example with `method="mMs"` is given:

```
> 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   0.351983694209677 0.653973479313352  0.243589743589744 0.830359859486074
2  1 2 13   0.151259072141879 0.503022336421492 0.0241860325286354 0.330856548876571
3  1 2 15   0.319069313374179  0.63259412856928                  1                 1
4  1 2 17   0.178148370508752 0.526723457826063  0.150422391245528 0.830359859486074
5  1 2 19   0.271037416339296 0.598295359038278  0.243589743589744 0.830359859486074
6  1 2 21   0.069361853035849 0.391110753434364 0.0457380457380457 0.483713149738174
7   1 3 1  0.0282571557303619 0.267877836323831                  1                 1
8   1 3 3 0.00910966767019882  0.14042219433087                0.5 0.908394020697585
9   1 3 5 0.00601881506586275 0.107860806851378  0.053014553014553 0.483713149738174
10  1 3 7   0.805098309573945 0.916151278840529  0.302494802494803 0.908394020697585
         fold.change          mean.AD         mean.NDC        median.AD       median.NDC
1   1.36310218942116 1387.24857612588 1017.71428942901 842.099704479647 859.416057239667
2  0.260164203455032 2189.81022237882 8417.03121835251 1306.14075983065 2551.86979248348
3   1.10246479498723 451.984655028113 409.976497284295 415.049207136642 418.503234905461
4  0.595242176244782 1520.86202090465 2555.03067759637 1215.58374522395 1690.44497083082
5  0.453628378851133 2531.33318363501 5580.19141140576 1827.95965127254 1867.42479547798
6  0.757716282697657 2637.13557152232 3480.37336895211 2249.79121136307 2928.81612007343
7   1.26296277410803 486.300802717053 385.047613980944  447.78689939818 350.215519118426
8   1.47980349935485 693.646150408692 468.742066572421 557.911640592165 456.690818828797
9   1.35894544224916 1993.46155077708 1466.91801510276  1874.0807319835 1440.05368538955
10 0.907896179874401 820.138301874463 903.339302504744 731.867870049993 470.674837625346
              sd.AD           sd.NDC
1  1646.15876708672 564.287945192393
2  2967.05315916364 18425.3711275158
3  165.941695749469 82.1672930729294
4  1062.94977081863 3156.77911820629
5  2444.53642628691 11805.2557183832
6  1276.80414608875 1559.53973583639
7   155.25816836103 123.083850273056
8  338.859052471425 93.5614899941108
9  718.813809075484 323.921664377623
10 432.900862308949 1425.95241316282
```

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")
```

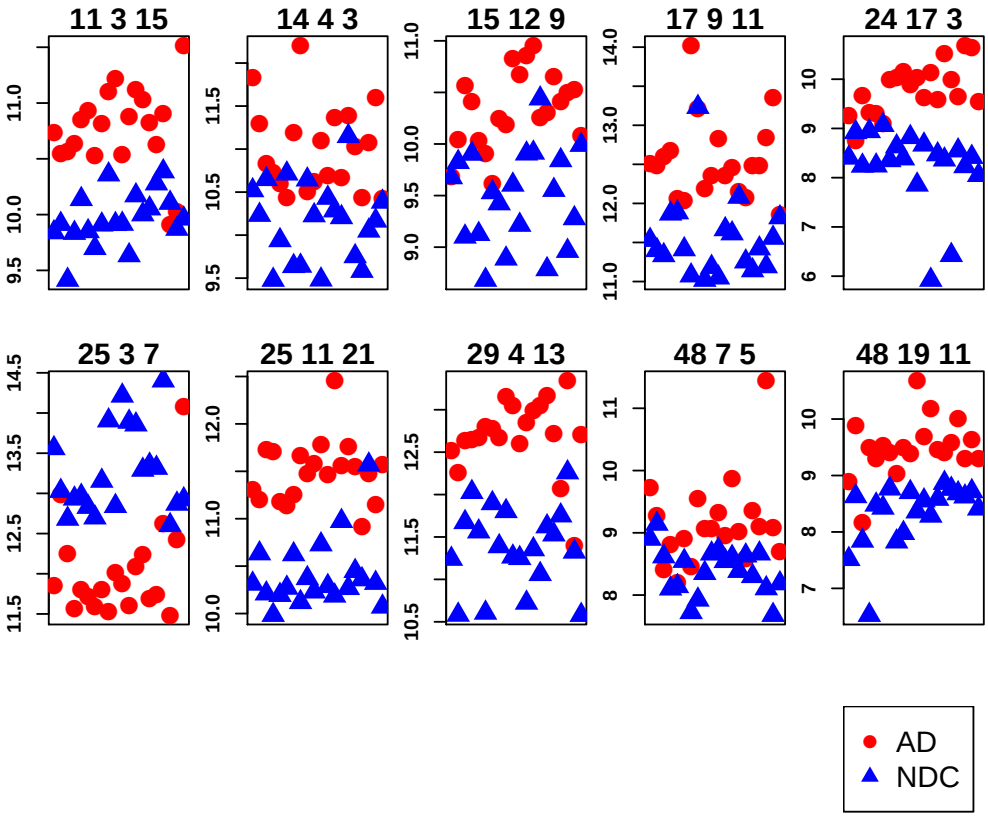
Because runtimes would take too long for this vignette *PAA* comes with pre-computated `selectFeatures.results` objects stored in '.RData' files. These objects can be loaded as follows:

```
> # 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

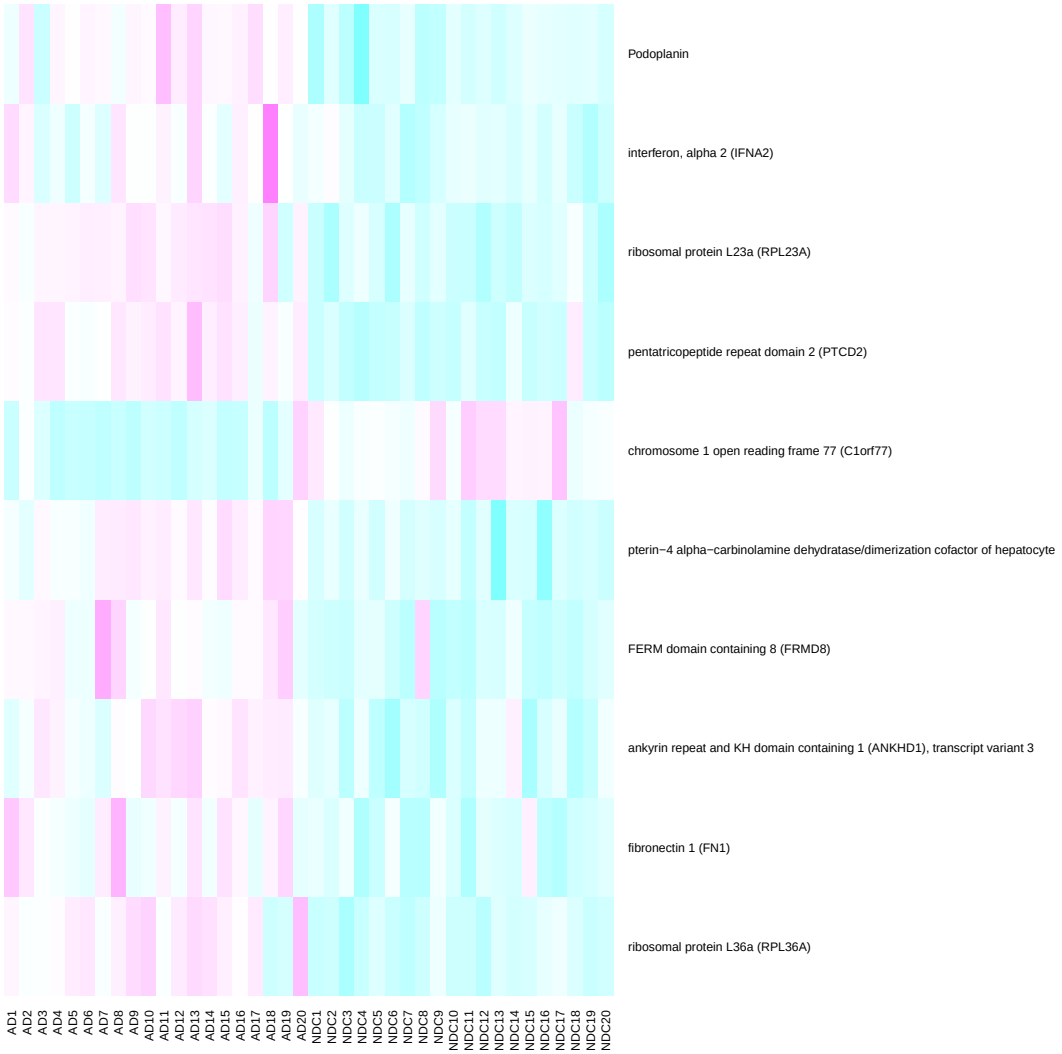
After the selection of a feature panel, these features should be validated by manual inspection and evaluation for further research. To aid results inspection, *PAA* provides several functions. The function `plotFeatures()` plots the intensities of all features (represented by BRC-IDs) that have been 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  1707.5487217474 1497.38674689785 1518.50548665975 1595.48520781193
2     14 4 3  3647.8856681814 2525.02827878685 1822.99945555959  1693.2902270162
3    15 12 9 821.899619465933 1053.14353763606 1517.30660391222 1358.10268235525
4    17 9 11  5841.3140798993 5741.40278219469 6210.83563636426 6537.74627659344
5    24 17 3 616.111573789624  430.50778352629  810.54024033577 640.802478533216
6     25 3 7 3684.06124930958 8097.25857845367 4873.09737651854 3029.75212058229
7   25 11 21 2540.94438380918 2354.59829803536 3390.11030199312 3353.44247657033
8    29 4 13 5874.42547230922 4891.00677037418 6347.91201462816 6413.66758809429
9     48 7 5 845.714240769055  620.14440193628 340.749562634244 449.474219299952
10 48 19 11 476.178151266892 943.905600292745 288.804357634443 720.297395448951
                AD5              AD6              AD7              AD8              AD9
1  1851.16995801813 1954.34250591408 1474.58808187167 1800.72128180474 2203.41666536497
2  1545.93055668746 1382.00035799687 2342.34218157283 4715.17557251859 1453.15299797788
3  1049.37102695366 959.981281957999 784.029298909074 1211.40462403153 1168.05152698585
```

```
4  4260.17714734067 4202.60187600593 16612.0265171095 9541.51053863053  4658.8588632926
5  632.069400444672 548.763100507282 1023.42880381063 1052.79856077289 1141.40761228461
6   3567.8782212016 3361.12781744635 3083.52734553156 3582.11181827476  2964.7945481156
7  2318.92221694186 2253.41013303313 2440.77082029899 3254.85907322306 2857.61042907193
8  6536.80850077055 7127.74897797586 7039.47858075845 6525.73661581423 9104.56738330518
9   293.28859081869 478.504192919484 353.200964938405 750.950212843063 534.827154135703
10 628.686647544015 736.022640348691 677.197314077335 523.364514901869 723.686113405843
               AD10             AD11             AD12             AD13             AD14
1  2386.46403448151 1487.73326737895 1883.45531898606 2229.51382781521 2089.56380522875
2  1576.31726776973 2192.54630997845  1657.0714848759  2643.2065805943 1622.60970111715
3  1817.67145801657 1632.14697480895 1852.35247893044 1983.67962329566 1225.49775172497
4  5223.88824469712 7296.75112793968 5235.96698390644 5625.42315543494 4562.07038409095
5  948.492518532499 1046.88437502484 793.938411158195 1128.04497935662 769.658352738974
6  4130.59006500495  3768.6290095135 3117.05276658092 4362.44872185692 4826.64287027535
7  3076.07852248745 3518.44620770821 2828.55863681375 5617.81938427695 3021.46387470043
8  8470.83889644311   6197.375428568 7401.90131165549 8122.56344668605 8489.87868607207
9  534.374903350561 640.036360803523 494.014987949269 936.872390455245 521.727313868953
10 668.009686426311 1640.47500006756  825.6169869154 1170.89751921492 702.412305275103
               AD15             AD16             AD17             AD18             AD19
1  1816.81480837383 1581.63855404133 1913.63049498382 964.097897606952 1041.31869662808
2  2680.75754761939 2088.89985475576 1386.72270539493 2163.21598235924 3100.93907875795
3  1264.90132812999  1606.128035886 1363.54480704356 1451.21232699924  1472.8558509718
4  4313.49722704237  5726.1793272771 5729.03626045886 7322.01008188934 10463.4326694858
5  1461.33398384387  1014.5357715599 796.654471494386 1643.68188498961 1600.00902244818
6   3305.5529166168 3411.90331260232 6299.56837933074 2867.60878558206 5513.99669518703
7  3482.21452076536 3007.38413310608 1935.38202306179  2846.6780325369 2274.68712494841
8  9182.19304833079 6731.10498346019  4314.2359530566 10389.9308274544  2697.2862285214
9  385.232741137462 654.526774344986 547.249319019918 2785.35671953292 543.590666369519
10    679.003582388 763.190819480539 1035.66536310848 628.178037013272 799.090556916149
               AD20             NDC1             NDC2             NDC3             NDC4
1  2926.42411753136 916.310255562927 965.297890850329 682.054562088009 912.950704810332
2  1377.25850041097 1467.91207671002 1203.85012261539 1605.84699431213  713.29858767954
3  1084.48003919482 817.847716743097 906.146982416223 548.946861729732 957.882043011654
4  3724.16200320842 2957.78866872182 2703.03269812683  2583.5743196281 3735.99667897858
5  744.952853092911 338.540631709138  484.80631656514 302.279573332991 491.625200407955
6  17381.9520086752 12078.4380584594 8356.16477086812  6570.9494040785 7820.46972808707
7  3041.02403868932 1271.70744537942 1587.81230232425 1184.55950308557 1012.47570612393
8  6708.44188234738 2418.89204876839  1531.7973860577 3265.91990969854  4176.0404600778
9  414.318514283749  478.04672302892 564.959917710933  393.11844852088 274.765462066987
10 626.652122277308 183.129536672821  396.28721824634 229.444162809794 92.2033511239305
               NDC5             NDC6             NDC7             NDC8             NDC9
1  1126.38247141383 922.659886109754 830.938986253421 961.606341894609 1312.65211553595
2  981.752307876404  1679.4377370426 796.743380621784 798.991549046095 1600.45415383023
3  558.713037627185 410.597277093104 739.751575990382 686.466408438075 473.595891923403
4  3777.33539556956 2722.26177707613 2158.43944120215 9622.39214229588 2075.93149199798
5  303.117065525581 536.157510885349 326.678547811917 396.881767196424 334.843022317661
6  8007.97121254133 7280.36488225517 6646.32014204794 9120.48431398333 15347.6266865229
7  1171.43537550505   1234.760415103 1577.46172386672 1112.00298493665 1325.10396521862
8  3025.96547287941 1557.58885760092 3812.53215134233 2681.25160743599 3572.53949683205
9  282.289985043158 373.757162655114 211.309445291489  242.78647758168 326.622127353132
10 357.247510556263 344.052731115053 435.198927146926 227.203421006667 252.122607579654
              NDC10            NDC11            NDC12            NDC13            NDC14
1  964.547283576047  965.92545559619  794.50575819914 1153.04044563743 1024.41205626165
2  1199.79776944195  714.82808736054 1381.10817346644 1247.85077230902 1181.44529164993
```

```
3  778.128372427853 594.239115574468 959.255211480267 964.787743665097  1388.6637961329
4  2326.72562790667 2119.09515194113 3252.01291938126 3134.98527115832 4336.07141691114
5  452.275401873642 231.547622778675 406.947836547416 60.6294493022822 354.768459059868
6  7335.85579632123 19002.2930164707 15171.4608698072 14820.5186810997 10027.6534437621
7  1201.24683215565 1695.59724444137 1250.25039970223  1165.9062640145 2011.78693289675
8  2463.80654651362 2434.45300087446 1689.94804150121 2615.46312616187   2135.879656365
9  405.535678033691 423.906146123585 373.155329687147 390.503241154251 334.165739139467
10 416.622316198235 328.548938140955 376.120270496846 311.854059551946 383.569582362546
              NDC15            NDC16            NDC17            NDC18            NDC19
1  1062.74220857401 1241.54257236708 1338.10713313903 1100.43290991954 936.367518922342
2  2271.31764943595 862.867035440605 766.574994407207 1060.18779683922 1149.13869569307
3  440.992612465661 751.178591098318 917.357004718083 498.687370891933 621.105094349124
4   2443.1633378586 2258.87688670426 2743.14047789468  2340.8324014454  3014.3285273019
5  329.753797565583 85.9622199864277 374.766024317663 300.196996312634 341.950351299524
6  10473.5885640039 10195.7433739508 21655.7767725156 6220.93753625123 7480.52915420366
7  1232.22526054351 1395.71156412758 1313.52593772083 3038.53221162454 1279.57569704464
8  3154.40516192712 2952.69278321963 3448.73707209167 4903.95735927502 2564.88063392489
9   397.00171296248 315.579611644105 405.872950164305 275.519666925887 204.822065149198
10 466.530753677265  433.52538788943 410.710097186036 394.782278488936 421.419441238472
              NDC20
1  997.772012656477
2  1338.41875267022
3  1016.73416943206
4  3615.51895537296
5  265.147174966629
6  7771.28134001059
7  1075.78204186532
8  1529.55534039403
9  291.125728322958
10 340.338492661371
```

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.