Protein Mass Spectra (SELDI) Data Processing and Classification with “caMassClass” library

By
Jarek Tuszynski (SAIC)

License of caMassClass

The caMassClass Software License, Version 1.0

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer of Article 3, below. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

2. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

   "This product includes software developed by the SAIC and the National Cancer Institute."

3. If no such end-user documentation is to be included, this acknowledgment shall appear in the software itself, wherever such third-party acknowledgments normally appear.

4. The names "The National Cancer Institute", "NCI" and "SAIC" must not be used to endorse or promote products derived from this software.

5. This license does not authorize the incorporation of this software into any third party proprietary programs. This license does not authorize the recipient to use any trademarks owned by either NCI or SAIC-Frederick.

6. THIS SOFTWARE IS PROVIDED "AS IS," AND ANY EXPRESSED OR IMPLIED WARRANTIES, (INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE) ARE DISCLAIMED. IN NO EVENT SHALL THE NATIONAL CANCER INSTITUTE, SAIC, OR THEIR AFFILIATES BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1 Discussion of Protein Mass Spectra (SELDI) Data Processing and Classification

1.1 Introduction

The purpose of this task is to build a tool that applies classification algorithms to proteomics data (especially SELDI data). Main intended use of those algorithms is distinguishing cancerous samples from normal samples; however they can be used for other classification problems as well.

Developed tools will be used to extend caWorkbench to allow researchers to perform standard classification operations on protein (SELDI) data collected and stored as a part of caARRAY. Most of the classification methods described in this document can be easily applied for other types of data.

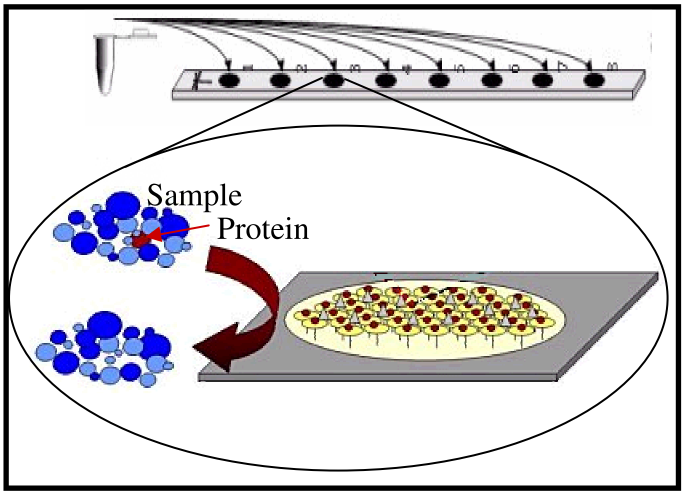


Figure 1: SELDI chip preparation. Drawing adapted from East Virginia Medical School – Virginia Prostate Center[10] website.

1.2 Background

SELDI is a relatively new process which adapted existing mass spectrometry methodology to protein study. An excellent overview this process can be found at [10][21]. The following summary of the process mostly bases on their account.

Chip Array surface is selected from variety of chips specialized in capturing different types of protein samples. Available types are:

- Hydrophobic for reversed-phase chromatography
- Normal phase chromatography
- Anion or cation exchange surface
- Metal binding (IMAC)
- Etc.

Chip goes through several preparation steps performed either manually or by robotic workstation. Sample is applied onto chip array, where targeted subset of protein binds to the chip surface, while the rest of the sample is washed away. Afterwards, energy absorbing molecule (EAM) is added in order to ionize the proteins and the chip with the sample is

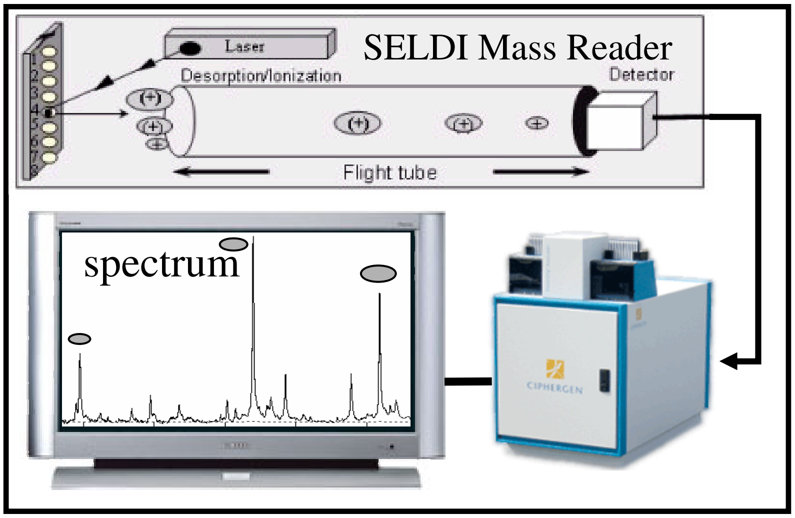


Figure 2: SELDI sample processing. Parts of drawing adapted from EVMS[10] website.

send to “Surface –Enhanced Laser Desorption / Ionization” (SELDI) mass reader. There laser is used to free ionize proteins and allow them to be pulled by magnetic field through vacuum “time-of-flight” tube, where they are separated based on their mass to charge ratio. On the other end of the flight tube their arrival time is recorded by a detector in form of a spectrum. The rest of this paper is concerned with studying those SELDI spectra. This process was first commercialized by Ciphergen Inc. However at present other manufacturers produce competing products.

1.3 Methods

Before SELDI or other protein data can be classified it has to go through several steps of what I will call pre-processing.

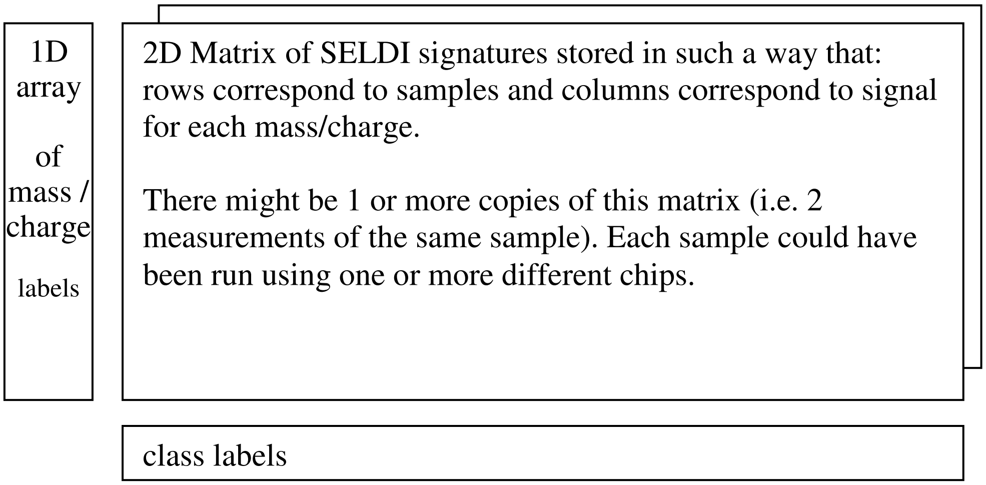
Many different researchers used very different methods in order to process and classify SELDI data. However generalized approach is as follows:

I. Data Input
II. Pre-processing
   1. Baseline subtraction
   2. Normalization
   3. Mass-drift Adjustment
   4. Peak finding and alignment (finding biomarkers, feature extraction)
   5. Merging of multiple sample copies
III. Classification
   1. Feature Selection
   2. Building Classifier

Many of the above steps are optional and were skipped by some or most of the researchers.

Data Input

In case of SELDI data exported from Ciphergen software, data at different stages of processing comes in different format.



2D Matrix of SELDI signatures stored in such a way that: rows correspond to samples and columns correspond to signal for each mass/charge.

There might be 1 or more copies of this matrix (i.e. 2 measurements of the same sample). Each sample could have been run using one or more different chips.

Figure 1: Conceptual view of most common format of SELDI data

1. Raw spectra, baseline subtracted and/or normalized spectra come in the form of separate Excel CSV ("comma separated values") file for each spectrum. Each file contains mass (M/Z) column and intensity column.
2. One could also export data after peak finding step. Those files are also in the CSV format. They store one row per peak, and can contain many different columns describing different aspects of each peak. Among them one should find: spectrum name and/or number, intensity (peak height), "Substance.Mass" (peak mass / position).
3. After peak alignment all data can be exported in a single CSV file that contains one row per spectrum with each column representing different cluster of peaks (biomarker, feature).

Other non vendor-specific formats will likely be developed and used in the future, but for time being those three seem to be most common.

Pre-Processing

Ciphergen SELDI machine comes with its own software which is able to perform some or most of the preprocessing steps. It is up to a user to decide at which point to export their data from Ciphergen environment and start processing it by themselves. However in case of data available on the web, which accompany many papers, researchers that want to reproduce published results have no choice of data format.

If input data is not base-line subtracted this step should be always performed. Base-line is a smooth line that follows local minimum without rising into peaks. Subtracting that line makes "valleys" of the spectrum rest at the zero line. This step is usually done by Ciphergen software, but codes to perform it are also available in PROcess library[2] and Cromwell package[3]. PROcess library bslnoff function divides spectrum into number of unequal sections, finds a minimum (or a quantile corresponding to given probability) of each section, replaces each intensity by that minimum and fits a smooth curve through all the points. Cromwell’s waveletSmoothAndBaselineCorrect function uses wavelets to smooth the spectrum and than uses cumulative (monotone) minimum as a baseline.

The second step of preprocessing is normalizing the multiple spectra. It usually involves cutting first low mass spectra where there is a lot of high frequency high volume noise, which can skew normalization. Afterwards, one finds mean intensity of each spectrum and scales all spectra in such a way as to match all mean intensities. Other ways of normalizing the data also exist for example Petricoin/Liotta study normalized the data by matching minimum and maximum of each signature [4][5][6][7].

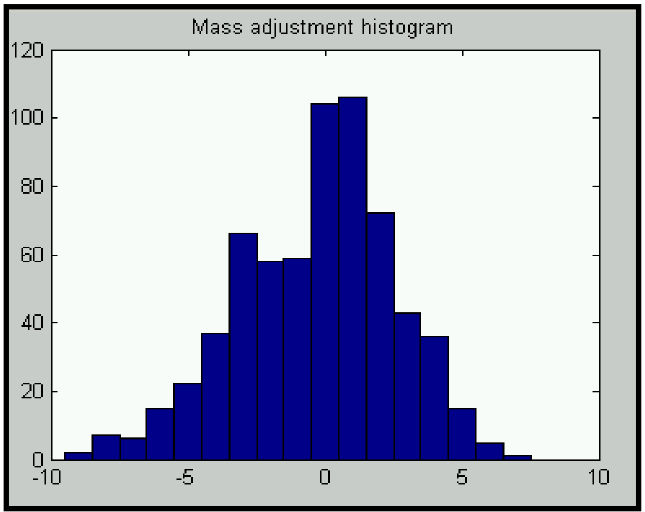


Figure 4: Example of a histogram of shifts to the right (+) or left (-) calculated during mass drift adjustment

At this stage an optional step of mass drift adjustment can be performed. Mass drift adjustment attempts to shift the whole spectrum one or more time steps forward or backward if that is going to improve that spectrum correlation with other samples. This step is especially useful in case of multiple copies of the same sample, which should have very high correlation. The process is done in the following way:

1. First we extract peak regions from all the spectra. That is done by finding a mean spectrum and identifying peak regions as the ones where mean spectrum is above average (in Matlab: peaks = S(:,mean(S,1)>mean(S(:))).
2. Then we create procedure for matching 2 spectra. First spectrum is not moved and the second is shifted one time step to the right or to the left as long as the correlation between two spectra improves
3. Finally we use the above procedure: first to match all spectra to their copies (if present) and than to match each spectrum to mean spectrum. Since mean spectrum will be changing due to those shifts, the procedure will probably have to be done two or three times before stabilizing. Most of the shifts are assumed to be a few time steps.

I did not found any references to other teams using mass drift adjustment, but it does seem to improve quality of the data.

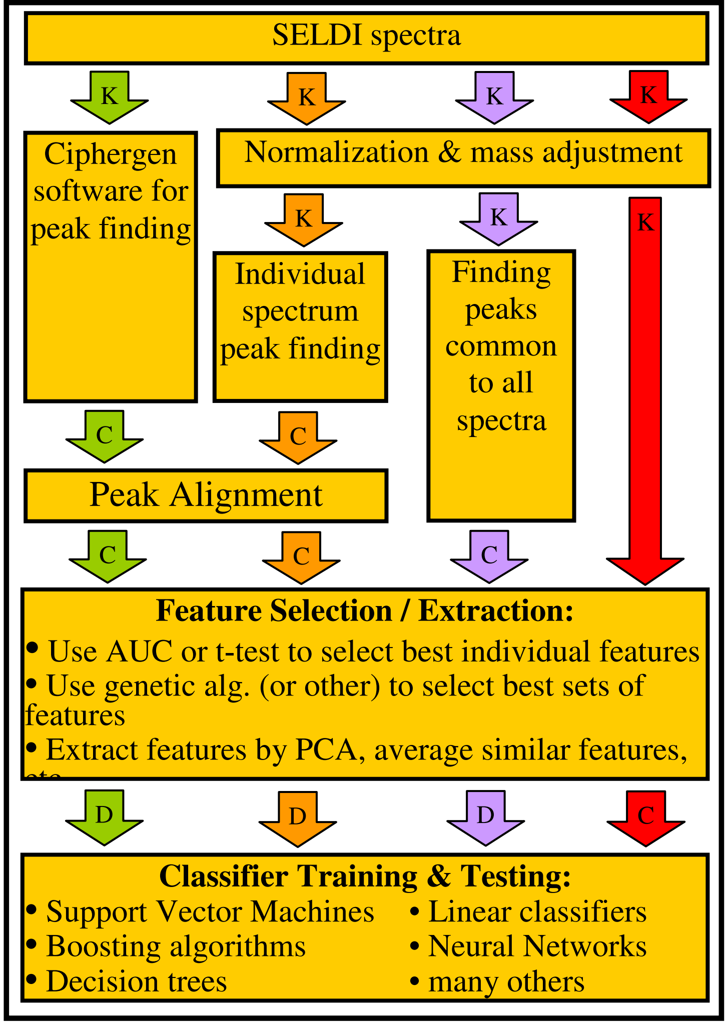

SELDI spectra
K
K
K
K
Ciphergen software for peak finding
Normalization & mass adjustment
K
K
K
Individual spectrum peak finding
Finding peaks common to all spectra
C
C
Peak Alignment
C
C
C
Feature Selection / Extraction:
• Use AUC or t-test to select best individual features
• Use genetic alg. (or other) to select best sets of features
• Extract features by PCA, average similar features, etc.
D
D
D
C
Classifier Training & Testing:
• Support Vector Machines
• Linear classifiers
• Boosting algorithms
• Neural Networks
• Decision trees
• many others

Figure 5: Algorithm families for classification of SELDI signatures show different approaches used by different teams: green: EVMS [8] & CPDR [9], brown: Baggerly [6], purple: an algorithm which combines peak finding and peak alignment, red - Petticoin/Liotta **[4].**

The next step would be peak finding and alignment in order to find biomarkers. However it seems like at this stage there are two major different approaches related to SELDI data classification: some research teams use peak alignment to reduce size of the data before classification [8][9][6] and other teams apply classification techniques to the raw data [4] . If the first approach is used than the steps are as follows:

1. in each spectrum separately peaks are detected using variety of different methods [1][2][3][6]
2. peaks from different spectra are aligned into single matrix [11][8][2][3] (see figure 4)

If the second approach is taken than we skip the above 2 steps and use feature selection approach to lower dimensionality of the data. This step however requires use of class labels and by definition is not a part of pre-processing.

The final pre-processing step is to merge multiple copies of each sample that could have been provided into single uniform set of features associated with each sample. If only a single copy is collected of each sample than this step should be skipped. There are two types of sample copies:

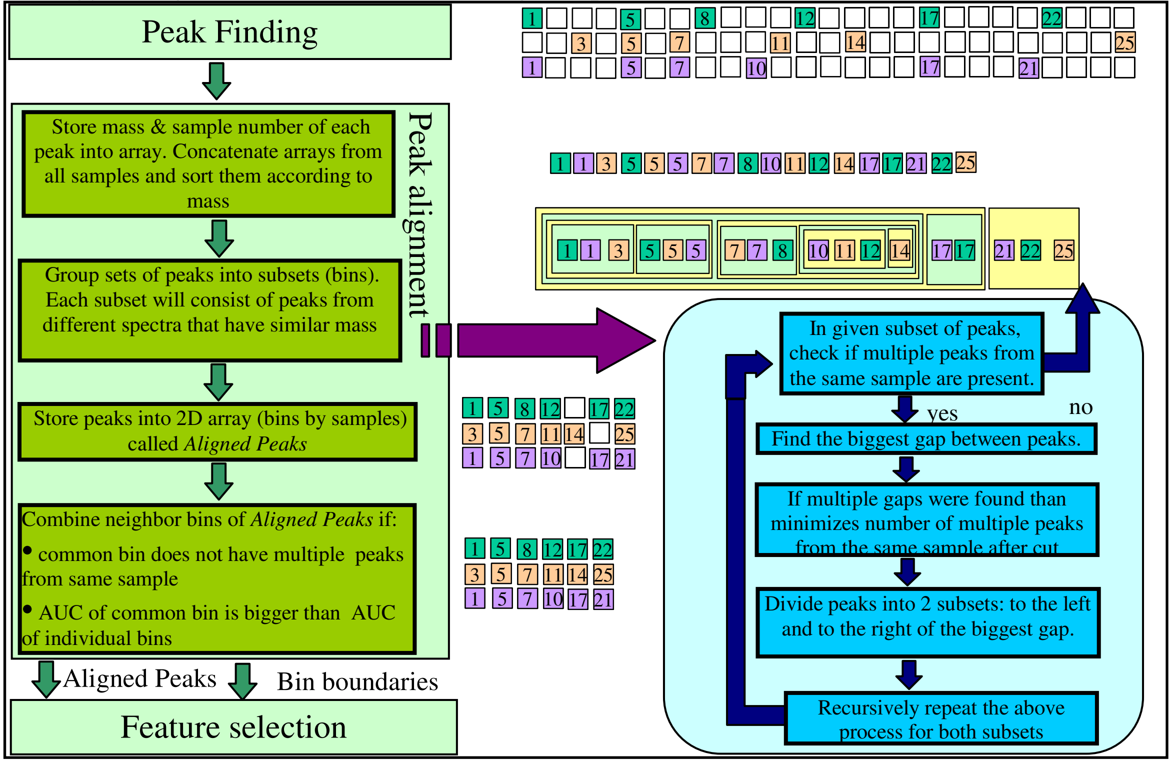
1. Equivalent copies that were taken under the same conditions and should be identical [8]
2. Copies are taken under different conditions (different chips, different hardware settings, etc.) are assumed to be different. [9]

In order to merge the equivalent copies one can:

1. Average them in order to get a signature with much better signal to noise ratio. That is especially true for the test set.
2. If more than 2 copies are collected than one can average only two (or more) copies that are most similar to each other and discard the outliers.
3. Even with 2 copies one can average the copies if they are highly correlated to each other and discard one if they are not. The discarded copy should be the one that resembles the least other samples.

4. In case of the train set one can also keep all the copies and treat them as separate training samples. That choice gives smaller sample purity, but creates larger train set.
5. Another possibility is to keep all the samples and their averages for the even larger number of train samples.

In case of copies taken under different conditions there seem to be only one merging strategy: merge them end-to-end creating samples with twice the number of features.



Peak Finding
Peak alignment
Store mass & sample number of each peak into array. Concatenate arrays from all samples and sort them according to mass
Group sets of peaks into subsets (bins). Each subset will consist of peaks from different spectra that have similar mass
Store peaks into 2D array (bins by samples) called *Aligned Peaks*
Combine neighbor bins of *Aligned Peaks* if:
• common bin does not have multiple peaks from same sample
• AUC of common bin is bigger than AUC of individual bins
Aligned Peaks
Bin boundaries
Feature selection
1 5 8 12 17 22
3 5 7 11 14 25
1 5 7 10 17 21
1 1 3 5 5 5 7 7 8 10 11 12 14 17 17 21 22 25
In given subset of peaks, check if multiple peaks from the same sample are present.
yes
no
Find the biggest gap between peaks.
If multiple gaps were found than minimizes number of multiple peaks from the same sample after cut
Divide peaks into 2 subsets: to the left and to the right of the biggest gap.
Recursively repeat the above process for both subsets

Figure 2: Peak alignment algorithm that follows method from [11].

Building classifiers

There are many classifiers that can be used. So the main purpose of this section is to provide framework to compare them to each other in order to choose the best one. The process is called cross-validation [14] and the general steps are as follow:

- For each classification method:
  - Repeat multiple times (10s – 100s)
    - Split train set of labeled samples into 2 groups: temporary train and test sets
    - Train each classifier on train set and test it on the test set
  - Collect statistics on each classification method: mean and variance of accuracy, or sensitivity/specificity
- Choose classification method with the best performance
- Apply this method to the full set of labeled samples

In this framework the step of training each classifier could be preceded or combined with feature selection, for example:

1. In order of dropping dimensionality of the data one can use t-test or Wilcoxon-test (equivalent to area under ROC curve) to rank each feature according to its individual strength of separating the data into two or more classes. Features with rank below certain threshold could be eliminated. [7][8]
2. Another approach is to look for very similar neighbor features (with high correlation between them) and keep only one of them – the one with higher separation capabilities, as measured by Wilcoxon-test. That approach is especially useful if no peak finding is used during preprocessing.
3. Feature selection can be also performed with goal of finding a good set of features instead of sets of good features. That is more time consuming approach but with potential high rewards. For example exhaustive search [6][7], genetic programming [4], or other methods, can be used to find the best set of features according to some criteria (statistical distance, accuracy of classifiers that could be build using those features (see figure 2), etc.)

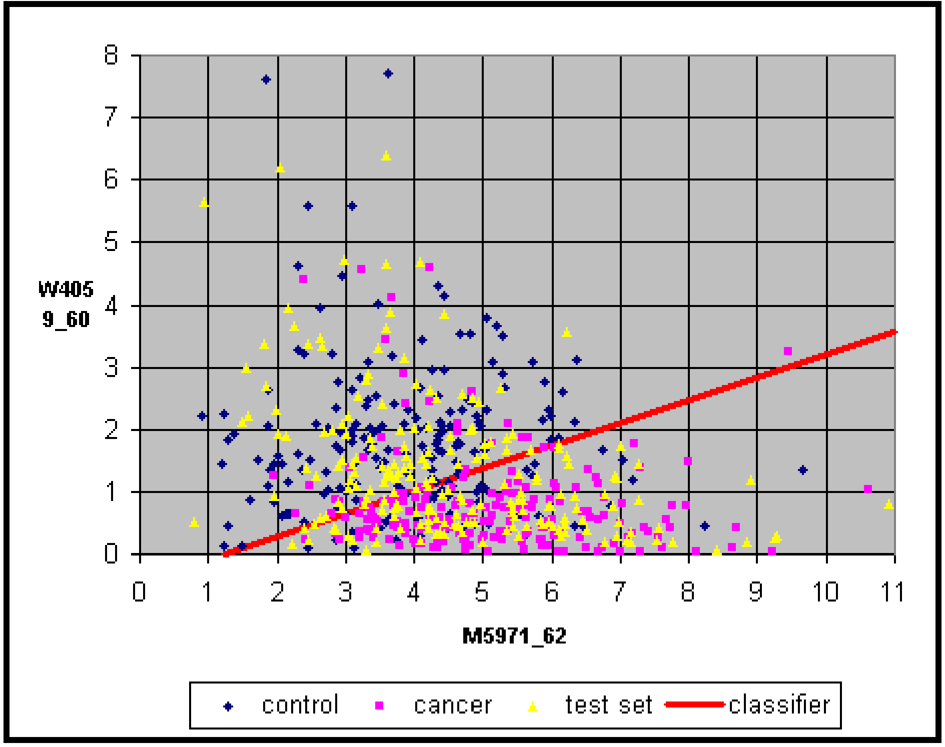


Figure 3: Example of feature selection. Among 95 features 2 were found that allow separation of 2 clusters with average accuracy of 78%, in this case test set was also predicted with 79% accuracy.

The classifiers particularly useful for working on SELDI classification problem have to be able to work with data sets that have usually much more features than samples. I have the best results with:

1. Fisher linear classifier when combined with feature selection [6][7]
2. Support vector machine classifiers

3. Neural network classifiers (we got the best results with a feed-forward neural network classifier with back-propagation)
4. Decision tree classifiers[8][9]
5. Boosting methods based on decision stump classifiers (AdaBoost[13], LogitBoost[12])

Pettricoin/Liotta team [4][21] also reported good results with self organizing maps (SOM) approach.

Table 1: Error rates using different approaches on different data sets. Some classification is between cancer and normal samples, other is between different stages of cancer.

	Peak finding by Ciphergen software & alignment	**Individual peak finding & alignment**	**Common peak finding**	**Selection of smoothed features without peak finding**	**Feature Selection without peak finding**
CPDR-1 Cancer/Normal	23±7% -my peak finding 17±6% - peak finding by CPDR	~25%		~25%	
CPDR-2 Cancer/Cancer		Very poor results	Very poor results	Very poor results	
CPDR-3 Cancer/Normal			48%	Very poor results	7% (but ~50% in blind test)
CPDR-3a Cancer/Normal	~20% in blind test	33±10%	30±8%		6% (20% in blind test)
CPDR-4 Cancer/Normal	21% in blind Cancer/ normal test				~30% in cross validation of cancer/ cancer
EVMS-VPC Cancer/Benign/Normal	2.5±2 % (Peak finding and alignment done by EVMS)	~25%	19.2±5%	10 feat. & ldc - 18.3±4% 200 feat. & svm - 24±4%	10 feat. & ldc - 19.3±4% 200 feat. & svm - 28±6%
NCI-Prostate Data Cancer/Benign/Normal	No data in Ciphergen format	17 ±5%	13±4%	10 feat. & ldc - 12±4% 200 feat. & svm - 15±4%	9 feat. & ldc - 13±4% 200 feat. & svm - 19±4%
NCI-Ovarian Data Set I (Lancet set)	No data in Ciphergen format	Poor results	Poor results	~20%	0% - Benign/Normal & Cancer

Cancer/Benign/Normal					using Eigen vectors; Normal/Cancer 5 feat. & ldc - 8±4%
NCI-Ovarian Data Set II Cancer/Benign/Normal	No data in Ciphergen format				0.05% error can be achieved with 3 point linear classifier
NCI-Ovarian Data Set III Cancer/Normal	No data in Ciphergen format	Impossible because data was not base-line corrected	Impossible because data was not base-line corrected		0% - using 2 sets of 2 features and linear classifier

1.4 Concerns

In SELDI spectra classification problem there are two major potential problems inherit to the process:

- Low data reproducibility – SELDI instruments are very sensitive to minute changes in machine settings and sample preparation protocol. As a result, two machines or the same machine at different times will likely give the different results for the same samples. That, among other problems, creates a potential for introducing bias into the data samples. For example if cancer samples are processed in a separate batch from normal samples, than there is a potential that the best classifier found to distinguish between them will not relay on biological differences but rather on differences in machine settings. That or similar scenario possibly happen to one of ovarian cancer datasets listed in [5] as suggested by [6][7]. Another example would be CPDR-3 data set where train dataset was collected at different time than test dataset. As a result my best classifiers which had 7% error rate during cross validation procedure had close to 50% error rate in during blind test.
- Possibility of false discovery – is a second potential problem, which is shared with other types of data that have much larger number of features (10 000s) than samples (100s), for example microarrays [22]. Many of the standard classification algorithms are so good at what they are doing, that they can find patterns even in the random data. For example if we use exhaustive search to find the best two-feature linear classifier (see figure 5) using data with N=50 000 features, than theoretically we have $N^2/2 = 10^9$ different data sets to evaluate, so even very unlikely events with probability of 1 in 10^9 should happen once. Also, the smaller number of samples the higher the chance that random data will arrange itself into desired pattern.

1.5 Conclusions and Recommendations

Our main conclusion is that there is no single best approach for classifying SELDI data, but rather several competing algorithms that have to be tried in order to find the optimal one. The choice of the algorithm depends on many factors, some of them listed in the next section. My recommendation would be to implement multiple algorithms for every step of data processing and implement them as a single cohesive R library that would provide a common interface to allow researchers to experiment with different method.

The Methods sections list a multitude of different approaches and algorithms. Some of the factors that could affect choice of the algorithm:

1. Data size vs. time and memory available – some methods are more appropriate for smaller data sets since they take too long to process, also some algorithms take too much memory. My machine has 1.5 GB and it is no uncommon for me to run out of memory on larger data sets.
2. Source of data and stage of processing – when working with data posted on the web by different research teams, one does not have a choice of the level of preprocessing done on the data. For example some data sets will be baseline subtracted and normalized other will be raw, yet another set will contain only extracted biomarkers. So the pre-processing steps to be chosen will have to match the data itself.
3. Number of copies collected – the choice of data merging techniques performed on the end of preprocessing will mostly depend on number and type of copies available.
4. Number of different categories – there is often a difference between two-way and multiple way classification, since some classification algorithms do not always support more than two classes. For example if one performs cancer/non-cancer classification than his choice of classifiers might be different than if one performs prostate cancer/non-cancer/BPH (benign prostatic hyperplasia) classification, and different again if one wants to study distinguishing features between four stages of cancer.
5. Purpose of the study – the choice of the classifier could be different if the final goal of the study is to find the best possible classifier vs. to find the best classifier as a way of identifying a limited set of features with the greatest power of distinguishing between multiple classes. One might want to find distinguishing features, since they have to correspond to different protein which we might want to identify. For example: decision tree, boosting and some feature selection algorithms work by finding limited sub-set of features and operating only on those, while neural networks, SVM, fisher and many other algorithms always use all the features provided. Because of that, the first set of algorithms gives you two results: a classifier and best set of features, while the second gives you only the classifier
6. Availability of different algorithms in any particular language – software and algorithms for SELDI data processing [1][2] [3], mass spectrometry data processing and classification [15-20] are available in R, Matlab, and C (C++) codes. This project is to be written in R language, what means that some libraries are available, while other will have to be rewritten.

The final library would follow the basic course of operation that contains following steps:

1. User inputs Process Parameters, which will uniquely describe the rest of the flow. The parameters are saved into Parameter Store, which will be retrieved by remaining processes.
2. Data is pre-processed according to user specifications retrieved from Parameter Store, and then stored in Pre-processed Data Store
3. Classifiers are built using pre-processed data and class labels. The steps of the process are specified by Parameter Store.
4. Classifier is verified by a User or applied by a Clinical Manager. That is done by running the classifier on unlabeled pre-processed data in order to predict the class labels.

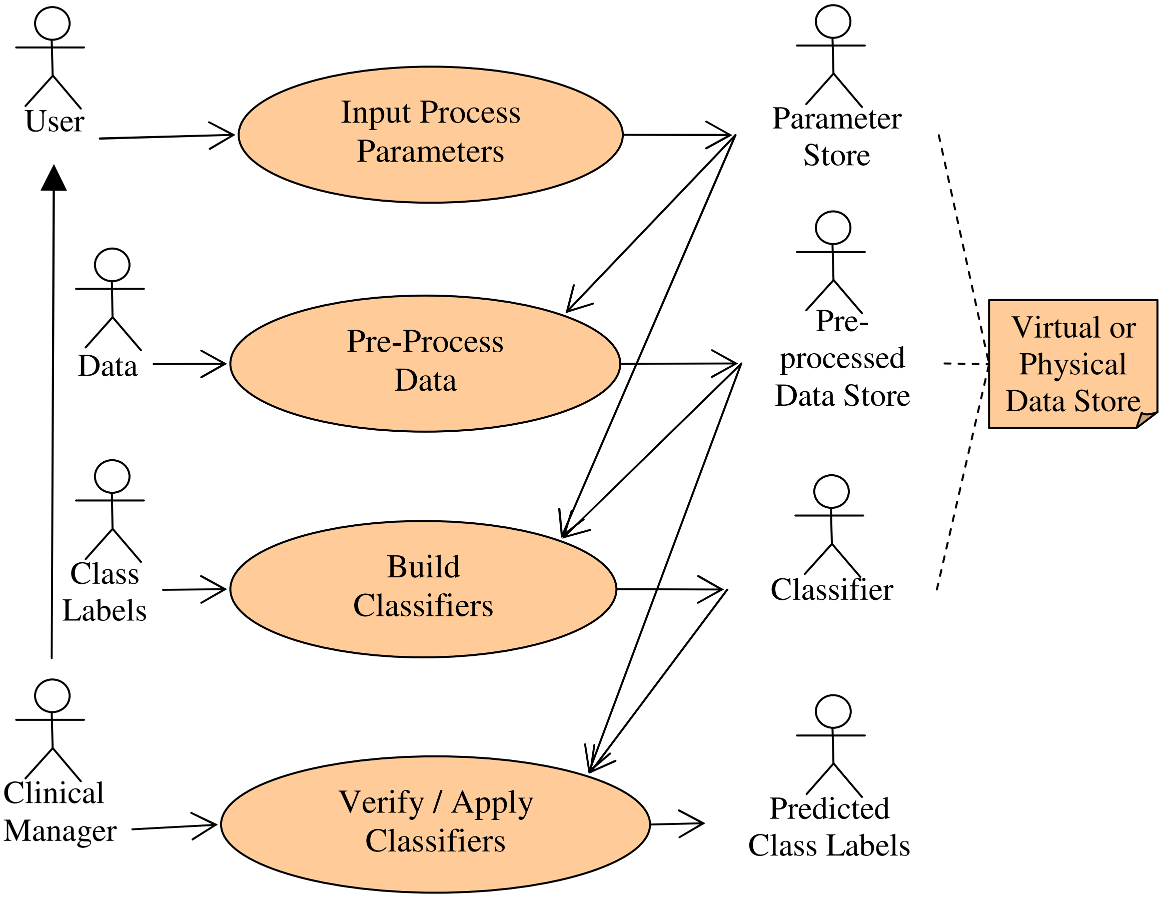


Figure 4: Basic course of *Pattern Recognition for Diagnosis and Treatment* use case

The above steps are common to all of described classification algorithms, and the choice of the actual algorithm will have to be saved in Parameter Store.

2 Functions in the caMassClass Library

2.1 msc.project.run - Read and Preprocess Protein Mass Spectra

Description

Read and preprocess protein mass spectra (SELDI) files where files could contain multiple copies of spectra taken from the same sample, or spectra from multiple experiments performed on the same sample.

Usage

```
msc.project.run(ProjectFile, directory.out=NULL, verbose = TRUE, ...)
```

Arguments

`ProjectFile`	path and name of text file in Excel's CSV format which is used to store information about a batch of Mass Spectra data files. See details.
`directory.out`	Optional character vector with name of directory where output files will be saved. Use "/" slashes in directory name. By default the directory containing `ProjectFile` and all Mass Spectra files is used, and this argument is provided in case that directory is read-only and user has to choose a different directory.
`verbose`	boolean flag turns debugging printouts on.
`...`	parameters to be passed to `msc.preprocess.run`

Details

High level processing of protein mass spectra (SELDI) data. `msc.project.read` supports projects with multiple sets of spectra taken under different experimental condition. Those sets will be called *batches*. With that in mind, following steps are performed:

- `msc.project.read(ProjectFile, directory)` is called which reads and saves different batches of mass spectra (SELDI) data into separate files. List of those files is saved in temporary "RInputFiles.csv" file. In future calls to `msc.project.run`, if above file exist than `msc.project.read` is not called again.
- Each batch of data is `load`ed and preprocessed by calls to `msc.preprocess.run`. All the required parameters have to be passed through "..." mechanism.
- In case of multiple batches of data results are `rbind`ed

Value

`X` Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X)` store M/Z mass of each row merged with batch name

`SampleLabels` Class label for each sample as read from `msc.project.read`

See Also

`msc.project.read`, `msc.preprocess.run`

Examples

```
  directory  = system.file("Test", package = "caMassClass")
  ProjectFile = file.path(directory,"InputFiles.csv")
  Data = msc.project.run(ProjectFile, '.',
      baseline.removal=0, mass.drift.adjustment=1, min.mass=3000,
      peak.extraction=1, merge.copies=7, shiftPar=0.0004)
```

2.2 msc.project.read - Read and Manage a Batch of Protein Mass Spectra

Description

Read and manage a batch of protein mass spectra (SELDI) files where files could contain multiple spectra taken from the same sample, or multiple experiments performed on the same sample.

Usage

```
msc.project.read(ProjectFile, directory.out=NULL)
```

Arguments

`ProjectFile` path and name of text file in Excel's CSV format which is used to store information about a batch of Mass Spectra data files. See details.

`directory.out` Optional character vector with name of directory where output files will be saved. Use "/" slashes in directory name. By default the directory containing `ProjectFile` and all Mass Spectra files is used, and this argument is provided in case that directory is read-only and user have to choose a different directory.

Details

Function `msc.project.read` allows to user to manage large batches of Mass Spectra files, especially when multiple copies of each sample are present. The `ProjectFile` contains all the information about the project. An example format might be:

Name,	**Class**,	**IMAC1**,	**IMAC2**,	**WCX1**,	**WCX2**
r0008,	1,	Nr/imac_r0008.csv,	Nr/imac_r0008(2).csv,	Nr/wcx_r0008.csv,	Nr/wcx_r0008(2).csv
r0012,	1,	Nr/imac_r0012.csv,	Nr/imac_r0012(2).csv,	Nr/wcx_r0012.csv,	Nr/wcx_r0012(2).csv
r0014,	1,	Nr/imac_r0014.csv,	Nr/imac_r0014(2).csv,	Nr/wcx_r0014.csv,	Nr/wcx_r0014(2).csv
r0021,	2,	Ca/imac_r0021.csv,	Ca/imac_r0021(2).csv,	Ca/wcx_r0021.csv,	Ca/wcx_r0021(2).csv
r0022,	2,	Ca/imac_r0022.csv,	Ca/imac_r0022(2).csv,	Ca/wcx_r0022.csv,	Ca/wcx_r0022(2).csv
r0024,	2,	Ca/imac_r0024.csv,	Ca/imac_r0024(2).csv,	Ca/wcx_r0024.csv,	Ca/wcx_r0024(2).csv
r0027,	2,	Ca/imac_r0027.csv,	Ca/imac_r0027(2).csv,	Ca/wcx_r0027.csv,	Ca/wcx_r0027(2).csv

`ProjectFile` always has the following format:

- column 1 - unique name for each sample - Those names will be used in the program to identify the samples
- column 2 - class label for each sample - in the classification part of the code those labels will be used as a response vector (target values). Usually a factor for classification, but could be a unique number for regression.
- columns 3+ - file path (from `directory`) for each file in the project. If `ProjectFile` has more than 3 columns than multiple copies of the same sample are present. In that case column labels (IMAC1, IMAC2, WCX1, WCX2) become important, since they distinguish between equivalent copies taken under the same conditions and copies taken under different conditions. In our example both kinds of copies exist: files in columns IMAC1 and IMAC2 contain two copies of spectra collected using Ciphergen's IMAC ProteinChip array and files in columns WCX1 and WCX2 used WCX array. The labels of those columns are expected to use letters as labels for different copies and numbers to mark multiple identical copies.

File names in `ProjectFile` could be compressed uzing zip and gzip file compression. For example if individual file name is in the format:

- "a.csv" - trivial case - uncompressed file
- "b.zip/a.csv" - file within zipped file
- "a.csv.gz" - gziped individual file

Value

List of .Rdata files storing data that was just read. Each file contains either 2D data (if only one copy of the the data existed) or 3D data (if multiple copies of the data existed). Multiple files are produced if multiple experiments were performed under different conditions. In above example two files will be produced: Data_IMAC.Rdata and Data_WCX.Rdata.

See Also

- `msc.msfiles.read.csv` is a lower level function useful for data that does not have multiple copies,
- `msc.preprocess.run` is usually used to process output of this function.
- `read.files` from **PROcess** library can read a single SELDI file and `rmBaseline` can read in a directory of files and substract their baselines.
- `ppc.read.raw.batch` and `ppc.read.raw.nobatch` from **ppc** library can also read SELDI files, assuming correct directory structure.

Examples

```
directory = system.file("Test", package = "caMassClass")
ProjectFile = file.path(directory,"InputFiles.csv")
FileNames = msc.project.read(ProjectFile, '.')
cat("File ",FileNames," was created\n")
```

2.3 msc.msfiles.read.csv - Read Protein Mass Spectra from CSV files

Description

Read multiple protein mass spectra (SELDI) files, listed in `FileList`, from a given directory and combine them into a single data structure. Files are in CSV format, possibly compresses. If `FileList` is an 1D list than data is stored as a matrix one file per column. If `FileList` is a 2D data-frame than data is stored in 3D array.

Usage

```
msc.msfiles.read.csv(directory=".", FileList="\.csv",
                     SampleNames=NULL, CopyNames=NULL)
```

Arguments

`directory` a character vector with name of directory where all the files can be found. Use "/" slashes in directory name. The default corresponds to the working directory `getwd`().

`FileList` List of files to read. List can be in the following formats:

- single string - a regular expression (see `regex`) to be used in selecting files to read, for example "\.csv"
- list - list of file names to be read
- data.frame (multiple lists of file names)- multiple copies of the same samples are present - see details

The last two formats also support file zip and gzip file compression. For example if individual file name is in the format:

- "dir/a.csv" - uncompressed file 'a.csv' in directory 'dir'
- "dir/b.zip/a.csv" - file 'a.csv' within zipped file 'b.zip'
- "dir/a.csv.gz" - gziped individual file

`SampleNames` Optional list of names to be used as sample/column names.

`CopyNames` Optional list of names to be used as copy/plane names in case FileList is an 2D data frame.

Details

All files should be in Excel's CSV format (table in text format: 1 row per line, comma delaminated columns). Each file is assumed to have two columns, in case of SELDI data: column 1 (x-axis) is mass/charge (M/Z), and column 2 (y-axis) is spectrum intensity. All files are assumed to have identical first (M/Z) column.

If multiple copies of the same sample were collected than one can store them in a 3D array (data cube) where each column correspond to a single sample, each row is a single mass (M/Z) and each plane is a single copy. To do so one has to pass a 2D data frame as `FileList` where each column contains file names of multiple copies of the same sample and each row contains filenames of a single copy of different samples.

Value

Data structure containing all the data read from the files. It can be in form of a 2D matrix (nFeatures *x* nSamples) or 3D array (nFeatures *x* nSamples *x* nCopies) depending on input.

See Also

- Part of `msc.project.run` pipeline.
- `msc.project.read` gives user much more flexibility in defining the meaning of the data to be read.
- `msc.preprocess.run` is often used as a next step in the process
- `read.files` from **PROcess** library can read a single SELDI file and `rmBaseline` can read in a directory of files and substract their baselines.
- `ppc.read.raw.batch` and `ppc.read.raw.nobatch` from **ppc** library can also read SELDI files, assuming correct directory structure.

Examples

```
# example of mode "single string" FileList
directory  = system.file("Test", package = "caMassClass")
X = msc.msfiles.read.csv(directory, "IMAC_normal_.*csv")
dim(X)
```

```
  # example of explicite 1D FileList
  ProjectFile = file.path(directory,"InputFiles.csv")
  FileList = read.csv(file=ProjectFile, comment.char = "")
  FileList[,3]
  X = msc.msfiles.read.csv(directory, FileList=FileList[,3],
SampleNames=FileList[,1])
  dim(X)

  # example of explicite 2D FileList
  FileList[,3:4]
  X = msc.msfiles.read.csv(directory, FileList=FileList[,3:4],
        SampleNames=FileList[,1], CopyNames=c("copy 1", "copy 2"))
  dim(X)
```

2.4 msc.preprocess.run - Preprocessing Pipeline of Protein Mass Spectra

Description

Pipeline for preprocessing protein mass spectra (SELDI) data before classification.

Usage

```
msc.preprocess.run ( X,
    baseline.removal = 0,
      breaks=200, qntl=0, bw=0.005,                      # bslnoff
    min.mass = 3000,                                    # msc.mass.cut
    mass.drift.adjustment = 1,
      shiftPar=0.0005,                                  #
msc.mass.adjust
    peak.extraction = 0,
     PeakFile=0, SNR=2, span=c(81,11), zerothresh=0.9, # msc.peaks.find
     BmrkFile=0, BinSize=c(0.002, 0.008), tol=0.97,     #
msc.peaks.align
     FlBmFile=0, FillType=0.9,                          #
msc.biomarkers.fill
    merge.copies = 4+2+1,                               #
msc.copies.merge
    verbose = TRUE)
```

Arguments

`X` Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X)` store M/Z mass of each row.

`baseline.removal` Remove baseline from each spectrum? (boolean or 0/1 integer). See function `msc.baseline.subtract` and `bslnoff` from **PROcess** library for other parameters that can be passed: **`breaks`**, **`qntl`** and **`bw`**.

`min.mass` Cutting place when removing data corresponding to low masses (m/z). See function `msc.mass.cut` for details.

`mass.drift.adjustment` Controls mass drift adjustment and scaling. If 0 than no mass adjustment or scaling will be performed; otherwise, it is passed to `msc.mass.adjust` function as `scalePar`. Because of that: 1 means that afterwards all samples will have the same mean, 2 means that afterwards all samples will have the same mean and medium. See function `msc.mass.adjust` for details and additional parameter **`shiftPar`** that can be passed.

`peak.extraction` Perform peak extraction and alignment, or keep on working with the raw spectra? (boolean or 0/1 integer). See following functions for other parameters that can be passed:

- `msc.peaks.find` - see parameters: **`PeakFile`**, **`SNR`**, **`span`** and **`zerothresh`**
- `msc.peaks.align` - see parameters: **`BmrkFile`**, **`BinSize`** and **`tol`**
- `msc.biomarkers.fill` - see parameters: **`FlBmFile`** and **`FillType`**

Especially filenames to store intermediate results.

`merge.copies` In case multiple copies of data exist should they be merged and how? Passed to `msc.copies.merge` function as `mergeType` variable. See that function for more details.

`verbose` Boolean flag turns debugging printouts on.

`breaks` parameter to be passed to `bslnoff` function from **PROcess** library by `msc.baseline.subtract`

`qntl` parameter to be passed to `bslnoff` function from **PROcess** library by `msc.baseline.subtract`

`bw` parameter to be passed to `bslnoff` function from **PROcess** library by `msc.baseline.subtract`

`shiftPar` parameter to be passed to `msc.mass.adjust`

`PeakFile` parameter to be passed to `msc.peaks.find`

`SNR` parameter to be passed to `msc.peaks.find`

`span` parameter to be passed to `msc.peaks.find`

`zerothresh` parameter to be passed to `msc.peaks.find`

`BmrkFile` parameter to be passed to `msc.peaks.align`

`BinSize` parameter to be passed to `msc.peaks.align`

`tol` parameter to be passed to `msc.peaks.align`

`FlBmFile` parameter to be passed to `msc.biomarkers.fill`

`FillType` parameter to be passed to `msc.biomarkers.fill`

Details

Function containing several pre-processing steps preparing protein mass spectra (SELDI) data for classification. This function is a "pipeline" performing several operations, all of which do not need class label information. Any and all steps are optional and can be skipped:

- Remove baseline from each spectrum, using `msc.baseline.subtract` and `bslnoff` from **PROcess** library.
- Remove data corresponding to low masses (m/z), using `msc.mass.cut`.
- Adjust for mass drift and normalize data, using `msc.mass.adjust`.
- Find peaks and align them into "biomarker" matrix, using `msc.peaks.find`, `msc.peaks.align` and `msc.biomarkers.fill`.
- Merge multiple copies of data, using `msc.copies.merge`.

Value

Return matrix containing features as rows and samples as columns, unless `merge.copies` was 0,4, or 8 when no merging is done and data is returned in same or similar format as the input format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X)` store M/Z mass of each row.

See Also

- Input data likely come from `msc.preprocess.run` or `msc.msfiles.read.csv` functions
- As mentioned above function uses the following lower level functions: `msc.baseline.subtract`, `bslnoff` from **PROcess** library, `msc.mass.cut`, `msc.mass.adjust`, `msc.peaks.find`, `msc.peaks.align`, `msc.biomarkers.fill`, and `msc.copies.merge`.
- Output data can be latter used for classification using `msc.classifier.test` function

Examples

```
# load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run preprocess
Y = msc.preprocess.run(X)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")
```

2.5 msc.baseline.subtract - Baseline Subtraction for Mass Spectra Data

Description

Perform baseline subtraction on batch of mass spectra data

Usage

```
msc.baseline.subtract(X, ...)
```

Arguments

X Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X)` store M/Z mass of each row/feature.

... parameters to be passed to `bslnoff` function from **PROcess** library. See details for explanation of `breaks`, `qntl`, and `bw`. Boolean parameter `plot` can be used to plot results.

Details

Perform baseline subtraction for every sample in a batch of data, using `bslnoff` function from **PROcess** library. The `bslnoff` function splits spectrum into `breaks` number of exponentially growing regions. Baseline is calculated by appling `quantile(...,probs=qntl)` to each region and smoothing the results using `loess(..., span=bw, degree=1)` function.

Value

Data in the same format and size as input variable `X` but with the subtracted baseline.

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.project.read` and `msc.msfiles.read.csv`
- Next step in the pipeline is `msc.mass.cut`
- This function uses `bslnoff` (from **PROcess** library) which is a single-spectrum baseline removal function implemented using `loess` function.
- Function `rmBaseline` (from **PROcess** library) can read all CSV files in directory and remove their baselines.

Examples

```
# load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run msc.baseline.subtract using 3D input
Y = msc.baseline.subtract(X)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")

# test on data provided in PROcess package (2D input)
directory  = system.file("Test", package = "PROcess")
X = msc.msfiles.read.csv(directory)
Y = msc.baseline.subtract(X, plot=TRUE)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")
```

2.6 msc.mass.cut - Remove Low Mass Portion of the Mass Spectra Data.

Description

Remove low-mass portion of the protein mass spectra (SELDI) data.

Usage

`msc.mass.cut( X, MinMass=3000)`

Arguments

`X` Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X))` store M/Z mass of each row.

`MinMass` Minimum mass threshold. All data below that mass will be deleted

Details

Low-mass portion of the protein mass spectra is removed since it is not expected to have any biological information, and it has large enough amplitude variations that can skew normalization process. This function also removes all the masses (features) where the values in all the samples are identical. That happens sometimes when the ends of the samples are set to zero.

Value

Data in the similar format as input variable `X` but likely with fewer features.

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.baseline.subtract`
- Next step in the pipeline is `msc.mass.adjust`

Examples

```
# load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run in 3D input
Y = msc.mass.cut( X, MinMass=3000)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")

# test on data provided in PROcess package (2D input)
directory  = system.file("Test", package = "PROcess")
X = msc.msfiles.read.csv(directory)
Y = msc.mass.cut( X, MinMass=4000)
cat("Size before: ", dim(X), " and after :", dim(Y), "\n")
```

2.7 msc.mass.adjust - Perform Normalization and Mass Drift Adjustment for Mass Spectra Data.

Description

Perform normalization and mass drift adjustment for protein mass spectra (SELDI) data.

Usage

```
msc.mass.adjust(X, scalePar=2, shiftPar=0.0005, AvrSamp=0)
msc.mass.adjust.calc(X, scalePar=2, shiftPar=0.0005, AvrSamp=0)
msc.mass.adjust.apply(X, shiftX, scaleY, shiftY)
```

Arguments

`X` Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X))` store M/Z mass of each row.

`scalePar` Controls scaling (normalization): 1 means that afterwards all samples will have the same mean, 2 means that afterwards all samples will have the same mean and medium (default)

`shiftPar` Controls mass adjustment. Shifting sample has to improve correlation by at least that amount to be considered. Designed to prevent shifts based on "improvement" on order of magnitude of machine accuracy. If set to too large will turn off shifting. Default = 0.0005.

`AvrSamp` Is used to normalize test set the same way train set was normalized. Test set is processed using `AvrSamp` array that was one of the outputs from train-set mass-adjustment. See examples.

`shiftX` matrix [nSamp *x* nCopy] - integer number of positions a sample should be shifted to the right (+) or left (-). Output from `msc.mass.adjust.calc` and input to `msc.mass.adjust.apply`.

`scaleY` matrix [nSamp *x* nCopy] - multiply each sample in order to normalize it. Output from `msc.mass.adjust.calc` and input to `msc.mass.adjust.apply`.

`shiftY` matrix [nSamp *x* nCopy] - subtract this number from scaled sample (if matching medians). Output from `msc.mass.adjust.calc` and input to `msc.mass.adjust.apply`.

Details

Mass adjustment assumes that SELDI data has some error associated with inaccuracy of setting the starting point of time measurement (x-axis origin or zero M/Z value). We try to correct this error by allowing the samples to shift a few time-steps to the left or to the right, if that will help with cross-correlation with other samples. The function performs the following steps

- normalize all samples in such a way as to make their means (and optionally medians) the same
- if multiple copies exist than
    - align multiple copies of each sample to each other
    - temporarily merge multiple copies of each sample to create a "super-sample" vector with more features
- align each sample to the mean of all samples
- recalculate mean of all samples and repeat above step

`msc.mass.adjust` function was split into two parts (one to calculate parameters and one to apply them) in order to give users more flexibility and information about what is done to the data. This split allows inspection, plotting and/or modification of `shiftX`, `shiftY`, `scaleY` parameters before data is modified. For example one can set `shiftX` to zero to perform normalization without mass adjustment or set `shiftY` to zero and `scaleY` to one to perform mass adjustment without normalization. Three function provided are:

- `msc.mass.adjust.calc` - calculates and returns all the normalization and mass drift adjustment parameters
- `msc.mass.adjust.apply` - performs normalization and mass drift adjustment using precalculated parameters
- `msc.mass.adjust` - simple interface version of above 2 functions

Value

Functions `msc.mass.adjust` and `msc.mass.adjust.apply` return modified spectra in the same format and size as `X`. Functions `msc.mass.adjust.calc` returns list containing the following:

`shiftX` matrix [nSamp *x* nCopy] - integer number of positions sample should be shifted to the right (+) or left (-)

`scaleY` matrix [nSamp *x* nCopy] - multiply each sample in order to normalize it

`shiftY` matrix [nSamp *x* nCopy] - subtract this number from scaled sample (if matching mediums)

`AvrSamp` Use AvrSamp returned from train-set mass-adjustment to process test-set

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline is `msc.mass.cut`
- Next step in the pipeline is either `msc.peaks.find` or `msc.copies.merge`

Examples

```
  # load input data
  if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
  load("Data_IMAC.Rdata")

  # run on 3D input data using long syntax
  out = msc.mass.adjust.calc (X)
  Y   = msc.mass.adjust.apply(X, out$ShiftX, out$ScaleY, out$ShiftY)

  # check what happened to means
  Z   = cbind(colMeans(X), colMeans(Y))
  colnames(Z) = c("copy 1 before", "copy 2 before", "copy 1 after",
"copy 2 after" )
  cat("Sample means after and after:\n")
  Z

  # check what happen to sample correlation
  A = msc.sample.correlation(X, PeaksOnly=TRUE)
  B = msc.sample.correlation(Y, PeaksOnly=TRUE)
  cat("Mean corelation between two copies of the same sample:\n")
  cat(" before: ", mean(A$innerCor)," after: ", mean(B$innerCor), "\n")
  cat("Mean corelation between unrelated samples:\n")
  cat(" before: ", mean(A$outerCor)," after: ", mean(B$outerCor), "\n")

  # run on 2D input data using short syntax
  # check what happened to means and medians
  Y = msc.mass.adjust(X[,,1], scalePar=2)
  Z = cbind(colMeans(X[,,1]), apply(X[,,1],2,median), colMeans(Y),
apply(Y,2,median))
  colnames(Z) = c("means before", "medians before", "means after",
"medians after" )
  Z
  Y = msc.mass.adjust(X[,,1], scalePar=1)
  Z = cbind(colMeans(X[,,1]), apply(X[,,1],2,median), colMeans(Y),
apply(Y,2,median))
  colnames(Z) = c("means before", "medians before", "means after",
"medians after" )
  Z

  # mass adjustment for train and test sets, where test set is
normalized in
  # the same way as train set was
  Xtrain = X[, 1:10,]
  Xtest  = X[,11:20,]
  out    = msc.mass.adjust.calc (Xtrain);
  Xtrain = msc.mass.adjust.apply(Xtrain, out$ShiftX, out$ScaleY,
out$ShiftY)
  out    = msc.mass.adjust.calc (Xtest , AvrSamp=out$AvrSamp);
  Xtest  = msc.mass.adjust.apply(Xtest , out$ShiftX, out$ScaleY,
out$ShiftY)
```

2.8 msc.peaks.find - Find Peaks of Mass Spectra

Description

Find Peaks in a Batch of Protein Mass Spectra (SELDI) Data.

Usage

```
msc.peaks.find(X, PeakFile=0, SNR=2, span=c(81,11), zerothresh=0.9)
```

Arguments

`X` Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X))` store M/Z mass of each row.

`PeakFile` optional filename. If provided than CSV file will be created in the same format as Ciphergen's peak-info file, with following columns of data: "Spectrum.Tag", "Spectrum.", "Peak.", "Intensity" and "Substance.Mass".

`SNR` signal to noise ratio (z-score) criterion for peak detection. Similar to `SoN` variable in `isPeak` from **PROcess** package.

`span` two moving window widths. Smaller one will be used for smoothing and local maxima finding. Larger one will be used for local variance estimation. Similar to `span` and `sm.span` variables in `isPeak` from **PROcess** package.

`zerothresh` Intensity threshold criterion for peak detection. Positive numbers in range [0,1), like default 0.9, will be used to calculate a single threshold used for all samples using `quantile(X,zerothresh)` equation. Negative numbers in range (-1, 0) will be used to calculate threshold for each single sample `i` using `quantile(X[i,],-zerothresh)`. Similar to `zerothrsh` variable in `isPeak` from **PROcess** package.

Details

Peak finding is done using the following algorithm:

```
x      = X[j,]
thresh = if(zerothresh>=0) quantile(X,zerothresh) else quantile(x,-zerothresh)
sig    = runmean(x, span[2])
rMax   = runmax (x, span[2])
rAvr   = runmed (x, span[1])
rStd   = runmad (x, span[1], center=rAvr)
peak   = (rMax == x) & (sig > thresh) & (sig-rAvr > SNR*rStd)
```

What means that a peak have to meet the following criteria to be classified as a peak:

- be a local maxima in `span[2]` neighborhood
- smoothed sample (`sig`) is above user defined threshold `zerothresh`
- locally calculated z-score (see http://mathworld.wolfram.com/z-Score.html) of the signal is above user defined signal-to-noise ratio

It is very similar to the `isPeak` and `getPeaks` functions from **PROcess** library (ver 1.3.2) written by Xiaochun Li. For example `getPeaks(X, PeakFile, SoN=SNR, span=span[1], sm.span=span[2], zerothrsh=zerothresh, area.w=0.003, ratio=0)` would give very similar results as `msc.peaks.find` the differences include: speed (`msc.peaks.find` uses much faster C-level code), different use of signal-to-noise-ratio variable, and `msc.peaks.find` does not do or use area calculations.

Value

A data frame, in the same format as data saved in `peakinfofile`, have five components:

`Spectrum.Tag`	sample name of each peak
`Spectrum.`	sample number of each peak
`Peak.`	peak number within each sample
`Intensity`	peak height (intensity)
`Substance.Mass`	x-axis position, or corresponding mass of the peak measured in M/Z, which were extracted from row names of the `X` matrix.

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.mass.adjust`
- Functions `msc.peaks.align` or `pk2bmkr` can be used to align peaks from different samples in order to find biomarkers.

- Peak data can be read and writen by msc.peaks.read.csv and msc.peaks.write.csv.
- Functions isPeak and getPeaks from **PROcess** package are very similar.
- Uses runmax, runmean, runmed, runmad functions.

Examples

```
  # load input data
  if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
  load("Data_IMAC.Rdata")

  # Find Peaks
  Peaks = msc.peaks.find(X)
  cat(nrow(Peaks), "peaks were found in", Peaks[nrow(Peaks),2],
"files.\n")
```

2.9 msc.peaks.align - Align Peaks of Mass Spectra into a "Biomarker" Matrix

Description

Align peaks from multiple protein mass spectra (SELDI) samples into a single "biomarker" matrix

Usage

```
  msc.peaks.align(Peaks, BmrkFile=0, SampFrac=0.3, BinSize=c(0.002,
0.008), ...)
  msc.peaks.alignment(S,  M,  H, Tag=0, SampFrac=0.3, BinSize=c(0.002,
0.008), ...)
```

Arguments

`Peaks` Peak information. Could have two formats: a filename where to find the data, or the data itself. In the first case, `Peaks` is string containing path to a file saved by msc.peaks.find, getPeaks (from **PROcess** package), or by other software. In the second case, it is a data-frame in the same format as returned by msc.peaks.find. A third way to pass the same input data is through use of `S, M, H` and `Tag` variables (described below) used by `msc.peaks.alignment` function.

`S` Peak sample number. Unique number of the sample the peak belongs to. Likely to come from `Peaks$Spectrum.`.

`M` Peak center mass. Position of the peak on the x-axis. Likely to come from `Peaks$Substance.Mass`.

`H` Ppeak height. Likely to come from `Peaks$Intensity`.

`Tag` Peak sample name. Unique name of the sample the peak belongs to. Likely to come from `Peaks$Spectrum.Tag`. Optional since is used only to set column-names of output data.

`BmrkFile` Optional filename. If provided than CSV file will be created in the same format as Ciphergen's biomarker file, with spectra (samples) as rows, and biomarkers as columns (features).

`SampFrac` After peak alignment, bins with fewer peaks than `SampFrac*nSamp` are removed.

`BinSize` Upper and lower bound of bin-sizes, based on expected experimental variation in the mass (m/z) values. Size of any bin is measured as `(R-L)/mean(R,L)` where L and R are masses (m/z values) of left and right boundaries. All resulting bin sizes will all be between `BinSize[1]` and `BinSize[2]`. Since SELDI data is often assumed to have

+-

3% mass drift than a good bin size is twice that number (0.006). Same as `BinSize` variable in `msc.peaks.clust`, except for default.

`...` Two additional parameters that can be passed to `msc.peaks.clust` are mostly for expert users fine-tuning the code:

- tol - gaps bigger than `tol*max(gap)` are assumed to be the same size as the largest gap. See details.
- verbose - boolean flag turns debugging printouts on.

Details

Two interfaces were provided to the same function:

- `msc.peaks.alignment` is a lower level function with more detailed inputs and outputs. Possibly easier to customize for other purposes than processing SELDI data.
- `msc.peaks.align` is a higher level function with simpler interface customized for processing SELDI data.

This function align peaks from different samples into bins in such a way as to satisfy constraints in following order:

- bin sizes are in between `BinSize[1]` and `BinSize[2]`
- no two peaks from the same sample are present in the same bin
- bins are split in such a way as to minimize bin size and maximize spaces between bins
- if there are multiple, equally good, ways to split a bin than bin is split in such a way as to minimize number of repeats on each smaller sub-bin

The algorithm used does the following:

- Store mass and sample number of each peak into an array
- Concatenate arrays from all samples and sort them according to mass
- Group sets of peaks into subsets (bins). Each subset will consist of peaks from different spectra that have similar mass. That is done by puting all peaks into a single bin and recursivly going through the following steps:
    - Check size of the current bin: if it is too small than we are done, if it is too big than it will be split and if it is already in the desired range than it will be split only if multiple peaks from the same sample are present.
    - If bin needs to be split than find the biggest gap between peaks
    - If multiple gaps were found with the same size as the largest gap (or within `tol` tolerance from it) than minimizes number of multiple peaks from the same sample after cut
    - Divide the bin into two sub-bins: to the left and to the right of the biggest gap
    - Recursively repeat the above four steps for both sub-bins
- Store peaks into 2D array (bins by samples)
- Remove bins with fewer peaks than `SampFrac*nSamp`

The algorithm for peak alignment is described as recursive algorithm but the actual implementation uses internal stack, instead in order to increase speed.

Value

`Bmrks` Biomarker matrix containing one sample per column and one biomarker per row. If a given sample does not have a peak in some bin than `NA` is inserted.

`BinBounds` Mass of left-most and right-most peak in the bin

References

The initial version of this function started as implementation of algorithm described on webpage of Virginia Prostate Center (at Virginia Medical School) documenting their PeakMiner Software. See http://www.evms.edu/vpc/seldi/peakminer.pdf

See Also

- Input comes most Likely from: `msc.peaks.find`, `getPeaks` (from **PROcess** package), or Ciphergen's software
- Output can be processed further by: `msc.biomarkers.fill` or `msc.copies.merge`
- Part of `msc.preprocess.run` pipeline
- Uses `msc.peaks.clust` function to do most of the work
- Uses `msc.peaks.read.csv` function to read peak file
- Uses `msc.biomarkers.write.csv` function to save results

- Function pk2bmkr from **PROcess** package performs similar function.

Examples

```
  # load input data
  if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
  load("Data_IMAC.Rdata")

  # Find and Align peaks
  Peaks = msc.peaks.find(X)
  cat(nrow(Peaks), "peaks were found in", Peaks[nrow(Peaks),2],
"files.\n")
  Y = msc.peaks.align(Peaks)
  print( t(Y$Bmrks) , na.print=".",  digits=2)
```

2.10 msc.peaks.clust - Clusters Peaks of Mass Spectra

Description

Clusters peaks from multiple protein mass spectra (SELDI) samples

Usage

```
  msc.peaks.clust(dM, S, BinSize=c(0,sum(dM)), tol=0.97, verbose=FALSE)
```

Arguments

`S` Peak sample number, used to identify the spectrum the peak come from.

`dM` Distance between sorted peak positions (masses, m/z).

`BinSize` Upper and lower bound of bin-sizes, based on expected experimental variation in the mass (m/z) values. Size of any bin is measured as `(R-L)/mean(R,L)` where `L` and `R` are masses (m/z values) of left and right boundaries. All resulting bin sizes will be between `BinSize[1]` and `BinSize[2]`. Default is `c(0,sum(dM))` which ensures that no `BinSizes` is not being used.

`tol` gaps bigger than `tol*max(gap)` are assumed to be the same size as the largest gap. See details.

`verbose` boolean flag turns debugging printouts on.

Details

This is a low level function used by `msc.peaks.alignment` and not intended to be directly used by many users. However it might be usefull for other code developers. It clusters peaks from different samples into bins in such a way as to satisfy constraints in following order:

- bin sizes are in between `BinSize[1]` and `BinSize[2]`

- no two peaks from the same sample are present in the same bin
- bins are split in such a way as to minimize bin size and maximize spaces between bins
- if there are multiple, equally good, ways to split a bin than bin is split in such a way as to minimize number of repeats on each smaller sub-bin

Value

The output is binary array of the same size as `dM` and `S` where left boundaries of each clusters-bin (biomarker) are marked

References

The initial version of this function started as implementation of algorithm described on webpage of Virginia Prostate Center (at Virginia Medical School) documenting their PeakMiner Software. See http://www.evms.edu/vpc/seldi/peakminer.pdf

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.peaks.find`
- Next step in the pipeline is `msc.peaks.align` and `msc.biomarkers.fill`
- Part of `msc.peaks.align` function

Examples

```
  # example with simple made up data (18 peaks, 3 samples)
  M = c(1,5,8,12,17,22, 3,5,7,11,14,25, 1, 5, 7,10,17,21) # peak
position/mass
  S = c(1,1,1, 1, 1, 1, 2,2,2, 2, 2, 2, 3, 3, 3, 3, 3, 3) # peak's
sample number
  idx = sort(M, index=TRUE)$ix;  # sort peaks by mass
  M   = M[idx];                  # sorted mass
  S   = S[idx];                  # arrange sample numbers in the same
order
  bin = msc.peaks.clust(diff(M), S, verbose=TRUE)
  rbind(S,M,bin)                 # show results

  # use the results to align peaks into biomarkers matrix
  Bmrks = matrix(NA,sum(bin),max(S)); # init feature (biomarker) matrix
  bin   = cumsum(bin);                # find bin numbers for each peak
in S array
  for (j in 1:length(S))              # Bmrks usually store height H of
each peak
    Bmrks[bin[j], S[j]] =  1;         # but in this example it will be
"1"
  Bmrks
```

2.11 msc.peaks.read.csv & msc.peaks.write.csv - Read and Write Mass Spectra Peaks in CSV Format

Description

Functions to read and write CSV (comma separated values) text files containing peaks in the format used by Ciphergen's peak file.

Usage

```
X = msc.peaks.read.csv(fname)
msc.peaks.write.csv(fname, X)
```

Arguments

`fname`	either a character string naming a file or a connection.
`X`	Peak information. A data-frame in the same format as returned by `msc.peaks.find`, containing five components: - Spectrum.Tag - sample name of each peak - Spectrum. - sample number of each peak - Peak. - peak number within each sample - Intensity - peak height (intensity) - Substance.Mass - x-axis position, or corresponding mass of the peak measured in M/Z

Value

Function `msc.peaks.read.csv` returns peak information data frame. See argument `X` above. Function `msc.peaks.write.csv` does not return anything.

See Also

`msc.peaks.find` and `msc.peaks.align`

Examples

```
example("msc.peaks.find") # create peak data
X = Peaks                 # Peak data is stored in variable 'Peaks'
msc.peaks.write.csv("peaks.csv", X)
X = msc.peaks.read.csv("peaks.csv")
file.remove("peaks.csv")
stopifnot(X==Peaks)
```

2.12 msc.biomarkers.fill - Fill Empty Spaces in Biomarker Matrix

Description

Fill empty spaces (NA's) in biomarker matrix created by `msc.peaks.align`

Usage

```
msc.biomarkers.fill( X, Bmrks, BinBounds, FillType=0.9, BmrkFile=0)
```

Arguments

`X` Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X))` store M/Z mass of each row.

`Bmrks` biomarker matrix containing one sample per column and one biomarker per row

`BinBounds` position (mass) of left-most and right-most peak in each bin

`FillType` how to fill empty spaces in biomarker data?

- if `0<=FillType<=1` than fill spaces with `quantile(probs=FillType)`. For example: if `FillType=1/2` than medium will be used, if `FillType=1` than maximum value will be used, if `FillType=0.9` than maximum will be used after discarding 10% of "outliers"
- if `FillType<0` than empty spaces will not be filled and NA's will remain
- if `FillType==2` than X value closest to the center of the bin will be used
- if `FillType==3` empty spaces will be set to zero

`BmrkFile` Optional filename. If provided than CSV file will be created in the same format as Ciphergen's biomarker file, with spectra (samples) as rows, and biomarkers as columns.

Details

This function attepts to correct a problem which is a side-effect of `msc.peaks.align` function. Namely numerous NA's in biomarker data, each time when some peak was found only in some of the samples. `msc.peaks.align` already removed the most problematic features using `SampFrac` variable, but likelly a lot of NA's remain and they can cause problem for some classification algorithms.

Value

Data in the same format and size as `Bmrks`

Note

The whole idea of filling spaces in biomarker matrix is a little bit suspect since we are mixing proverbial apples and oranges. However, it might be better than the other options of filling empty spaces with zeros or keeping NA's.

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Input comes most likely from: `msc.peaks.align`, or from Ciphergen's software
- Output can be processed further by `msc.copies.merge`
- Biomarkers matrix can be read and writen by `msc.biomarkers.read.csv` and `msc.biomarkers.write.csv`.
- Function `pk2bmkr` from **PROcess** package lso perform similar function.

Examples

```
# load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")
Y = msc.peaks.align(msc.peaks.find(X))
dim(Y$Bmrks)
print( Y$Bmrks , na.print=".",  digits=2)

# run msc.biomarkers.fill
Z = msc.biomarkers.fill( X, Y$Bmrks, Y$BinBounds)
dim(Z)
print( Z[,,1] , na.print=".",  digits=2)
print( Z[,,2] , na.print=".",  digits=2)

# run msc.biomarkers.fill with other FillType
Z = msc.biomarkers.fill( X, Y$Bmrks, Y$BinBounds, FillType=2)
```

msc.biomarkers.read.csv {caMassClass}　　　　　　　　　R Documentation

2.13 msc.biomarkers.read.csv & msc.biomarkers.write.csv - Read and Write biomarker matrix in CSV format

Description

Functions to read and write CSV (comma separated values) text files containing biomarkers (aligned peaks) in the format used by Ciphergen's biomarker file, with spectra (samples) as rows, and biomarkers as columns (features).

Usage

```
  X = msc.biomarkers.read.csv(fname)
  msc.biomarkers.write.csv(fname, X)
```

Arguments

`fname`	either a character string naming a file or a connection.
`X`	biomarker matrix containing one sample per column and one biomarker per row. Notice that this data is in format which is a transpose of data in CSV file.

Value

Function `msc.biomarkers.read.csv` returns peak information data frame. See argument `X` above. Function `msc.biomarkers.write.csv` does not return anything.

See Also

`msc.biomarkers.fill`

Examples

```
  example("msc.peaks.align", verbose=FALSE) # create biomarkers data
  X = Y$Bmrks   #  biomarkers data is stored in variable 'Y$Bmrks'
  msc.biomarkers.write.csv("biomarkers.csv", X)
  Y = msc.biomarkers.read.csv("biomarkers.csv")
  file.remove("biomarkers.csv")
  stopifnot( all(X==Y, na.rm=TRUE) )
```

2.14 msc.copies.merge - Merge Multiple Copies of Mass Spectra Samples

Description

Protein mass spectra (SELDI) samples are sometimes scanned multiple times in order to reduce hardware or software based errors. `msc.copies.merge` function is used to merge, concatenate, and/or average all of those copies together in preparation for classification.

Usage

```
msc.copies.merge( X, mergeType, PeaksOnly=TRUE)
```

Arguments

`X` Spectrum data in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names (`rownames(X)`) store M/Z mass of each row. If X is in matrix format [nFeatures *x* nSamples] nothing will be done.

`mergeType` an integer variable in [0,11] range, telling how to merge samples and what to do with bad copies:

- 0 - do nothing
- add 1 - if all original copies are to be concatenated as separate samples
- add 2 - if copies are to be averaged and the average added as a separate sample
- add 4 - if for each sample the worst copy is to be deleted
- add 8 - if for each sample in case of large differences between copies, a single bad copy of a sample is to be replaced with the best copy. Not to be used with previous option. See details.

`PeaksOnly` This variable is being passed to function `msc.sample.correlation`. Set it to `TRUE` in case of raw spectra and switch to `FALSE` in case of data where only peaks (biomarkers) are present.

Details

Quality of a sample is measured by calculating for each copy of each sample two variables: inner correlation (average correlation between multiple copies of the same sample) and outer correlation (average correlation between each sample and every other sample within the same copy). Inner correlation measures how similar copies are to each other and outer correlation measures how similar each copy is to everybody else. For example in case of experiment using SELDI technology to distinguish cancerous samples and non-cancerous samples one can assume that most of the proteins present in both cancerous and non-cancerous samples will be the same. In that case one will expect high correlation between samples and even higher correlation between copies of the same sample

if `mergeType/4` (`mergeType %/% 4`) is

- 0 - all copies are kept
- 1 - if inner correlation is smaller than outer correlation, or in other words, if a signature is more similar to other signatures than to other copies of the same signature, than there is some problem with that signature. In that case that bad signature can be replaced with the best copy of the signature.
- 2 - rate each copy of each sample using `score=outer_correlation + inner_correlation` measure. Delete worst copy.

Option 2 is more suitable in case of data with a lot of copies, when we can afford dropping one copy. Option 1 is designed to patch the most serious problems with the data.

There are also four merging options, if `mergeType mod 4` (`mergeType %% 4`) is

- 0 - no merging is done to the data and it is left as 3D array
- 1 - all copies are concatenated `X = cbind(X[,,1], X[,,2], ..., X[,,nCopy])` so they seem as separate samples
- 2 - all copies are averaged `X = (X[,,1] + X[,,2] + ... + X[,,nCopy])/nCopy)`
- 3 - all copies are first averaged and than concatenated with extra average copy `X = cbind(X[,,1], X[,,2], ..., X[,,nCopy], Xavr)`

In preparation for classification one can use multiple copies in several ways: option 2 above improves (one hopes) accuracy of each sample, while options 1 and 3 increase number of samples available during classification. So the choice is: do we want a lot of samples during classification or fewer, better samples?

The best option of `mergeType` depends on kind of data.

- 0 if data has single copy.
- 1+2+4 will produce the largest number of samples since we will keep all the copies and an average of all the copies
- 2+8 will produce single most accurate sample from multiple copies (usually if more than 2 copies are present) since we will delete outliers before averaging all the copies

Value

Return matrix containing features as rows and samples as columns, unless `mergeType` is 0,4, or 8 when no merging is done and data is returned in same or similar format as the input format.

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Previous step in the pipeline was `msc.mass.adjust` or peak finding functions: `msc.peaks.find`, `msc.peaks.align`, and `msc.biomarkers.fill`
- Next step in the pipeline is data classification `msc.classifier.test`
- Uses `msc.sample.correlation`

Examples

```
# load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
```

```
load("Data_IMAC.Rdata")

# run msc.copies.merge
Y = msc.copies.merge(X, 1+2+4)
colnames(Y)
```

2.15 msc.classifier.test - Test a Classifier through Cross-validation

Description

Test Classifier through Cross-validation. Common interface for Cross-validation of several standard classifiers. Includes feature selection and feature scaling steps. Allows to specify that some test samples are multiple copies of the same sample, and should return the same label.

Usage

```
 msc.classifier.test( X, Y, iters=50, SplitRatio=2/3, verbose=FALSE,
                    RemCorrCol=0, KeepCol=0, prior=1, same.sample=NULL,
                    ScaleType=c("none", "min-max", "avr-std", "med-mad"),
                    method=c("svm", "nnet", "lda", "qda", "LogitBoost",
"rpart"), ...)
```

Arguments

`X` A matrix or data frame with training data. Rows contain samples and columns contain features/variables

`Y` Class labels for the training data samples. A response vector with one label for each row/component of `x`. Can be either a factor, string or a numeric vector. Labels with 'NA' value signify test data-set.

`iters` Number of iterations. Each iteration consist of splitting the data into train and test sets, performing the classification and storing results

`SplitRatio` Splitting ratio:

- if `(0<=SplitRatio<1)` then `SplitRatio` fraction of points from Y will be set toTRUE
- if `(SplitRatio==1)` then one random point from Y will be set to TRUE
- if `(SplitRatio>1)` then `SplitRatio` number of points from Y will be set to TRUE

`RemCorrCol` See `msc.classifier.run`.

`KeepCol` See `msc.classifier.run`.

`ScaleType` See `msc.classifier.run`.

`prior` See `msc.classifier.run`.

`same.sample` See `msc.classifier.run`.

`method` See `msc.classifier.run`.

`verbose` boolean flag turns debugging printouts on.

`...` Additional parameters to be passed to classifiers. See `method` for suggestions.

Details

This function follows standard cross-validation steps:

- Class labels `Y` are used to divide data `X` into train set (with known labels) and test set (labels are unknown and will be calculated)
- For number of iterations repeat the following steps:
    - split train data into temporary train and test sets using `msc.sample.split` function
    - train and test the chosen classifier using temporary train and test data sets and `msc.classifier.run` function
- Calculate the overall performance of the calassifer
- Train the classifier using the whole train data set (all labaled samples)
- Use this classifier to predict values of the whole test data set (all samples without label - NA.)

Value

`Y` Predicted class labels. If there were any unknown samples in input data, marked by `NA`'s in input `Y`, than output `Y` will only hold prediction of those samples, otherwise prediction will be made for all samples.

`Res` Holds fraction of correct prediction during cross-validation for each iteration. `mean(Res)` will give you average acuracy.

`Tabl` Contingency table of predictions shows all the input label compared to output labels

Note

This function is not fully tested and might be changed in future versions

See Also

- Input comes most likely from `msc.preprocess.run` and/or `msc.project.run` functions.
- Uses `msc.classifier.run`, `msc.features.select` and `msc.features.scale` functions.
- Best classifier parameter set can be found by `tune` function from **e1071** package.

- Uses variety of classification algorithms: svm, nnet, LogitBoost, lda, qda, rpart

Examples

```
  data(iris)
  A = msc.classifier.test(iris[,-5],iris[,5], method="LogitBoost",
nIter=2)
  A
  cat("correct classification in",100*mean(A$Res),"+-
",100*sd(A$Res),"percent of cases\n")
```

2.16 msc.classifier.run - Train and Test Chosen Classifier.

Description

Common interface for training and testing several standard classifiers. Includes feature selection and feature scaling steps. Allows to specify that some test samples are multiple copies of the same sample, and should return the same label.

Usage

```
msc.classifier.run( xtrain, ytrain, xtest, ret.prob=FALSE,
                  RemCorrCol=0, KeepCol=0, prior=1, same.sample=NULL,
                  ScaleType=c("none", "min-max", "avr-std", "med-mad"),
                  method=c("svm", "nnet", "lda", "qda", "LogitBoost",
"rpart"), ...)
```

Arguments

`xtrain` A matrix or data frame with training data. Rows contain samples and columns contain features/variables

`ytrain` Class labels for the training data samples. A response vector with one label for each row/component of `x`. Can be either a factor, string or a numeric vector.

`xtest` A matrix or data frame with test data. Rows contain samples and columns contain features/variables

`ret.prob` if set to TRUE than the a-posterior probabilities for each class are returned as attribute called "probabilities".

`same.sample` optional parameter which allows to specify that some (or all) test samples have multiple copies which should be used to predict a single label for all of them. Can be either a factor, string or a numeric vector, with unique values for different samples and identical values for copies of the same sample.

`RemCorrCol` If non-zero than some of the highly correlated columns are removed using `msc.features.remove` function with `ccMin=RemCorrCol`.

`KeepCol`
If non-zero than columns with low AUC are removed.

- if `KeepCol` smaller than 0.5 - do nothing
- if `KeepCol` in between [0.5, 1] - keep columns with AUC bigger than `KeepCol`
- if `KeepCol` bigger than one - keep top "`KeepCol`" number of columns

`ScaleType`
Optional parameter, if provided than following types are recognized

- "none" - no scaling is performed
- "min-max" - data minimum is mapped to 0 and maximum is mapped to 1
- "avr-std" - data is mapped to zero mean and unit variance
- "med-mad" - data is mapped to zero median and unit mad (median absolute deviation)

`prior`
class weights. following types are recognized

- `prior==1` - all samples in all classes have equal weight (default)
- `prior==2` - all classes have equal weight
- `prior` is a vector - a named vector of weights for the different classes, used for asymetric class sizes.

`method`
classifier to be used. Following ones are recognized (followed by some parameters that could be passed through ... :

- "svm" - see `svm` from **e1071** package. Possible parameters: `cost`, `gamma`
- "nnet" - see `nnet` from **nnet** package. Possible parameters: `size`, `decay`, `maxit`
- "LogitBoost" - see `LogitBoost`. Possible parameter: `nIter`
- "lda" - see `lda` from **MASS** package. Possible parameters: `method`
- "qda" - see `qda` from **MASS** package. Possible parameters: `method`
- "rpart" - see `rpart` from **rpart** package. Possible parameters: `minsplit`, `cp`, `maxdepth`

`...`
Additional parameters to be passed to classifiers. See `method` for suggestions.

Details

This function performs the following steps:

- Remove highly correlated columns and columns with low AUC with msc.features.select function
- Scale each feature separatly using msc.features.scale function
- Train chosen classifier using xtrain and ytrain
- Predict labels of xtest using trained model
- If same.sample variable is given than synchronise predicted labels in such a way that all copies of the same sample return the same label.
- Return labels. If ret.prob=TRUE then return a-posterior probabilities as well.

Value

Predicted class labels for each sample in xtest. If ret.prob=TRUE than the a-posterior probabilities of each sample belonging to each class are returned as attribute called "probabilities". The returned probabilities do not take into account same.sample variable, used to synchronize predicted labels.

Note

This function is not fully tested and might be changed in future versions

References

- "A Practical Guide to Support Vector Classification" by Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin (http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf)

See Also

- Used by msc.classifier.test function.
- Best classifier parameter set can be found by tune function from **e1071** package.
- Uses msc.features.select and msc.features.scale functions.
- Uses variety of classification algorithms: svm, nnet, LogitBoost, lda, qda, rpart

Examples

```
  data(iris)
  mask   = msc.sample.split(iris[,5], SplitRatio=1/4) # very few points 
to train
  xtrain = iris[ mask,-5]   # use output of msc.sample.split to ...
  xtest  = iris[!mask,-5]   # create train and test subsets
  ytrain = iris[ mask, 5]
  ytest  = iris[!mask, 5]
  table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="svm")
)
  table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, 
method="LogitBoost") )
```

```
  table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="nnet")
)
  table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="lda")
)
  table(ytrain, msc.classifier.run(xtrain,ytrain,xtrain, method="qda")
)
```

2.17 msc.sample.split - Split Data into Test and Train Set

Description

Split data from vector Y into two sets in predefined ratio while preserving relative ratios of different labels in Y. Used to split the data used during classification into train and test subsets.

Usage

```
msc.sample.split( Y, SplitRatio = 2/3, group = NULL )
```

Arguments

`Y` Vector of data labels. If there are only a few labels (as is expected) than relative ratio of data in both subsets will be the same.

`SplitRatio` Splitting ratio:

- if `(0<=SplitRatio<1)` then `SplitRatio` fraction of points from Y will be set toTRUE
- if `(SplitRatio==1)` then one random point from Y will be set to TRUE
- if `(SplitRatio>1)` then `SplitRatio` number of points from Y will be set to TRUE

`group` Optional vector/list used when multiple copies of each sample are present. In such a case `group` contains unique sample labels, marking all copies of the same sample with the same label, and the function tries to place all copies in either train or test subset. If provided than has to have the same length as `Y`.

Value

Returns logical vector of the same length as Y with random `SplitRatio*length(Y)` elements set to TRUE.

See Also

- Used by msc.classifier.test function.
- Similar to sample function.
- Variable group is used in the same way as f argument in split and INDEX argument in tapply

Examples

```
  library(MASS)
  data(cats)   # load cats data
  Y = cats[,1] # extract labels from the data
  msk = msc.sample.split(Y, SplitRatio=3/4)
  table(Y,msk)
  t=sum( msk)  # number of elements in one class
  f=sum(!msk)  # number of elements in the other class
  stopifnot( round((t+f)*3/4) == t ) # test ratios

  # example of using group variable
  g = rep(seq(length(Y)/4), each=4); g[48]=12;
  msk = msc.sample.split(Y, SplitRatio=1/2, group=g)
  table(Y,msk) # try to get correct split ratios ...
  split(msk,g) # ... while keeping samples with the same group label
together

  # test results
  print(paste( "All Labels numbers: total=",t+f,", train=",t,",
test=",f,
        ", ratio=", t/(t+f) ) )
  U = unique(Y)       # extract all unique labels
  for( i in 1:length(U)) {  # check for all labels
    lab = (Y==U[i])   # mask elements that have label U[i]
    t=sum( msk[lab])  # number of elements with label U[i] in one class
    f=sum(!msk[lab])  # number of elements with label U[i] in the other
class
    print(paste( "Label",U[i],"numbers: total=",t+f,", train=",t,",
test=",f,
                 ", ratio=", t/(t+f) ) )
  }

  # use results
  train = cats[ msk,2:3]  # use output of msc.sample.split to ...
  test  = cats[!msk,2:3]  # create train and test subsets
  z = lda(train, Y[msk])  # perform classification
  table(predict(z, test)$class, Y[!msk]) # predicted & true labels

  # see also LogitBoost example
```

2.18 msc.features.select - Reduce Number of Features Prior to Classification

Description

Select subset of individual features that are potentially most useful for classification.

Usage

```
msc.features.select( x, y, RemCorrCol=0.98, KeepCol=0.6)
```

Arguments

`x` A matrix or data frame with training data. Rows contain samples and columns contain features/variables

`y` Class labels for the training data samples. A response vector with one label for each row/component of `x`. Can be either a factor, string or a numeric vector.

`RemCorrCol` If non-zero than some of the highly correlated columns are removed using `msc.features.remove` function with `ccMin=RemCorrCol`.

`KeepCol` If non-zero than columns with low AUC are removed.

- if `KeepCol` smaller than 0.5 - do nothing
- if `KeepCol` in between [0.5, 1] - keep columns with AUC bigger than `KeepCol`
- if `KeepCol` bigger than one - keep top `KeepCol` number of columns

Details

This function reduces number of features in the data prior to classification, using following steps:

- calculate AUC measure for each feature using `colAUC`
- remove some of the highly correlated neighboring columns using `msc.features.remove` function.
- remove columns with low AUC

This function finds subset of individual features that are potentially most useful for classification, and each feature is rated individually. However, often set of two or more very poor individual features can produce a superior classifier. So, this function should be used with care. I found it very useful when classifying raw protein mass spectra (SELDI) data, for reducing dimensionality of the data from 10 000's to 100's prior of classification, instead of peak-finding (see `msc.peaks.find`).

Value

Vector of column indexes to be kept.

See Also

- Used by `msc.classifier.test` function.

- Uses colAUC, msc.features.remove and msc.features.scale functions.

Examples

```
  # load input data
  if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
  load("Data_IMAC.Rdata")

  X = t(X[,,1])
  cidx = msc.features.select(X, SampleLabels, KeepCol=0.7)
  cat(length(cidx),"features were selected out
of",ncol(X),"min(auc)=0.7\n")
  cidx = msc.features.select(X, SampleLabels, KeepCol=400)
  cat(length(cidx),"features were selected out of",ncol(X),"\n")
  cat(" min(auc)=", min(colAUC(X[,cidx], SampleLabels)),"\n")
  Y = X[,cidx]
```

2.19 msc.features.remove - Remove Highly Correlated Features

Description

Remove Highly Correlated Features. The function checks neighbor features looking for highly correlated ones and removes one of them. Used in order to drop dimensionality of the data.

Usage

```
msc.features.remove(Data, Auc, ccMin=0.9, verbose=FALSE)
```

Arguments

`Data` Data containing one sample per row and one feature per column.

`Auc` A measure of usefulness of each column/feature, used to choose which one of two highly correlated columns to remove. Usually a measure of discrimination power of each feature as measured by colAUC, student t-test or other method. See details.

`ccMin` Minimum correlation coefficient of "highly correlated" columns.

`verbose` Boolean flag turns debugging printouts on.

Details

If colAUC was used and there were more than two classes present than `Auc` is a matrix with multiple measurments for each feature. In such a case `Auc = apply(Auc, 2, mean)` is run in order to extract a single measure per feature. If other measures are desired, like `Auc = apply(Auc, 2, max)`, than they should be called beforehand.

Value

Vector of column indexes to be kept.

See Also

- Used by msc.classifier.test and msc.features.select functions.
- Uses colAUC

Examples

```
# load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

X = t(X[,,1])
auc = colAUC(X,SampleLabels)
quantile(auc)
cidx = msc.features.remove(X, auc, verbose=TRUE)
Y = X[,cidx]
```

2.20 msc.features.scale - Scale Classification Data

Description

Scale features of the data to be used for classification. Scaling factors are extracted from each column/feature of the train data-set and applied to both train and test sets.

Usage

```
msc.features.scale( xtrain, xtest, type = c("min-max", "avr-std", "med-
mad"))
```

Arguments

`xtrain` A matrix or data frame with train data. Rows contain samples and columns contain features/variables

`xtest` A matrix or data frame with test data. Rows contain samples and columns contain features/variables

`type` Following types are recognized

- "min-max" - data minimum is mapped to 0 and maximum is mapped to 1
- "avr-std" - data is mapped to zero mean and unit variance
- "med-mad" - data is mapped to zero median and unit mad (median absolute deviation)

Details

Many classification algorithms perform better if input data is scaled beforehand. Some of them perform scaling internally (for example `svm`), but many don't. For some it makes no difference (for example `rpart` or `LogitBoost`).

In case `xtrain` contains `NA` values or infinities all non-finite numbers are omited from scaling parameter calculations.

Value

`xtrain` A matrix or data frame with scaled train data.

`xtest` A matrix or data frame with scaled test data.

See Also

Used by `msc.classifier.test` and `msc.features.select` functions.

Examples

```
  library(e1071)
  data(iris)
  mask  = msc.sample.split(iris[,5], SplitRatio=1/4) # very few points
to train
  xtrain = iris[ mask,-5]  # use output of msc.sample.split to ...
  xtest  = iris[!mask,-5]  # create train and test subsets
  ytrain = iris[ mask, 5]
  ytest  = iris[!mask, 5]
  x = msc.features.scale(xtrain, xtest)
  model = svm(x$xtrain, ytrain, scale=FALSE)
  table(predict(model, x$xtest), ytest)
  model = svm(xtrain, ytrain, scale=FALSE)
  table(predict(model, xtest), ytest)
```

2.21 msc.sample.correlation - Sample Correlation

Description

Calculates correlations between different samples and correlations between different copies of the same sample

Usage

```
msc.sample.correlation(X, PeaksOnly=FALSE)
```

Arguments

`X` Spectrum data either in matrix format [nFeatures *x* nSamples] or in 3D array format [nFeatures *x* nSamples *x* nCopies]. Row names `(rownames(X))` store M/Z mass of each row.

`PeaksOnly` Should only peaks be used in calculating the correlation? In case of raw mass spectra data it does not make much sense to calculate correlation of "valleys" between peaks so one can set this flag to TRUE and only points above sample mean will be used.

Details

Function calculates for each copy of each sample two variables:

- inner correlation - average correlation between multiple copies of the same sample. Inner correlation measures how similar copies are to each other. For example `innerCor[iSamp,iCopy]` measures average correlation between `X[,iSamp,iCopy]` and all other copies of that sample. In case of one copy of the data `innerCor` is set to one. In case of two copies `innerCor[iSamp,1]=innerCor[iSamp,2] =cor(X[,iSamp,1],X[,iSamp,2])`. In case of 3 copies `innerCor[iSamp,1] = (cor(X[,iSamp,1],X[,iSamp,2]) + cor(X[,iSamp,1],X[,iSamp,3]))/2` ,etc.
- outer correlation - average correlation between each sample and every other sample within the same copy. Outer correlation measures how similar each copy is to everybody else.

Value

Returns list with two components: `innerCor` and `outerCor` both of size [nSamples *x* nCopies].

See Also

- Part of `msc.preprocess.run` and `msc.project.run` pipelines.
- Used by `msc.copies.merge` function.
- Uses `cor` function

Examples

```
 # load input data
if (!file.exists("Data_IMAC.Rdata")) example("msc.project.read")
load("Data_IMAC.Rdata")

# run in 3D input data using long syntax
out = msc.mass.adjust.calc (X);
Y   = msc.mass.adjust.apply(X, out$ShiftX, out$ScaleY, out$ShiftY)

# check what happen to sample correlation
A = msc.sample.correlation(X, PeaksOnly=TRUE)
```

```
B = msc.sample.correlation(Y, PeaksOnly=TRUE)
cat("Mean corelation between two copies of the same sample:\n")
cat(" before: ", mean(A$innerCor)," after: ", mean(B$innerCor), "\n")
cat("Mean corelation between unrelated samples:\n")
cat(" before: ", mean(A$outerCor)," after: ", mean(B$outerCor), "\n")
```

3 Generic Tool Functions

The functions in this section are generic tools that were written in order to support the rest of the library.

3.1 LogitBoost - LogitBoost Classification Algorithm

Description

Train logitboost classification algorithm using decision stumps (one node decision trees) as weak learners.

Usage

```
LogitBoost(xlearn, ylearn, nIter=ncol(xlearn))
```

Arguments

`xlearn` A matrix or data frame with training data. Rows contain samples and columns contain features

`ylearn` Class labels for the training data samples. A response vector with one label for each row/component of `xlearn`. Can be either a factor, string or a numeric vector.

`nIter` An integer, describing the number of iterations for which boosting should be run, or number of decision stumps that will be used.

Details

The function was adapted from logitboost.R function written by Marcel Dettling. See references and "See Also" section. The code was modified in order to make it much faster for very large data sets. The speed-up was achieved by implementing a internal version of decision stump classifier instead of using calls to `rpart`. That way, some of the most time consuming operations were precomputed once, instead of performing them at each iteration. Another difference is that training and testing phases of the classification process were split into separate functions.

Value

An object of class "LogitBoost" including components:

`Stump` List of decision stumps (one node decision trees) used:

- column 1: feature numbers or each stump, or which column each stump operates on
- column 2: threshold to be used for that column
- column 3: bigger/smaller info: 1 means that if values in the column are above threshold than corresponding samples will be labeled as `lablist[1]`. Value "-1" means the opposite.

If there are more than two classes, than several "Stumps" will be `cbind`'ed

`lablist` names of each class

References

See "Boosting for Tumor Classification of Gene Expression Data", Dettling and Buhlmann (2002), available on the web page http://stat.ethz.ch/~dettling/boosting.html.

http://www.cs.princeton.edu/~schapire/boost.html

See Also

- `predict.LogitBoost` has prediction half of LogitBoost code
- `logitboost` function from **boost** library
- `logitboost` function from **logitboost** library (not in CRAN or BioConductor but can be found at http://stat.ethz.ch/~dettling/boosting.html) is very similar but much slower on very large datasets. It also perform optional cross-validation.

Examples

```
data(iris)
Data  = iris[,-5]
Label = iris[, 5]

# basic interface
model = LogitBoost(Data, Label, nIter=20)
Lab   = predict(model, Data)
Prob  = predict(model, Data, type="raw")
t     = cbind(Lab, Prob)
t[1:10, ]

# two alternative call syntax
p=predict(model,Data)
q=predict.LogitBoost(model,Data)
pp=p[!is.na(p)]; qq=q[!is.na(q)]
stopifnot(pp == qq)
```

```
# accuracy increases with nIter (at least for train set)
table(predict(model, Data, nIter= 2), Label)
table(predict(model, Data, nIter=10), Label)
table(predict(model, Data),           Label)

# example of spliting the data into train and test set
mask = msc.sample.split(Label)
model = LogitBoost(Data[mask,], Label[mask], nIter=10)
table(predict(model, Data[!mask,], nIter=2), Label[!mask])
table(predict(model, Data[!mask,]),          Label[!mask])
```

3.2 predict.LogitBoost - Prediction Based on LogitBoost Classification Algorithm

Description

Prediction or Testing using logitboost classification algorithm

Usage

```
predict.LogitBoost(object, xtest, type = c("class", "raw"), nIter=NA,
...)
```

Arguments

`object` An object of class "LogitBoost" see "Value" section of `LogitBoost` for details

`xtest` A matrix or data frame with test data. Rows contain samples and columns contain features

`type` See "Value" section

`nIter` An optional integer, used to lower number of iterations (decision stumps) used in the decision making. If not provided than the number will be the same as the one provided in `LogitBoost`. If provided than the results will be the same as running `LogitBoost` with fewer iterations.

`...` not used but needed for compatibility with generic predict method

Details

Logitboost algorithm relies on a voting scheme to make classifications. Many (`nIter` of them) week classifiers are applied to each sample and their findings are used as votes to make the final classification. The class with the most votes "wins". However, with this scheme it is common for two cases have a tie (the same number of votes), especially if number of iterations is even. In that case NA is returned, instead of a label.

Value

If type = "class" (default) label of the class with maximal probability is returned for each sample. If type = "raw", the a-posterior probabilities for each class are returned.

See Also

LogitBoost has training half of LogitBoost code

Examples

```
# See LogitBoost example
```

3.3 Moving Window Analysis of a Vector

Description

A collection of functions to perform fast moving window (running, rolling window) analysis of vectors.

Usage

```
  runmean(x, k, endrule=c("NA", "trim", "keep", "constant", "func"))
  runmin (x, k, endrule=c("NA", "trim", "keep", "constant", "func"))
  runmax (x, k, endrule=c("NA", "trim", "keep", "constant", "func"))
  runmad (x, k, center=runmed(x,k,endrule="keep"), constant=1.4826,
          endrule=c("NA", "trim", "keep", "constant", "func"))
  runquantile(x, k, probs, type=7,
          endrule=c("NA", "trim", "keep", "constant", "func"))
  EndRule(x, y, k,
          endrule=c("NA", "trim", "keep", "constant", "func"), Func,
...)
```

Arguments

`x`	numeric vector of length n
`k`	width of moving window; must be an odd integer bigger than one.
`endrule`	character string indicating how the values at the beginning and the end, of the data, should be treated. Only first and last `k2` values at both ends are affected, where `k2` is the half-bandwidth `k2 = k %/% 2`.

- `"trim"` - trim the ends output array length is equal to `length(x)-2*k2` `(out = out[(k2+1):(n-k2)])`. This option mimics output of `apply` `(embed(x,k),1,FUN)` and other related functions.
- `"keep"` - fill the ends with numbers from `x` vector `(out[1:k2] = x[1:k2])`
- `"constant"` - fill the ends with first and last calculated value in output array `(out[1:k2] = out[k2+1])`

- `"NA"` - fill the ends with NA's `(out[1:k2] = NA)`
- `"func"` - applies the underlying function to smaller and smaller sections of the array. For example in case of mean: `for(i in 1:k2) out[i]=mean(x[1:i])`. This option is not optimized and could be very slow for large windows.

Similar to `endrule` in `runmed` function which has the following options: “`c("median", "keep", "constant")`”.

`center` moving window center used by `runmad` function defaults to running median (`runmed` function). Similar to `center` in `mad` function.

`constant` scale factor used by `runmad`, such that for gaussian distribution X, `mad`(X) is the same as `sd`(X). Same as `constant` in `mad` function.

`probs` numeric vector of probabilities with values in [0,1] range used by `runquantile`. For example `Probs=c(0,0.5,1)` would be equivalent to running `runmin`, `runmed` and `runmax`. Same as `probs` in `quantile` function.

`type` an integer between 1 and 9 selecting one of the nine quantile algorithms, same as `type` in `quantile` function. Another even more readable description of nine ways to calculate quantiles can be found at http://mathworld.wolfram.com/Quantile.html.

`y` numeric vector of length n, which is partially filled output of one of the `run` functions. Function `EndRule` will fill the remaining begining and end sections using method chosen by `endrule` argument.

`Func` Function name that `EndRule` will use in case of `endrule="func"`.

`...` Additionam parameters that `EndRule` will use in case of `endrule="func"`.

Details

Apart from the end values, the result of y = runFUN(x, k) is the same as “`for(j=(1+k2):(n-k2)) y[j]=FUN(x[(j-k2):(j+k2)])`”, where FUN stands for min, max, mean, mad or quantile functions.

The main incentive to write this set of functions was relative slowness of majority of moving window functions available in R and its packages. With exception of `runmed`, a running window median function, all functions listed in "see also" section are slower than very inefficient “`apply(embed(x,k),1,FUN)`” approach. Speeds of above functions are as follow:

- `runmin`, `runmax`, `runmean` run at O(n)
- `runquantile` and `runmad` run at O(n*k)
- `runmed` - related R function run at O(n*log(k))

Functions `runquantile` and `runmad` are using insertion sort to sort the moving window, but gain speed by remembering results of the previous sort. Since each time the window is moved, only one point changes, all but one points in the window are already sorted. Insertion sort can fix that in O(k) time.

Function runquantile when run in single probability mode automatically recognizes probabilities: 0, 1/2, and 1 as special cases and return output from functions: runmin, runmed and runmax respectivly.

All run* functions are written in C, but runmin, runmax and runmean also have R code versions that can be used if DLL is not loaded.

Function EndRule applies one of the five methods (see endrule argument) to process end-points of the input array x.

Value

Functions runmin, runmax, runmean and runmad return a numeric vector of the same length as x. Function runquantile returns a matrix of size [n *x* length(probs)].

References

Hyndman, R. J. and Fan, Y. (1996) Sample quantiles in statistical packages, American Statistician, 50, 361.

See Also

Links related to each function:

- runmin - min, rollMin from **fSeries** library
- runmax - max, rollMax from **fSeries** library
- runmean - mean, kernapply, filter, rollMean from **fSeries** library, subsums from **magic** library
- runquantile - quantile, runmed, smooth
- runmad - mad, rollVar from **fSeries** library
- generic running window functions: apply (embed(x,k), 1, FUN) (fastest), rollFun from **fSeries** (slow), running from **gtools** package (extrimly slow for this purpose), subsums from **magic** library can perform running window operations on data with any dimensions.
- EndRule - smoothEnds(y,k) function is similar to EndRule(x,y,k,endrule="func", median)

Examples

```
# test runmin, runmax and runmed
k=15; n=200;
x = rnorm(n,sd=30) + abs(seq(n)-n/4)
col = c("black", "red", "green", "blue", "magenta", "cyan")
```

```
  plot(x, col=col[1], main = "Moving Window Analysis Functions")
  lines(runmin(x,k), col=col[2])
  lines(runmed(x,k), col=col[3])
  lines(runmax(x,k), col=col[4])
  legend(0,.9*n, c("data", "runmin", "runmed", "runmax"), col=col,
lty=1 )

  #test runmean and runquantile
  y=runquantile(x, k, probs=c(0, 0.5, 1, 0.25, 0.75),
endrule="constant")
  plot(x, col=col[1], main = "Moving Window Quantile")
  lines(runmean(y[,1],k), col=col[2])
  lines(y[,2], col=col[3])
  lines(runmean(y[,3],k), col=col[4])
  lines(y[,4], col=col[5])
  lines(y[,5], col=col[6])
  lab = c("data", "runmean(runquantile(0))", "runquantile(0.5)",
  "runmean(runquantile(1))", "runquantile(.25)", "runquantile(.75)")
  legend(0,0.9*n, lab, col=col, lty=1 )

  #test runmean and runquantile
  k =25
  m=runmed(x, k)
  y=runmad(x, k, center=m)
  plot(x, col=col[1], main = "Moving Window Analysis Functions")
  lines(m    , col=col[2])
  lines(m-y/2, col=col[3])
  lines(m+y/2, col=col[4])
  lab = c("data", "runmed", "runmed-runmad/2", "runmed+runmad/2")
  legend(0,1.8*n, lab, col=col, lty=1 )

  # speed comparison
  x=runif(100000); k=991;
  system.time(runmean(x,k))
  system.time(filter(x, rep(1/k,k), sides=2)) #the fastest alternative
  k=91;
  system.time(runmad(x,k))
  system.time(apply(embed(x,k), 1, mad)) #the fastest alternative
```

3.4 base64encode & base64decode - Convert R vectors to/from the Base64 format

Description

Convert R vectors of any type to and from the Base64 format for encrypting any binary data as string using alphanumeric subset of ASCII character set.

Usage

```
  z = base64encode(x, ...)
  x = base64decode(z, what, ...)
```

Arguments

`x` vector or any structure that can be converted to a vector by `as.vector` function. Strings are also allowed.

`z` String with Base64 code, using [A-Z,a-z,0-9,+,/,=] subset of characters

`what` Either an object whose mode will give the mode of the vector to be created, or a character vector of length one describing the mode: one of '"numeric", "double", "integer", "int", "logical", "complex", "character", "raw". Same as variable `what` in `readBin` and `base64decode` functions.

`...` parameters to be passed to `bin2raw` and `raw2bin` functions.

Details

The Base64 encoding is designed to encode arbitrary binary information for transmission by electronic mail. It is defined by MIME (Multipurpose Internet Mail Extensions) specification RFC 1341, RFC 1421, RFC 2045 and others. Triplets of 8-bit octets are encoded as groups of four characters, each representing 6 bits of the source 24 bits. Only a 65-character subset ([A-Z,a-z,0-9,+,/,=]) present in all variants of ASCII and EBCDIC is used, enabling 6 bits to be represented per printable character

Value

Function `base64encode` returns a string with Base64 code. Function `base64decode` returns vector of appropriate mode and length (see `x` above).

References

- Base64 description in "Connected: An Internet Encyclopedia" http://www.freesoft.org/CIE/RFC/1521/7.htm
- MIME RFC 1341 http://www.faqs.org/rfcs/rfc1341.html
- MIME RFC 1421 http://www.faqs.org/rfcs/rfc1421.html
- MIME RFC 2045 http://www.faqs.org/rfcs/rfc2045.html
- Portions of the code are based on Matlab code by Peter Acklam http://home.online.no/~pjacklam/matlab/software/util/datautil/

See Also

- `bin2raw` and `raw2bin` are being used to convert R vectors to and from the raw binary format.
- `xmlValue` from **XML** package often reads XML code which sometimes is encoded in Base64 format.

Examples

```
x = (10*runif(10)>5) # logical
for (i in c(NA, 1, 2, 4)) {
  y = base64encode(x, size=i)
```

```
    z = base64decode(y,typeof(x), size=i)
    stopifnot(x==z)
  }
  print("Checked base64 for encode/decode logical type")

  x = as.integer(1:10) # integer
  for (i in c(NA, 1, 2, 4)) {
    y = base64encode(x, size=i)
    z = base64decode(y,typeof(x), size=i)
    stopifnot(x==z)
  }
  print("Checked base64 encode/decode for integer type")

  x = (1:10)*pi        # double
  for (i in c(NA, 4, 8)) {
    y = base64encode(x, size=i)
    z = base64decode(y,typeof(x), size=i)
    stopifnot(mean(abs(x-z))<1e-5)
  }
  print("Checked base64 for encode/decode double type")

  x = log(as.complex(-(1:10)*pi))        # complex
  y = base64encode(x)
  z = base64decode(y,typeof(x))
  stopifnot(x==z)
  print("Checked base64 for encode/decode complex type")

  x = "Chance favors the prepared mind" # character
  y = base64encode(x)
  z = base64decode(y,typeof(x))
  stopifnot(x==z)
  print("Checked base64 for encode/decode character type")
```

3.5 bin2raw & raw2bin - Convert R vectors to/from the raw binary format

Description

Convert R vectors of any type to and from the raw binary format, stored as vector of type "raw".

Usage

```
r = bin2raw(x, size=NA)
x = raw2bin(r, what, size=NA, signed = TRUE)
```

Arguments

x vector or any structure that can be converted to a vector by as.vector function. Strings are also allowed.

r vector of type "raw"

`what` Either an object whose mode will give the mode of the vector to be created, or a character vector of length one describing the mode: one of '"numeric", "double", "integer", "int", "logical", "complex", "character", "raw". Same as variable `what` in `readBin` and `base64decode` functions.

`size` integer. The number of bytes per element in the byte stream stored in `r`. The default, '`NA`', uses the natural size. See details.

`signed` logical. Only used for integers of sizes 1 and 2, when it determines if the quantity stored as raw should be regarded as a signed or unsigned integer.

Details

Quoting from `readBin` documentation:

"If 'size' is specified and not the natural size of the object, each element of the vector is coerced to an appropriate type before being written or as it is read. Possible sizes are 1, 2, 4 and possibly 8 for integer or logical vectors, and 4, 8 and possibly 12/16 for numeric vectors. (Note that coercion occurs as signed types except if 'signed = FALSE' when reading integers of sizes 1 and 2.) Changing sizes is unlikely to preserve 'NA's, and the extended precision sizes are unlikely to be portable across platforms."

Value

Function `bin2raw` returns vector of raw values (see `r` above), where each 1-byte raw value correspond to 1-byte of 1-byte of the binary form of other types. Length of the vector is going to be "number of bytes of a single element in array `x`" times `length(x)`. Function `raw2bin` returns vector of appropriate mode and length (see `x` above), where each 1-byte raw value correspond to 1-byte of the binary form of other types. Length of the vector is going to be number of bytes per element in array `x` times `length(x)`. If parameter `what` is equal to "character" than a string (of length 1) is returned instead of vector f characters.

Note

At the moment those two functions use calls to `writeBin` and `readBin` functions to do the job. I hope to change it in the future by writing a C code.

See Also

`readBin`, `writeBin`

Examples

```
x = (10*runif(10)>5) # logical
for (i in c(NA, 1, 2, 4)) {
  y = bin2raw(x, size=i)
```

```
    z = raw2bin(y,typeof(x), size=i)
    stopifnot(x==z)
  }
  print("Checked bin2raw and raw2bin conversion for logical type")

  x = as.integer(1:10) # integer
  for (i in c(NA, 1, 2, 4)) {
    y = bin2raw(x, size=i)
    z = raw2bin(y,typeof(x), size=i)
    stopifnot(x==z)
  }
  print("Checked bin2raw and raw2bin conversion for integer type")

  x = (1:10)*pi        # double
  for (i in c(NA, 4, 8)) {
    y = bin2raw(x, size=i)
    z = raw2bin(y,typeof(x), size=i)
    stopifnot(mean(abs(x-z))<1e-5)
  }
  print("Checked bin2raw and raw2bin conversion for double type")

  x = log(as.complex(-(1:10)*pi))        # complex
  y = bin2raw(x)
  z = raw2bin(y,typeof(x))
  stopifnot(x==z)
  print("Checked bin2raw and raw2bin conversion for complex type")

  x = "Chance favors the prepared mind" # character
  y = bin2raw(x)
  z = raw2bin(y,typeof(x))
  stopifnot(x==z)
  print("Checked bin2raw and raw2bin conversion for character type")
```

3.6 colAUC - Columnwise Area Under ROC Curve (AUC)

Description

Area Under ROC Curve (AUC) calculated for every column of the matrix.

Usage

```
auc = colAUC(X, y)
p   = colAUC(X, y, p.val=TRUE)
```

Arguments

`X` A matrix or data frame. Rows contain samples and columns contain features/variables.

`y` Class labels for the `X` data samples. A response vector with one label for each row/component of `X`. Can be either a factor, string or a numeric vector.

`p.val` a boolean flag: if set to TRUE than "Wilcoxon rank sum test" p-values (see wilcox.test) will be returned instead of AUC values

Details

AUC is a very useful measure of similarity between two classes measuring area under "Receiver Operating Characteristic" or ROC curve. In case of data with no ties all sections of ROC curve are either horizontal or vertical, in case of data with ties diagonal sections can also occur. Area under the ROC curve is calculated using `trapz` function. AUC is always in between 0.5 (two classes are statistically identical) and 1.0 (there is a threshold value that can achieve a perfect separation between the classes).

This measure is very similar to Wilcoxon rank sum test (see `wilcox.test`), which is also called Mann-Whitney test. Wilcoxon-test's p-value can be calculated by `p=pnorm( n1*n2*(1-auc), mean=n1*n2/2, sd=sqrt(n1*n2*(n1+n2+1)/12) )` where `n1` and `n2` are numbers of elements in two classes being compared.

The main purpose of this function was to calculate AUC's of large number of features, fast. It is being used to help with classification of protein mass spectra data that often have up to 50K features, as a fast and dirty way of lowering dimensionality of the data before applying standard classification algorithms like `nnet` or `svd`.

Value

An output is a single matrix with the same number of columns as `X` and "n choose 2" (*n!/((n-2)! 2!)*) number of rows, where n is number of unique labels in `y` list. For example, if `y` contains only two unique class labels (`length(unique(lab))==2`) than output matrix will have a single row containing AUC of each column. If more than two unique labels are present than AUC is calculated for every possible pairing of classes ("n choose 2" of them).

References

- Mason, S.J. and N.E. Graham. (2002) "Areas beneath the relative operating characteristics (ROC) and relative operating levels (ROL) curves: Statistical significance and interpretation, " Q. J. R. Meteorol. Soc. textbf{30} (1982) 291-303.
- See http://www.medicine.mcgill.ca/epidemiology/hanley/software/ to find four articles below:
    - Hanley and McNeil "The Meaning and Use of the Area under a Receiver Operating Characteristic (ROC) Curve." Radiology 1982: 143: 29-36.
    - Hanley and McNeil "A Method of Comparing the Areas under ROC curves derived from same cases." Radiology 1983: 148: 839-843.
    - McNeil and Hanley "Statistical Approaches to the Analysis of ROC curves." Medical Decision Making 1984: 4(2): 136-149.

- McNeil and Hanley "Statistical Approaches to the Analysis of ROC curves." Medical Decision Making 1984: 4(2): 136-149.

See Also

`AUC` from **ROC** package, `roc.area` from **verification** package, `wilcox.test`

Examples

```
  # load MASS library with "cats" data set that have following columns:
sex,
  # body weight, hart weight
  library(MASS)
  data(cats)
  colAUC(cats[,2:3], cats[,1])

  # compare with examples from roc.area function: using Data from Mason
and Graham (2002).
  a<- (1981:1995)
  b<- c(0,0,0,1,1,1,0,1,1,0,0,0,0,1,1)
  c<- c(.8, .8, 0, 1,1,.6, .4, .8, 0, 0, .2, 0, 0, 1,1)
  d<- c(.928,.576, .008, .944, .832, .816, .136, .584, .032, .016, .28,
.024, 0, .984, .952)
  A<- data.frame(a,b,c,d)
  names(A)<- c("year", "event", "p1", "p2")
  if (library(verification, logical.return=TRUE)) {
    roc.area(A$event, A$p1)           # for model with ties
    roc.area(A$event, A$p2)           # for model without ties
  }
  wilcox.test(p2~event, data=A)
  # colAUC output is the same as roc.area's A.tilda values
  colAUC(A[,3:4], A$event)
  # colAUC output is the same as roc.area's  and wilcox.test's p values
  colAUC(A[,3:4], A$event, p.val=TRUE)

  # example of 3-class data
  data(iris)
  colAUC(iris[,-5], iris[,5])
```

ENVI {caMassClass}   R Documentation

3.7 read.ENVI & write.ENVI - Read and write binary data in ENVI format

Description

Read and write binary data in ENVI format, which is supported by most GIS software.

Usage

```
   read.ENVI(filename, headerfile=paste(filename, ".hdr", sep=""))
  write.ENVI(X, filename, interleave = c("bsq", "bil", "bip"))
```

Arguments

`X`	data to be saved in ENVI file. Can be a matrix or 3D array.
`filename`	character string with name of the file (connection)
`headerfile`	optional character string with name of the header file
`interleave`	optional character string specifying interleave to be used

Details

ENVI binary files use a generalized raster data format that consists of two parts:

- binary file - flat binary file equivalent to memory dump, as produced by `writeBin` in R or `fwrite` in C/C++.
- header file - small text (ASCII) file containing the metadata associated with the binary file. This file can contain the following fields, followed by equal sign and a variable:
  - `samples` - number of columns
  - `lines` - number of rows
  - `bands` - number of bands (channels, planes)
  - `data type` - following types are supported:
    - 1 - 1-byte unsigned integer
    - 2 - 2-byte signed integer
    - 3 - 4-byte signed integer
    - 4 - 4-byte float
    - 5 - 8-byte double
    - 9 - 2x8-byte complex number made up from 2 doubles
    - 12 - 2-byte unsigned integer
  - `header offset` - number of bytes to skip before raster data starts in binary file.
  - `interleave` - Permutations of dimensions in binary data:
    - `BSQ` - Band Sequential (X[col,row,band])
    - `BIL` - Band Interleave by Line (X[col,band,row])
    - `BIP` - Band Interleave by Pixel (X[band,col,row])
  - `byte order` - the endian-ness of the saved data:
    - 0 - means little-endian byte order, format used on PC/Intel machines
    - 1 - means big-endian (aka IEEE, aka "network") byte order, format used on UNIX and Macintosh machines

Fields `samples`, `lines`, `bands`, `data type` are required, while `header offset`, `interleave`, `byte order` are optional. All of them are in form of integers except `interleave` which is a string.

This generic format allows reading of many raw file formats, including those with embedded header information. Also it is a handy binary format to exchange data between

PC and UNIX/Mac machines, as well as different languages like: C, Fortran, Matlab, etc. Especially since header files are simple enough to edit by hand.

File type supported by most of GIS (geographic information system) software including: ENVI software, Freelook (free file viewer by ENVI), ArcGIS, etc.

Value

Function `read.ENVI` returns either a matrix or 3D array. Function `write.ENVI` does not return anything.

See Also

`readBin`, `writeBin`

Examples

```
  X = array(1:60, 3:5)
  write.ENVI(X, "temp.nvi")
  Y = read.ENVI("temp.nvi")
  stopifnot(X == Y)
  readLines("temp.nvi.hdr")

  d = c(20,30,40)
  X = array(runif(prod(d)), d)
  write.ENVI(X, "temp.nvi", interleave="bil")
  Y = read.ENVI("temp.nvi")
  stopifnot(X == Y)
  readLines("temp.nvi.hdr")

  file.remove("temp.nvi")
  file.remove("temp.nvi.hdr")
```

3.8 trapz - Trapezoid Rule Numerical Integration

Description

Computes the integral of Y with respect to X using trapezoid rule integration.

Usage

```
trapz(x, y)
```

Arguments

`x` Sorted vector of x-axis values.

`y` Vector of y-axis values.

Details

The function has only two lines:

```
    idx = 2:length(x)
    return (as.double( (x[idx] - x[idx-1]) %*% (y[idx] + y[idx-1])) /
2)
```

Value

Integral of Y with respect to X or area under the Y curve.

Note

Trapezoid rule is not the most accurate way of calculating integrals (it is exact for linear functions), for example Simpson's rule (exact for linear and quadratic functions) is more accurate.

References

D. Kincaid & W. Chaney, "Numerical Analysis", 1991, p.445

See Also

- `intg` from **PROcess** package
- `trapezint` from **ROC** package
- `integrate`
- Matlab's `trapz` function (http://www.mathworks.com/access/helpdesk/help/techdoc/ref/trapz.html)

Examples

```
  # integral of sine function in [0, pi] range suppose to be exactly 2.
  # lets calculate it using 10 samples:
  x = (1:10)*pi/10
  trapz(x, sin(x))
  # now lets calculate it using 1000 samples:
  x = (1:1000)*pi/1000
  trapz(x, sin(x))
```

3.9 combs - All Combinations of k Elements from Vector v

Description

Finds all unordered combinations of `k` elements from vector `v`.

Usage

```
combs(v,k)
```

Arguments

`v` Any numeric vector

`k` Number of elements to choose from vector `v`. Integer smaller or equal than length of `v`.

Value

`combs(v,k)` (where `v` has length `n`) creates a matrix with *n!/((n-k)! k!)* (`n` choose `k`) rows and `k` columns containing all possible combinations of `n` elements taken `k` at a time.

See Also

I discovered recently that R packages already have two functions with similar capabilities: `combinations` from **gTools** package and `nchoosek` from **vsn** package. Also similar to Matlab's `nchoosek` function (http://www.mathworks.com/access/helpdesk/help/techdoc/ref/nchoosek.html)

Examples

```
  #example: combs(1:3,2) returns matrix with following rows (1 2), (1
3), (2 3)
  combs(1:3,2)
```

4 References

[1] Cyphergen's ProteinChip Software 3.0 User Manual.

[2] PROcess R library by Xiaochun Li
http://bioconductor.org/repository/devel/package/Source/PROcess_0.9.tar.gz

[3] University of Texas - M.D. Anderson Cancer Center – Cromwell Matlab package
http://bioinformatics.mdanderson.org/cromwell.html

[4] Petricoin EF, Ardekani AM, Hitt BA, Levine PJ, Fusaro VA, Steinberg SM, Mills GB, Simone C, Fishman DA, Kohn EC, Liotta LA: Use of proteomic patterns in serum to identify ovarian cancer. Lancet 2002, 359:572-577.

[5] Clinical Proteomics Program Databank website at www.ncifdaproteomics.com (unfortunately this website changes often).

[6] Baggerly KA, Morris JS, Coombes KR. Reproducibility of SELDI-TOF protein patterns in serum: comparing data sets from different experiments. Bioinformatics. 2004 Jan 29.

[7] Sorace JM, and Zhan, M. A data review and re-assessment of ovarian cancer serum proteomic profiling BMC Bioinformatics 2003, 4:24.

[8] Bao-Ling Adam, Yinsheng Qu, John W. Davis, Michael D. Ward, Mary Ann Clements, Lisa H. Cazares, O. John Semmes, Paul F. Schellhammer, Yutaka Yasui, Ziding Feng, and George L. Wright, Jr.. Serum Protein Fingerprinting Coupled with a Pattern-matching Algorithm Distinguishes Prostate Cancer from Benign Prostate Hyperplasia and Healthy Men Cancer Res 2002 62: 3609-3614.

[9] Bañez LL, Prasanna P, Sun L, Ali A, Zou Z, Adam B-L, McLeod DG, Moul JW and Srivastava S: Diagnostic Potential of Serum Proteomic Patterns in Prostate Cancer. J. Urol. (in press), 2003.

[10] Virginia Medical School – Virginia Prostate Center - Overview of the SELDI System; http://www.evms.edu/vpc/seldi/seldiprocess/index.html .

[11] Virginia Medical School – Virginia Prostate Center - Overview of the PeakMiner Software http://www.evms.edu/vpc/seldi/peakminer.pdf .

[12] M. Dettling & P. Bühlmann; Boosting for Tumor Classification with Gene Expression Data; Bioinformatics, June 12, 2003

[13] Yinsheng Qu, Bao-Ling Adam, Yutaka Yasui, Michael D. Ward, Lisa H. Cazares, Paul F. Schellhammer, Ziding Feng, O. John Semmes, and George L. Wright, Jr.;Boosted Decision Tree Analysis of Surface-enhanced Laser Desorption/Ionization Mass Spectral

Serum Profiles Discriminates Prostate Cancer from Noncancer Patients; Clin Chem 2002 48: 1835-1843.

[14] Wagner M, Naik DN, Pothen A, Kasukurti S, Devineni RR, Adam BL, Semmes OJ., Wright GL; Computational protein biomarker prediction: a case study for prostate cancer BMC Bioinformatics 2004, 5:26 (11 March 2004)

[15] SOM (self organizing maps) Toolbox, Matlab - http://www.cis.hut.fi/projects/somtoolbox/

[16] PRTools; pattern recognition and classification toolbox, Matlab - http://www.ph.tn.tudelft.nl/~bob/PRTOOLS.html , http://prtools.org/prtools.html

[17] rpart; Recursive partitioning and regression tree package; R - http://cran.r-project.org/src/contrib/Descriptions/rpart.html

[18] nnet, neural networks; R - http://www.math.mcgill.ca/sysdocs/R/library/nnet/html/nnet.html

[19] svm, support vector machines, R - http://www.maths.lth.se/help/R/.R/library/e1071/html/svm.html

[20] MATLAB Support Vector Machine Toolboxes - http://theoval.sys.uea.ac.uk/~gcc/svm/toolbox/ , http://asi.insa-rouen.fr/~arakotom/toolbox/index , http://www.ece.osu.edu/~maj/osu_svm/

[21] Liotta LA, Petricoin E. SELDI-TOF-based serum proteomics pattern diagnostics for early detection of cancer. Current Opinion in Biotechnology 2004, **15**:24-30

[22] Downey, Tom; With Microarrays, Pitfalls of false discovery; Genome Technology; 01/2003

[23] David G. Stork and Elad Yom-Tov; Computer Manual in MATLAB to accompany Pattern Classification; Wiley Interscience; ISBN: 0-471-42977-5