Package 'MSclassifR'

May 7, 2026

Type Package

Title Automated Classification of Mass Spectra

Version 0.5.0

Maintainer Alexandre Godmer <alexandre.godmer@aphp.fr>

Description

Functions to classify mass spectra in known categories and to determine discriminant mass-to-charge values (m/z). Includes easy-to-use preprocessing pipelines for Matrix Assisted Laser Desorption Ionisation - Time Of Flight Mass Spectrometry (MALDI-TOF) mass spectra, methods to select discriminant m/z from labelled libraries, and tools to predict categories (species, phenotypes, etc.) from selected features. Also provides utilities to build design matrices from peak intensities and labels. While this package was developed with the aim of identifying very similar species or phenotypes of bacteria from MALDI-TOF MS, the functions of this package can also be used to classify other categories associated to mass spectra; or from mass spectra obtained with other mass spectrometry techniques. Parallelized processing and optional C++-accelerated functions are available (notably to deal with large datasets) from version 0.5.0. If you use this package in your research, please cite the associated publication (<doi:10.1016/j.eswa.2025.128796>). For a comprehensive guide, additional applications, and detailed examples, see <https://github.com/agodmer/MSclassifR_examples>.

URL https://github.com/agodmer/MSclassifR_examples, https://doi.org/10.1016/j.eswa.2025.128796

BugReports https://github.com/agodmer/MSclassifR_examples/issues

License GPL (>= 3)

Encoding UTF-8

LazyData true

Language en-US

Depends R (>= 4.0), cp4p, caret, statmod, MALDIquant, MALDIrppa

Imports reshape2, ggplot2, dplyr, stats, limma, car, Rcpp, Matrix, methods

Suggests doParallel, foreach, ranger, randomForest, mixOmics, VSURF, vita, Boruta, glmnet, e1071, xgboost, nnet, mclust, mltools, metap, MALDIquantForeign, matrixStats, rmarkdown

LinkingTo Rcpp

NeedsCompilation yes

RoxygenNote 7.3.1

Author Alexandre Godmer [aut, cre],
Quentin Giai Gianetto [aut],
Karen Druart [aut]

ByteCompile true

Repository CRAN

Date/Publication 2025-12-09 11:20:01 UTC

Contents

`build_XY_from_peaks` *Build design matrix X and response Y from peak intensities*

Description

Constructs a sample-by-peak design matrix (X) and an outcome vector/factor (Y) from peak-intensity input. Accepts either a numeric matrix/data.frame (rows = samples, columns = peaks) or a list of aligned per-sample peak vectors. Optionally applies per-sample max normalization and can return X as a sparse dgCMatrix for memory efficiency.

Usage

```
build_XY_from_peaks(
  peaks,
  labels,
  normalize = c("max", "none"),
  sparse = FALSE,
  name_cols = FALSE,
  name_digits = 4
)
```

Arguments

peaks	Peak data from which to build the design matrix X. Either: • a numeric matrix/data.frame of intensities with rows = samples and columns = peaks, or • a list of per-sample numeric vectors (or small tables) aligned to a common set of peaks (e.g., same names or column order). Values are assumed non-negative; NAs are allowed and are ignored when computing per-sample maxima.
labels	Outcome/response labels used to build Y. A vector or factor with one entry per sample (length must equal nrow(peaks) for matrix/data.frame input, or length(peaks) for list input).
normalize	Normalization to apply to each sample's peak intensities before constructing X. One of "max" or "none" (matched via match.arg). "max" scales each sample by its maximum non-NA intensity; "none" applies no scaling.
sparse	Logical; if TRUE, return X as a sparse Matrix::dgCMatrix. If FALSE, return a base R dense matrix. Default is FALSE.
name_cols	Logical; if TRUE, set column names of X from the m/z values returned by MALDIquant::intensityMatrix (formatted as "mz_<mz>"). Default is FALSE. This only applies when peaks is a list of MALDIquant::MassPeaks (or when attr(X, "mass") is available); otherwise it is ignored. Enabling this can add noticeable overhead for very wide matrices.
name_digits	Integer scalar; number of decimal digits to use when formatting m/z values into column names if name_cols = TRUE. Default is 4. Must be a non-negative integer. Ignored when name_cols = FALSE or when no m/z vector is available (i.e., for plain matrix/data.frame or generic list inputs).

Value

A list with:

- X: numeric matrix or Matrix::dgCMatrix of dimension n_samples x n_peaks
- Y: response vector/factor aligned to rows of X (returned as supplied/coerced by the function)

Examples

```
data("CitrobacterRKIspectra", "CitrobacterRKImetadata", package = "MSclassifR")

spectra <- SignalProcessing(CitrobacterRKIspectra)
peaks <- MSclassifR::PeakDetection(x = spectra, averageMassSpec = FALSE)

labels <- CitrobacterRKImetadata$Species   # adjust to your label column

xy <- build_XY_from_peaks(peaks, labels, normalize = "max", sparse = TRUE)
```

`build_X_from_peaks_fast` *Build a sample-by-m/z intensity matrix from a list of peaks (fast, C++-backed)*

Description

Converts a list of MALDIquant MassPeaks into a numeric matrix X (rows = samples, columns = target m/z), by matching each target m/z to the nearest peak within a tolerance. If requested, per-row max normalization is applied. Spectra that initially produce no matches can be retried with an increased tolerance. Internally uses Rcpp for speed (binary search per m/z).

Usage

```
build_X_from_peaks_fast(
  peaks,
  moz,
  tolerance = 6,
  normalize = TRUE,
  noMatch = 0,
  bump_if_empty = TRUE,
  toleranceStep = 2,
  max_bumps = 5L
)
```

Arguments

`peaks`	List of MALDIquant::MassPeaks objects (one per sample). Each element must provide pk@mass (numeric vector of m/z) and pk@intensity (numeric vector of intensities) of the same length.
`moz`	Numeric vector of target m/z values (in Da). Will be sorted and uniqued; the output matrix columns follow this sorted order.
`tolerance`	Numeric scalar (Da). A target m/z is matched to the nearest peak only if the absolute difference is <= tolerance. Default 6.
`normalize`	Logical; if TRUE, per-row max normalization is applied after matching (i.e., each sample is divided by its maximum non-NA intensity). Default TRUE.

noMatch	Numeric scalar; intensity value to insert when no peak is matched for a given target m/z. Default 0.
bump_if_empty	Logical; if TRUE, any spectrum resulting in an all-noMatch (or all-zero after normalization) row will be retried by increasing the tolerance in steps of toleranceStep, up to max_bumps attempts. Default TRUE.
toleranceStep	Numeric scalar (Da); the increment used when bumping the tolerance for empty rows. Default 2.
max_bumps	Integer; maximum number of bumps when retrying empty rows. Default 5.

Details

- Matching: for each target m/z, the nearest peak is chosen if its distance is <= tolerance; otherwise noMatch is used. Ties are resolved by nearest distance via binary search.
- Normalization: when normalize = TRUE, each row is divided by its maximum non-NA intensity (guarded to avoid division by zero).
- Empty rows: when bump_if_empty = TRUE, rows with all noMatch (or all zeros after normalization) are retried with increased tolerance (by toleranceStep) up to max_bumps times.
- Performance: implemented with C++ helpers (map_spectrum_to_moz_cpp and build_X_from_peaks_cpp) for speed on large datasets.

Value

A numeric matrix X of dimension n x p:

- n = length(peaks)
- p = length(unique(sort(moz))) Column names are the sorted moz coerced to character. Values are intensities (possibly normalized) or noMatch for unmatched positions.

See Also

MALDIquant::createMassPeaks; internal C++ helpers map_spectrum_to_moz_cpp and build_X_from_peaks_cpp (not user-facing).

Examples

```
# Minimal example with synthetic MassPeaks
if (requireNamespace("MALDIquant", quietly = TRUE)) {
  set.seed(1)
  # Two spectra with slightly jittered peaks around 1000, 1500, 2000 Da
  mz1 <- c(999.7, 1500.2, 2000.1); int1 <- c(10, 50, 30)
  mz2 <- c(1000.3, 1499.8, 2000.4); int2 <- c(12, 60, 28)
  p1 <- MALDIquant::createMassPeaks(mass = mz1, intensity = int1)
  p2 <- MALDIquant::createMassPeaks(mass = mz2, intensity = int2)
  peaks <- list(p1, p2)

  # Target m/z grid (unsorted, will be sorted internally)
  moz <- c(2000, 1500, 1000)

  X <- build_X_from_peaks_fast(peaks, moz, tolerance = 1, normalize = TRUE)
```

```
  dim(X)
  colnames(X)
  X
}

# Typical usage in a pipeline:
# spectra <- SignalProcessing(yourSpectra)
# peaks   <- MSclassifR::PeakDetection(x = spectra, averageMassSpec = FALSE)
# moz     <- c(1000, 1500, 2000)  # from selection or prior knowledge
# X       <- build_X_from_peaks_fast(peaks, moz, tolerance = 6, normalize = TRUE)
# Then pass X to SelectionVar/SelectionVarStat_fast/LogReg, etc.
```

`calculate_distance` *Function calculating the distance between two vectors.*

Description

This function calculates the distance between two vectors using the specified distance metric. Distance metrics available are "p-norm", "Chebyshev", "Canberra", "Overlap", "HEOM" (Heterogeneous Euclidean-Overlap Metric), "HVDM" (Heterogeneous Value Difference Metric). Used in the `fast_find_neighbors` function of our package.

Usage

```
calculate_distance(x, y, nominal_indices, p_code)
```

Arguments

`x`	Vector of numeric and/or categorical values.
`y`	Vector of numeric and/or categorical values.
`nominal_indices`	Vector indicating which positions in `x` and `y` contain categorical variables.This distinction is needed because: • For categorical variables, distances are often based on whether values match or not. • Hybrid distance metrics like HEOM and HVDM require knowing which variables are nominal to apply appropriate distance calculations.
`p_code`	Numeric code representing the distance metric to use: • p >= 1: p-norm • p = 0: Chebyshev • p = -1: Canberra • p = -2: Overlap (nominal attributes only) • p = -3: HEOM (Heterogeneous Euclidean-Overlap Metric) • p = -4: HVDM (Heterogeneous Value Difference Metric)

Details

Different distance metrics handle nominal variables differently:

- For pure numeric metrics (p >= 1, p = 0, p = -1), nominal features are ignored
- For the Overlap metric (p = -2), only nominal features are considered
- For HEOM (p = -3), numeric features use normalized Euclidean distance while nominal features use overlap distance (1 if different, 0 if same)
- For HVDM (p = -4), a specialized metric combines normalized differences for numeric features and value difference metric for nominal features

Value

A numeric value representing the distance between the two vectors.

CitrobacterRKImetadata

Metadata of mass spectra corresponding to the bacterial species Citrobacter *sp. from The Robert Koch-Institute (RKI) database of microbial MALDI-TOF mass spectra*

Description

Metadada of the `CitrobacterRKIspectra` list of mass spectra.

Usage

```
data("CitrobacterRKImetadata", package = "MSclassifR")
```

Format

A data frame with 14 rows (each corresponding to a mass spectrum), and five columns that contain (in order): the strain name, the species name, the spot, a sample number and the name of the strain associated with the spot.

Details

The Robert Koch-Institute (RKI) database of microbial MALDI-TOF mass spectra contains raw mass spectra. Only mass spectra of the *Citrobacter* bacterial species were collected. Metadata were manually reported from raw data.

Source

The raw data were downloaded from this link : https://zenodo.org/record/163517#.YIkWiNZuJCp. The dataset focuses only on mass spectra from *Citrobacter*.

References

Lasch, Peter, Stammler, Maren, & Schneider, Andy. (2018). Version 3 (20181130) of the MALDI-TOF Mass Spectrometry Database for Identification and Classification of Highly Pathogenic Microorganisms from the Robert Koch-Institute (RKI) [Data set]. Zenodo.doi:10.5281/zenodo.163517

`CitrobacterRKIspectra` *Mass spectra corresponding to the bacterial species* Citrobacter *sp. from The Robert Koch-Institute (RKI) database of microbial MALDI-TOF mass spectra*

Description

Mass spectra of the `CitrobacterRKIspectra` dataset.

Usage

```
data("CitrobacterRKIspectra", package = "MSclassifR")

#####
#Plotting the first mass spectrum
#library("MSclassifR")
#PlotSpectra(SpectralData=CitrobacterRKIspectra[[1]],absx = "ALL", Peaks = NULL,
#            Peaks2 = NULL, col_spec = 1, col_peak = 2, shape_peak = 3,
#            col_peak2 = 2, shape_peak2 = 2)
```

Format

A list that contains 14 objects of class S4 corresponding each to a each mass spectrum.

Details

The Robert Koch-Institute (RKI) database of microbial MALDI-TOF mass spectra contains raw mass spectra. Only mass spectra of the *Citrobacter* bacterial species were collected.

Source

The raw data were downloaded from this link : `https://zenodo.org/record/163517#.YIkWiNZuJCp`. The dataset focuses only on mass spectra from *Citrobacter*.

References

Lasch, Peter, Stammler, Maren, & Schneider, Andy. (2018). Version 3 (20181130) of the MALDI-TOF Mass Spectrometry Database for Identification and Classification of Highly Pathogenic Microorganisms from the Robert Koch-Institute (RKI) [Data set]. Zenodo.doi:10.5281/zenodo.163517

d_left_join *Function joining two tables based not on exact matches*

Description

This function joins two tables based on a distance metric of one or more columns. It gives the same results that the `distance_left_join` function of the `fuzzyjoin` R package, but its execution time is faster. It is used to match mass spectra to short-listed mass-to-charge values in `PredictLogReg` and `PredictFastClass` functions.

Usage

```
d_left_join (x, y, by = NULL, method = "euclidean", max_dist = 1,
             distance_col = NULL)
```

Arguments

`x`	A `data.frame` object.
`y`	A `data.frame` object.
`by`	Columns by which to join the two tables. `NULL` by default (tables are joined by matching the common column names that appear in both datasets x and y).
`method`	Method to use for computing distance, either "euclidean" (default) or "manhattan".
`max_dist`	A `numeric` value indicating the maximum distance to use for joining. 1 by default.
`distance_col`	A `character` that specifies the name of a new column to be added to the output, which will contain the calculated distances between both tables. If NULL, no column is added (default).

Value

Returns a data frame that contains the results of joining the `x` dataset onto the `y` datasets, where all rows from `x` are preserved and matching data from `y` is added according to the specified criteria.

fast_cvpvi *Fast cross-validated permutation variable importance (ranger-based)*

Description

Computes cross-validated permutation variable importance (PVI) using the ranger random-forest algorithm. For each CV fold, a ranger model is trained on the training split and permutation importance is computed (OOB) inside ranger in C++. Importances are averaged across folds to obtain a stable CV importance vector. Optionally appends artificial “false” features to estimate the null distribution and pi0, then selects the top (1 - pi0) proportion of features. The evaluation can parallelize across folds (Windows-safe PSOCK) while avoiding CPU oversubscription.

Usage

```
fast_cvpvi(
  X,
  Y,
  k = 5,
  ntree = 500,
  nbf = 0,
  nthreads = max(1L, parallel::detectCores() - 1L),
  folds_parallel = c("auto", "TRUE", "FALSE"),
  mtry = NULL,
  sample_fraction = 1,
  min_node_size = 1L,
  seed = 123
)
```

Arguments

X	Numeric matrix (n x p); samples in rows, features in columns. Column names should be feature IDs (e.g., m/z). Non-finite values are set to zero internally for modeling.
Y	Factor or numeric response of length n. A factor triggers classification; numeric triggers regression.
k	Integer; number of cross-validation folds. Default 5.
ntree	Integer; number of trees per fold model. Default 500.
nbf	Integer (>= 0); number of artificial "false" (noise) features to append to X for estimating the null distribution of importances. Default 0 disables this (the null is then approximated using mirrored negative importances).
nthreads	Integer; total threads available. When parallelizing folds, each fold worker gets one ranger thread to avoid oversubscription; when not parallelizing folds, ranger uses up to nthreads threads. Default is max(1, detectCores() - 1).
folds_parallel	Character; "auto", "TRUE", or "FALSE". • "auto": parallelize across folds when k > 1 and nthreads >= 4 (default). • "TRUE": force fold-level parallelism (PSOCK cluster). • "FALSE": evaluate folds sequentially (ranger can then use multiple threads).
mtry	Optional integer; variables tried at each split. If NULL, defaults to floor(sqrt(p)) for classification or max(floor(p/3), 1) for regression.
sample_fraction	Numeric in (0, 1]; subsampling fraction per tree (speed/ regularization knob). Default 1.
min_node_size	Integer; ranger minimum node size. Larger values speed up training and yield smaller trees. Default 1.
seed	Integer; RNG seed. Default 123.

Details

- One ranger model is trained per fold (training split). Permutation importance (importance = "permutation") is computed in C++ using OOB. The per-fold importances are averaged to obtain CV importances.
- Null and pi0: if `nbf > 0`, false peaks are created to get negative importances. For this, `nbf` noise features (uniform between `min(X)` and `max(X)`) are appended and negative importances among them help shape the null. If `nbf = 0`, the null is approximated by mirroring negative importances of true features. An estimator of the proportion of useless features over high quantiles yields pi0. If no negative importances occur, pi0 is set to 0 (conservative).
- Parallelism: with `folds_parallel = "auto"/"TRUE"`, folds run in parallel using a PSOCK cluster (Windows-safe). Each worker sets ranger num.threads = 1 to avoid oversubscription. With `"FALSE"`, folds are sequential and ranger uses up to `nthreads` threads, which can be faster for small k or very large p.

Value

A list with:

- nb_to_sel: integer; number of selected features (floor(p * (1 - pi0))).
- sel_moz: character vector of selected feature names (columns of X).
- imp_sel: named numeric vector of CV importances for selected features.
- fold_varim: matrix (features x folds) of per-fold permutation importances.
- cv_varim: matrix (features x 1) of averaged importances across folds.
- pi0: estimated proportion of null features.

References

Alexandre Godmer, Yahia Benzerara, Emmanuelle Varon, Nicolas Veziris, Karen Druart, Renaud Mozet, Mariette Matondo, Alexandra Aubry, Quentin Giai Gianetto, MSclassifR: An R package for supervised classification of mass spectra with machine learning methods, Expert Systems with Applications, Volume 294, 2025, 128796, ISSN 0957-4174, doi:10.1016/j.eswa.2025.128796.

See Also

ranger::ranger; for a holdout-based (validation-fold) permutation alternative, see a custom implementation using predict() on permuted features. For a full feature-selection wrapper, see SelectionVar with MethodSelection = "cvp".

Examples

```
## Not run:
set.seed(1)
n <- 120; p <- 200
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("mz_", seq_len(p))
Y <- factor(sample(letters[1:3], n, replace = TRUE))

if (requireNamespace("ranger", quietly = TRUE)) {
```

```
  out <- fast_cvpvi(
    X, Y,
    k = 5,
    ntree = 300,
    nbf = 50,
    nthreads = max(1L, parallel::detectCores() - 1L),
    folds_parallel = "auto",
    seed = 42
  )
  head(out$sel_moz)
  # CV importances for top features
  head(sort(out$cv_varim[,1], decreasing = TRUE))
}

## End(Not run)
```

`fast_find_neighbors` *Function finding k Nearest Neighbors for each row of a matrix*

Description

This function finds the k nearest neighbors for each row in a matrix using the specified distance metric. Distance metrics available are "p-norm", "Chebyshev", "Canberra", "Overlap", "HEOM" (Heterogeneous Euclidean-Overlap Metric), "HVDM" (Heterogeneous Value Difference Metric). See the `calculate_distance` function of our package for more details on the distances.

Usage

```
fast_find_neighbors(data, nominal_indices, p_code, k)
```

Arguments

`data`	Matrix where the k nearest neighbors for each row are searched.
`nominal_indices`	Vector of column indices indicating which features are categorical (nominal) variables. This is crucial for proper distance calculation as nominal and numeric features require different handling. For example, if columns 2 and 5 contain categorical variables, nominal_indices should be `c(2, 5)`. See `calculate_distance` function.
`p_code`	Numeric code representing the distance metric to use: • p >= 1: p-norm • p = 0: Chebyshev • p = -1: Canberra • p = -2: Overlap (nominal attributes only) • p = -3: HEOM (Heterogeneous Euclidean-Overlap Metric) • p = -4: HVDM (Heterogeneous Value Difference Metric)
`k`	Number of nearest neighbors to find.

Value

A matrix where each row contains the indices of the k nearest neighbors for the corresponding example.

See Also

calculate_distance

`fast_generate_synthetic` *Function generating synthetic examples using SMOTE*

Description

This function generates synthetic examples using the SMOTE algorithm. For each example in the minority class, this function generates synthetic examples by interpolating between the example and its k nearest neighbors. This function is used in the `smote_classif` function.

Usage

```
fast_generate_synthetic(dat, k, n, p_code)
```

Arguments

`dat`	A `data.frame` object.
`k`	Number of nearest neighbors to consider.
`n`	Number of synthetic examples to generate.
`p_code`	Numeric code representing the distance metric to use: • p >= 1: p-norm • p = 0: Chebyshev • p = -1: Canberra • p = -2: Overlap (nominal attributes only) • p = -3: HEOM (Heterogeneous Euclidean-Overlap Metric) • p = -4: HVDM (Heterogeneous Value Difference Metric)

Value

A data frame containing the generated synthetic examples.

See Also

smote_classif

`fast_mda` *Fast MDA-style variable selection using ranger permutation importance*

Description

Computes feature importances with a single multiclass (or regression) random forest using the ranger engine and its C++ permutation importance. The null distribution of importances is estimated either by appending artificial "false" (noise) features (when `nbf > 0`) or by mirroring negative importances (when `nbf = 0`). An estimator of the proportion of useless features yields pi0, and the top (1 - pi0) proportion of true features are selected. This implementation is a fast, OS-agnostic alternative to repeated/random-forest-based MDA schemes.

Usage

```
fast_mda(
  X,
  Y,
  ntree = 1000,
  nbf = 0,
  nthreads = max(1L, parallel::detectCores() - 1L),
  mtry = NULL,
  sample_fraction = 1,
  min_node_size = 1L,
  seed = 123
)
```

Arguments

`X`	Numeric matrix (n x p); samples in rows, features in columns. Column names should be feature IDs (e.g., m/z). Non-finite values are set to zero internally for modeling.
`Y`	Factor (classification) or numeric (regression) response of length n. The default `mtry` is chosen based on the task: floor(sqrt(p)) for classification; max(floor(p/3), 1) for regression.
`ntree`	Integer; number of trees. Default 1000.
`nbf`	Integer (>= 0); number of artificial "false" (noise) features to append to X to estimate the null distribution. Default 0 disables this and uses mirrored negative importances as the null.
`nthreads`	Integer; total number of threads for ranger. Default is max(1, parallel::detectCores() - 1).
`mtry`	Optional integer; variables tried at each split. If NULL (default), computed as floor(sqrt(p)) for classification or max(floor(p/3), 1) for regression.
`sample_fraction`	Numeric in (0, 1]; subsampling fraction per tree (speed/ regularization knob). Default 1.

`min_node_size`	Integer; ranger minimum node size. Larger values speed up training and yield simpler trees. Default 1.
`seed`	Integer; RNG seed for reproducibility. Default 123.

Details

- A single ranger model is fit with importance = "permutation". This computes permutation importance in C++ using OOB (fast and stable).
- Null and pi0:
  - If `nbf` > `0`, `nbf` false features (uniform between `min(X)` and `max(X)`) are appended; negative importances among them help shape the null. An estimator of the proportion of useless features over high quantiles (e.g., 0.75–1) yields pi0 and is adjusted for the number of false features.
  - If `nbf` = `0`, the null is approximated by mirroring negative importances of true features. If no negative importances occur, pi0 is set to 0 (conservative).
- Task: factors in `Y` trigger probability = TRUE; numeric `Y` triggers regression.
- Robustness: any non-finite importances are set to zero. Selection is performed only among the original (true) features; false features are discarded.
- Performance: this is typically 5–20x faster than randomForest-based MDA and fully multithreaded via `nthreads`.

Value

A list with:

- nb_to_sel: integer; number of selected features (floor(p_true * (1 - pi0))).
- sel_moz: character vector of selected feature names (columns of X).
- imp_sel: named numeric vector of importances for selected features (true features only).
- all_imp: named numeric vector of importances for all true features.
- pi0: estimated proportion of null features.

References

Alexandre Godmer, Yahia Benzerara, Emmanuelle Varon, Nicolas Veziris, Karen Druart, Renaud Mozet, Mariette Matondo, Alexandra Aubry, Quentin Giai Gianetto, MSclassifR: An R package for supervised classification of mass spectra with machine learning methods, Expert Systems with Applications, Volume 294, 2025, 128796, ISSN 0957-4174, doi:10.1016/j.eswa.2025.128796.

See Also

ranger::ranger; for cross-validated permutation importance, see fast_cvpvi. For a wrapper that plugs MDA/CVP into broader selection workflows, see SelectionVar (MethodSelection = "mda" or "cvp").

Examples

```
## Not run:
set.seed(1)
n <- 100; p <- 300
X <- matrix(rnorm(n * p), n, p)
colnames(X) <- paste0("mz_", seq_len(p))
Y <- factor(sample(letters[1:3], n, replace = TRUE))

if (requireNamespace("ranger", quietly = TRUE)) {
  out <- fast_mda(
    X, Y,
    ntree = 500,
    nbf = 50,
    nthreads = max(1L, parallel::detectCores() - 1L),
    seed = 42
  )
  out$nb_to_sel
  head(out$sel_moz)
  # Top importances
  head(sort(out$all_imp, decreasing = TRUE))
}

## End(Not run)
```

LogReg *Fast supervised classifier with m/z subsetting and optional sampling*

Description

Trains a multiclass classifier on a subset of m/z features using cross-validation. For kind = "rf", it automatically delegates to a ranger-based algorithm (LogReg_rf_fast) when available for maximum speed and parallelism; otherwise it uses the caret R package with method = "ranger" as a fast fallback. Other kinds ("linear", "nnet", "svm", "xgb") are trained via caret with compact grids and optional parallelization. Features (columns) are selected by matching their numeric column names to moz. Optional class-balancing (among up/down-sampling or SMOTE) can be applied.

Usage

```
LogReg(
  X,
  moz,
  Y,
  number = 2,
  repeats = 2,
  Metric = c("Kappa", "Accuracy", "F1", "AdjRankIndex", "MatthewsCorrelation"),
  kind = "linear",
  Sampling = c("no", "up", "down", "smote"),
```

```
  ncores = max(1L, parallel::detectCores() - 1L),
  num.trees = 500L,
  tuneLength = 5L,
  seed = 123L
)
```

Arguments

X	Numeric matrix or data.frame with samples in rows and features (m/z) in columns. Column names must be numeric (or coercible), e.g., "1234.567" or "mz_1234.567". Non-finite values are set to 0.
moz	Numeric vector of m/z values to keep. Only columns of X whose numeric names match values in moz are used. An error is thrown if none match.
Y	Factor (or coercible) of class labels; length must equal nrow(X).
number	Integer; number of CV folds (k). Default 2.
repeats	Integer; number of CV repeats. Default 2.
Metric	Character; selection metric. One of "Kappa", "Accuracy", "F1", "AdjRankIndex", "MatthewsCorrelation". For non-caret metrics, custom summary functions are used.
kind	Character; model type. One of "linear" (multinom), "nnet" (nnet), "rf" (random forest), "svm" (svmLinear2), "xgb" (xgbTree). Default "linear".
Sampling	Character; class-balancing strategy. One of "no", "up", "down", "smote". For "smote", the function smote_classif(Y ~ ., data.frame(Y, X)) is used before training. For "up"/"down", caret's in-fold sampling is used.
ncores	Integer; number of CPU cores to use for caret's parallel backend (doParallel). Default is all but one core. Ignored if doParallel is unavailable.
num.trees	Integer; number of trees for random forests (ranger engine). Default 500. Used when kind = "rf" and either the caret "ranger" fallback is used or the caret-free LogReg_rf_fast is available.
tuneLength	Integer; size of the hyperparameter search (caret-based models). Default 5 (compact grid).
seed	Integer; random seed for reproducibility. Default 123.

Details

- Feature subsetting: X is subset to columns whose numeric names match moz. This avoids expensive joins/transposes and guarantees stable feature order.
- Random forests: if the function LogReg_rf_fast is available in the namespace (see its documentation), this function delegates the "rf" case to it for maximum speed and Windows-friendly parallel CV. Otherwise, it uses caret with method = "ranger" (still fast and parallelizable).
- Sampling: "smote" is applied once, before training; "up"/"down" are applied in-fold by caret via trainControl(sampling = ...). "no" leaves the data unchanged.
- Parallelism: if ncores > 1 and doParallel is installed, a PSOCK cluster is registered for caret. The fast RF engine (LogReg_rf_fast) internally handles fold-level parallelism and ranger threading to avoid oversubscription.

Value

A list with:

- train_mod: the fitted model (caret::train object) or, if kind = "rf" and LogReg_rf_fast is available, the structure returned by LogReg_rf_fast (contains the final ranger model and CV details).
- boxplot: ggplot object of resampling metric distributions (caret paths) or the boxplot returned by LogReg_rf_fast.
- Confusion.Matrix: caret::confusionMatrix on the fitted model (caret paths) or the confusion matrix returned by LogReg_rf_fast.
- stats_global: data.frame summarizing per-fold metrics (Metric, Mean, Sd) for caret paths; from LogReg_rf_fast otherwise.

See Also

LogReg_rf_fast, ranger::ranger, caret::train, caret::confusionMatrix

Examples

```
## Not run:
set.seed(1)
X <- matrix(runif(2000), nrow = 100, ncol = 20)
colnames(X) <- as.character(round(seq(1000, 1190, length.out = 20), 4))
moz <- as.numeric(colnames(X))[seq(1, 20, by = 2)]
Y <- factor(sample(letters[1:3], 100, replace = TRUE))

# Fast RF (delegates to LogReg_rf_fast if available; else caret + ranger)
fit_rf <- LogReg(X, moz, Y, number = 3, repeats = 1, kind = "rf",
                 Metric = "Kappa", Sampling = "no", ncores = 4,
                 num.trees = 300, seed = 42)
fit_rf$Confusion.Matrix

# Linear (multinom) with macro F1 metric
fit_lin <- LogReg(X, moz, Y, number = 3, repeats = 1, kind = "linear",
                  Metric = "F1", Sampling = "no", ncores = 2)
fit_lin$stats_global

## End(Not run)
```

`LogReg_rf_fast` *Fast random-forest classifier with stratified CV and in-fold sampling (ranger, caret-free)*

Description

Trains a multiclass random-forest classifier using the ranger algorithm with a compact hyperparameter search and repeated stratified cross-validation. Feature columns are first subset by the provided m/z list (moz). Class balancing (no/up/down/SMOTE) is applied only within training folds to avoid leakage, and again on the full data before fitting the final model. The evaluation across folds can be parallelized in a Windows-safe manner (PSOCK), while avoiding CPU oversubscription by giving each fold worker one ranger thread. Returns the final ranger model, per-fold metrics, a confusion matrix on the full data, and a ggplot boxplot of resampling metrics.

Usage

```
LogReg_rf_fast(
  X,
  moz,
  Y,
  number = 5,
  repeats = 1,
  Metric = c("Kappa", "Accuracy", "F1", "AdjRankIndex", "MatthewsCorrelation"),
  Sampling = c("no", "up", "down", "smote"),
  ncores = max(1L, parallel::detectCores() - 1L),
  num.trees = 500L,
  tuneLength = 5L,
  folds_parallel = c("auto", "TRUE", "FALSE"),
  seed = 123L,
  mtry = NULL,
  splitrule = "gini",
  sample.fraction = 1,
  min.node.size.grid = c(1L, 5L, 10L),
  min_node_frac = 1/3
)
```

Arguments

X	Numeric matrix or data frame; rows are samples and columns are features (m/z). Column names must be numeric (coercible with as.numeric), representing the feature m/z. Non-finite values are set to 0 internally.
moz	Numeric vector of m/z to keep. Only columns of X whose numeric names match values in moz are used. An error is raised if none match.
Y	Factor (or coercible to factor) of class labels; length must equal nrow(X).
number	Integer; number of CV folds (k). Default 5.
repeats	Integer; number of CV repeats. Default 1.
Metric	Character; CV selection metric. One of "Kappa", "Accuracy", "F1", "AdjRankIndex", "MatthewsCorrelation". The best hyperparameters maximize this metric averaged over folds.
Sampling	Character; class-balancing strategy applied within each training fold (and before the final fit on the full data). One of "no", "up", "down", "smote".

- "up": up-samples minority classes to the majority count (base R).
- "down": down-samples majority classes to the minority count (base R).
- "smote": uses the package's internal smote_classif(Y ~ ., data.frame(Y, X), C.perc = "balance").

`ncores` Integer; number of CPU cores to use. Controls both fold-level parallelism and ranger threads when not parallelizing folds. Default is all but one core.

`num.trees` Integer; number of trees per ranger model. Default 500.

`tuneLength` Integer; upper bound on the size of the hyperparameter grid. If the full grid (mtry × min.node.size) is larger, a random subset of size `tuneLength` is used. Default 5.

`folds_parallel` Character; "auto", "TRUE", or "FALSE".

- "auto": parallelize across folds when ncores >= 2 and total folds (number × repeats) >= 2.
- "TRUE": force fold-level parallelism (PSOCK on Windows).
- "FALSE": evaluate folds sequentially; ranger then uses up to `ncores` threads per fit.

`seed` Integer; RNG seed for reproducibility. Default 123.

`mtry` Optional integer; if provided, fixes the number of variables tried at each split. If NULL (default), a small grid around floor(sqrt(p)) is used, where p = number of features.

`splitrule` Character; ranger split rule (e.g., "gini", "extratrees"). Default "gini".

`sample.fraction`
Numeric in (0, 1]; subsampling fraction per tree in ranger. Default 1.

`min.node.size.grid`
Integer vector; candidate values for ranger's `min.node.size` used to build the tuning grid. Default c(1, 5, 10).

`min_node_frac` Numeric in (0, 1]. Safety cap for ranger's min.node.size per fold/final fit: the value used is min(requested_min.node.size, floor(min_node_frac * n_train)), with a lower bound of 1. This prevents root-only trees (near-uniform class probabilities) on small training folds (e.g., with SMOTE). Applied inside CV and for the final model. Default: 1/3 (set to 1 to disable capping).

Details

- Feature subsetting: X is subset to columns whose numeric names match `moz`. This avoids expensive joins/transposes and guarantees consistent feature order.
- Cross-validation: folds are stratified by Y and repeated `repeats` times. Sampling is applied only to training indices in each fold (to prevent leakage) and again before the final fit.
- Hyperparameter search: a compact grid over mtry (around sqrt(p)) and min.node.size (from `min.node.size.grid`), optionally downsampled to `tuneLength`. The best combination maximizes the chosen metric averaged over folds.
- Parallel strategy: by default ("auto"), the code parallelizes across folds with a PSOCK cluster (Windows-safe) and sets ranger's num.threads = 1 inside each worker to avoid oversubscription. If you set folds_parallel = "FALSE", folds run sequentially and each ranger fit uses up to `ncores` threads for strong single-fit parallelism.

- Metrics:
  - Accuracy and Cohen's Kappa computed from the confusion matrix.
  - F1 is macro-averaged across classes.
  - AdjRankIndex uses mclust::adjustedRandIndex.
  - MatthewsCorrelation is the multiclass MCC.

Value

A list with:

- train_mod: list with fields
  - model: the fitted ranger::ranger object (final model on full data)
  - method: "ranger"
  - best_params: data.frame with the best hyperparameters found by CV
  - cv_score: best mean CV score (according to `Metric`)
  - metric: the metric name used
- boxplot: ggplot object showing the distribution of per-fold metric values
- Confusion.Matrix: caret::confusionMatrix for predictions of the final model on the full data
- stats_global: data.frame with columns Metric, Mean, Sd summarizing per-fold metrics
- resample: data.frame of per-fold metrics (columns: variable, value, fold)

See Also

ranger::ranger, caret::confusionMatrix

Examples

```
## Not run:
set.seed(1)
X <- matrix(runif(3000), nrow = 100, ncol = 30)
colnames(X) <- as.character(round(seq(1000, 1290, length.out = 30), 4))
moz <- as.numeric(colnames(X))[seq(1, 30, by = 2)]  # keep half the m/z
Y <- factor(sample(letters[1:3], 100, replace = TRUE))

fit <- LogReg_rf_fast(
  X, moz, Y,
  number = 3, repeats = 1,
  Metric = "Kappa",
  Sampling = "no",
  ncores = 4,
  num.trees = 300,
  tuneLength = 4,
  seed = 42
)
fit$train_mod$best_params
fit$Confusion.Matrix

## End(Not run)
```

MSclassifR *Automated classification of mass spectra*

Description

This package provides R functions to classify mass spectra in known categories, and to determine discriminant mass-to-charge values. It was developed with the aim of identifying very similar species or phenotypes of bacteria from mass spectra obtained by Matrix Assisted Laser Desorption Ionisation - Time Of Flight Mass Spectrometry (MALDI-TOF MS). However, the different functions of this package can also be used to classify other categories associated to mass spectra; or from mass spectra obtained with other mass spectrometry techniques. It includes easy-to-use functions for pre-processing mass spectra, functions to determine discriminant mass-to-charge values (m/z) from a library of mass spectra corresponding to different categories, and functions to predict the category (species, phenotypes, etc.) associated to a mass spectrum from a list of selected mass-to-charge values.

If you use this package in your research, please cite the associated publication: doi:10.1016/j.eswa.2025.128796.

For a comprehensive guide, additional applications, and detailed examples of using this package, please visit our GitHub repository: here.

Value

No return value. Package description.

Author(s)

Alexandre Godmer, Quentin Giai Gianetto

References

Alexandre Godmer, Yahia Benzerara, Emmanuelle Varon, Nicolas Veziris, Karen Druart, Renaud Mozet, Mariette Matondo, Alexandra Aubry, Quentin Giai Gianetto, MSclassifR: An R package for supervised classification of mass spectra with machine learning methods, Expert Systems with Applications, Volume 294, 2025, 128796, ISSN 0957-4174, doi:10.1016/j.eswa.2025.128796.

PeakDetection *Detection of peaks in MassSpectrum objects*

Description

Detects peaks on a list of MALDIquant MassSpectrum objects, with an optional preliminary averaging step and an optional discrete-bin alignment of detected peaks. Per-spectrum peak detection can be parallelized on Unix-alike systems (Linux/macOS) for very large inputs; on Windows a serial/vectorized path is used. The result is a list of MALDIquant MassPeaks ready for downstream matrix building (e.g., with MALDIquant::intensityMatrix or build_X_from_peaks_fast).

Usage

```
PeakDetection(
  x,
  averageMassSpec = TRUE,
  labels = NULL,
  averageMassSpectraMethod = "median",
  SNRdetection = 3,
  binPeaks = TRUE,
  PeakDetectionMethod = "MAD",
  halfWindowSizeDetection = 11,
  AlignMethod = "strict",
  Tolerance = 0.002,
  n_workers = NULL,
  verbose = TRUE,
  min_parallel_n = 2000L,
  chunk_size = 1000L,
  ...
)
```

Arguments

x	List of MALDIquant::MassSpectrum objects (one per sample). These are typically obtained after preprocessing (baseline, smoothing, normalization).
averageMassSpec	Logical; if TRUE, average spectra using MALDIquant::averageMassSpectra before peak detection. If labels is provided and its length equals length(x), a groupwise averaging is performed; otherwise all spectra are averaged. Default TRUE.
labels	Optional factor/character vector for groupwise averaging (same semantics as MALDIquant::averageMassSpectra). Ignored if averageMassSpec = FALSE.
averageMassSpectraMethod	Character, "median" (default) or "mean". Passed to MALDIquant::averageMassSpectra when averageMassSpec = TRUE.
SNRdetection	Numeric; signal-to-noise ratio threshold for peak detection (MALDIquant::detectPeaks argument SNR). Default 3.
binPeaks	Logical; if TRUE, align detected peaks into discrete bins using MALDIquant::binPeaks (method/tolerance set by AlignMethod/Tolerance). Default TRUE.
PeakDetectionMethod	Character; MALDIquant::detectPeaks method, e.g., "MAD" (default) or "SuperSmoother".
halfWindowSizeDetection	Integer; half window size for local maxima (MALDIquant::detectPeaks argument halfWindowSize). Default 11.
AlignMethod	Character; MALDIquant::binPeaks method, "strict" (default) or "relaxed".
Tolerance	Numeric; MALDIquant::binPeaks tolerance (units consistent with your m/z axis). Default 0.002.

n_workers	Integer or NULL; requested number of parallel workers for the Unix mclapply path. On Windows, this is ignored (serial path). The effective number is sanitized by an internal helper (.safe_n_workers) to avoid oversubscription and R CMD check issues. Default NULL (auto).
verbose	Logical; if TRUE, print progress messages. Default TRUE.
min_parallel_n	Integer; minimum number of spectra at which the function will attempt Unix parallelization via mclapply. Defaults to 2000. Increase to be more conservative, decrease to parallelize more aggressively. Set to Inf to effectively disable Unix parallelization regardless of n_workers.
chunk_size	Integer; number of spectra per chunk/task submitted to mclapply on Unix. Larger chunks reduce scheduling overhead but use more memory per task. Default 1000.
...	Reserved for future extensions or pass-through to MALDIquant/MALDIrppa.

Details

- Averaging: if `averageMassSpec = TRUE` and `labels` is provided with length(labels) == length(x), MALDIquant::averageMassSpectra performs a groupwise averaging by labels. Otherwise, all spectra are averaged. If `averageMassSpec = FALSE`, the input list is used as-is.
- Peak detection: peak finding uses MALDIquant::detectPeaks with the given SNR/method/halfwindow. This is applied per spectrum (serial or parallel).
- Discrete-bin alignment: when `binPeaks = TRUE`, MALDIquant::binPeaks aligns detected peaks to a shared discrete grid (method `AlignMethod`, tolerance `Tolerance`), enabling consistent feature columns across spectra.
- Parallelization: on Windows, a single serial/vectorized call to MALDIquant::detectPeaks is used (fast enough for small/medium inputs). On Unix-alike systems, when `length(x) >= min_parallel_n` and `n_workers > 1`, the list is split into chunks of size `chunk_size` and processed with parallel::mclapply using the requested number of workers.
- Meta-data: if `labels` is provided, label information is appended best-effort to MassPeaks metaData (file/fullName), preserving existing fields where possible.

Value

A list of MALDIquant::MassPeaks objects (one per input or averaged spectrum). If `binPeaks = TRUE`, all MassPeaks are aligned to the same discrete m/z bins (shared centers), facilitating fast matrix construction.

See Also

MALDIquant::averageMassSpectra, MALDIquant::detectPeaks, MALDIquant::binPeaks; MALDIquant::intensityMatrix; build_X_from_peaks_fast for a fast matrix builder from MassPeaks.

Examples

```
if (requireNamespace("MALDIquant", quietly = TRUE)) {
  # Two toy spectra with peaks near 1000, 1500, 2000 Da
  mass <- seq(900, 2100, by = 1)
```

```
  make_spectrum <- function(shift) {
    inten <- dnorm(mass, 1000 + shift, 2) * 50 +
             dnorm(mass, 1500 - shift, 2) * 80 +
             dnorm(mass, 2000 + shift, 2) * 40 +
             rnorm(length(mass), 0, 0.2)
    MALDIquant::createMassSpectrum(mass = mass, intensity = inten)
  }
  spectra <- list(make_spectrum(0.3), make_spectrum(-0.3))

  # Detect peaks without averaging; align in strict bins
  peaks <- PeakDetection(
    x = spectra,
    averageMassSpec = FALSE,
    SNRdetection = 3,
    PeakDetectionMethod = "MAD",
    binPeaks = TRUE,
    AlignMethod = "strict",
    Tolerance = 0.5,
    verbose = TRUE
  )

  # Build an intensity matrix (rows = spectra, cols = aligned m/z bins)
  X <- MALDIquant::intensityMatrix(peaks)
  dim(X)
}
```

`PlotSpectra` *Plot spectral data with optional peak markers*

Description

Create a ggplot of a mass spectrum with optional points for detected peaks and optional vertical lines (and points) highlighting user-specified m/z values.

Usage

```
PlotSpectra(
  SpectralData,
  absx = "ALL",
  Peaks = NULL,
  Peaks2 = NULL,
  col_spec = 1,
  col_peak = 2,
  shape_peak = 3,
  col_peak2 = 2,
  shape_peak2 = 2,
  tol = 0
)
```

Arguments

SpectralData	An object containing spectrum data with numeric slots @mass (m/z) and @intensity (signal). Typically a MALDIquant MassSpectrum-like S4 object.
absx	Either the string "ALL" to plot the full m/z range, or a numeric length-2 vector c(min, max) specifying the m/z window to display.
Peaks	Optional MassPeaks object (e.g., from MALDIquant) providing detected peak positions (@mass) and intensities (@intensity) to plot as points.
Peaks2	Optional numeric or character vector of m/z values to highlight. If character/factor, values are coerced to numeric (non-numeric chars removed). Vertical dashed lines are drawn at these m/z. If Peaks is also supplied, points are plotted for peaks whose m/z match Peaks2 (within tol).
col_spec	Colour for the spectrum line. Default: 1 (black).
col_peak	Colour for points corresponding to all peaks in Peaks. Default: 2 (red).
shape_peak	Point shape for Peaks points. Default: 3.
col_peak2	Colour for points corresponding to the subset of Peaks that match Peaks2. Default: 2 (red).
shape_peak2	Point shape for the Peaks2-matched points. Default: 2.
tol	Numeric tolerance (in m/z units) used to match Peaks@mass to Peaks2. Set to 0 for exact matching (may miss due to floating-point precision). Default: 0.

Value

A ggplot object representing the spectrum and optional annotations.

Examples

```
if (requireNamespace("MALDIquant", quietly = TRUE)) {
# Load mass spectra
 data("CitrobacterRKIspectra", package = "MSclassifR")
# Plot raw mass spectrum
 PlotSpectra(SpectralData = CitrobacterRKIspectra[[1]])
# standard pre-processing of mass spectra
 spectra <- SignalProcessing(CitrobacterRKIspectra)
# Plot pre-processed mass spectrum
 PlotSpectra(SpectralData=spectra[[1]])
# detection of peaks in pre-processed mass spectra
 peaks <- PeakDetection(x = spectra, averageMassSpec=FALSE)
# Plot peaks on pre-processed mass spectrum
 PlotSpectra(SpectralData=spectra[[1]],Peaks=peaks[[1]],col_spec="blue",col_peak="black")
}
```

PredictFastClass *Fast class prediction from peak lists using linear regressions*

Description

Builds a sample-by-m/z matrix from a list of MALDIquant MassPeaks and predicts the class of each spectrum by fitting, for each class, a linear regression of the spectrum's intensities on the training spectra of that class. The class minimizing the AIC is selected as the predicted label. In parallel, an F-test p-value is computed per class to quantify how unlikely the spectrum is to belong to the training database; the minimum across classes is returned as `p_not_in_DB`. The peak-to-m/z matching is done in C++ via `build_X_from_peaks_fast()` for speed.

Usage

```
PredictFastClass(
  peaks,
  mod_peaks,
  Y_mod_peaks,
  moz = "ALL",
  tolerance = 6,
  normalizeFun = TRUE,
  noMatch = 0,
  chunk_size = 2000L,
  ncores = 1L,
  verbose = FALSE
)
```

Arguments

`peaks`	List of MALDIquant::MassPeaks objects to classify (one per spectrum). Each element must expose `@mass` (numeric m/z) and `@intensity` (numeric) of the same length. Names/metaData are used to populate the `name` column.
`mod_peaks`	Numeric training matrix of dimension n_train x p (rows = spectra, columns = m/z features) used as regressors per class. Column names must be m/z values (character) and must include all m/z requested in `moz`.
`Y_mod_peaks`	Factor of length n_train giving the class labels for rows of `mod_peaks`.
`moz`	Either "ALL" or a numeric vector of target m/z. If "ALL" (default), the column names of `mod_peaks` are used. Otherwise, the provided m/z are used (they must all be present among the column names of `mod_peaks`).
`tolerance`	Numeric (Da). A target m/z is matched to the nearest peak only if the absolute difference is <= `tolerance`. Default 6.
`normalizeFun`	Logical; if TRUE, per-spectrum max normalization is applied after matching (i.e., each row of the new matrix is divided by its maximum). Default TRUE.
`noMatch`	Numeric; intensity value inserted when no peak is matched for a given target m/z. Default 0.

chunk_size	Integer; rows per block when building the new matrix from peaks (passed to build_X_from_peaks_fast(), if used). Default 2000.
ncores	Integer; number of cores to use when building the new matrix from peaks (R side). Default 1.
verbose	Logical; print progress messages. Default FALSE.

Details

- Matrix building: build_X_from_peaks_fast() maps each spectrum in peaks to the target m/z grid with nearest-within-tolerance matching (C++). If normalizeFun = TRUE, each row is divided by its maximum (guarded to avoid division by zero). Spectra with initially no matches are retried with a slightly increased tolerance (internal bumping).
- Alignment to training: columns of the new matrix must align to mod_peaks. The function stops if any requested m/z is missing from mod_peaks.
- Per-class regression: for each class k, it regresses the new spectrum's intensities on the columns of mod_peaks belonging to class k (after removing entries where the new spectrum is non-finite). If the number of training spectra exceeds the number of non-missing points in the spectrum, a random subset of columns (size = length(non-missing) - 1) is used to avoid singular fits. Fitting is done via stats::lm.fit for speed.
- Selection and scores: pred_cat is the class with smallest AIC across fitted models. For each class, an F-test p-value is computed from the model summary; p_not_in_DB is the minimum across classes (1 if a class model fails).

Value

A data.frame with columns:

- name: spectrum name (from MassPeaks metaData fullName/file if available).
- p_not_in_DB: minimum F-test p-value across classes (smaller suggests the spectrum matches the training database; larger suggests "not in DB").
- pred_cat: predicted class (label with smallest AIC).

See Also

build_X_from_peaks_fast; MALDIquant::createMassPeaks; stats::lm.fit

Examples

```
## Not run:
if (requireNamespace("MALDIquant", quietly = TRUE)) {
  set.seed(1)
  # Create a small training set (mod_peaks) with 2 classes
  p <- 6
  moz <- as.character(round(seq(1000, 1500, length.out = p), 2))
  mod_peaks <- rbind(
    matrix(runif(5 * p, 0, 1), nrow = 5, dimnames = list(NULL, moz)),
    matrix(runif(5 * p, 0, 1), nrow = 5, dimnames = list(NULL, moz))
  )
```

```
  Y_mod <- factor(rep(c("A", "B"), each = 5))

  # Two spectra to classify: generate MassPeaks near moz
  mk_peaks <- function(shift = 0) {
    MALDIquant::createMassPeaks(
      mass = as.numeric(moz) + rnorm(length(moz), shift, 0.2),
      intensity = runif(length(moz), 10, 100)
    )
  }
  peaks <- list(mk_peaks(0.1), mk_peaks(-0.1))

  res <- PredictFastClass(
    peaks = peaks,
    mod_peaks = mod_peaks,
    Y_mod_peaks = Y_mod,
    moz = "ALL",
    tolerance = 1,
    normalizeFun = TRUE
  )
  res
}

## End(Not run)
```

PredictLogReg *Prediction of the category to which a mass spectrum belongs*

Description

Predicts the category (species, phenotype, etc.) of each spectrum in a list of MALDIquant MassPeaks using one or more trained models (e.g., multinomial logistic regression from LogReg). Peaks are matched to a given shortlist of discriminant m/z values (moz) within a tolerance; unmatched positions are filled with `noMatch`. If several models are supplied, the function also produces meta-predictions: per-class Fisher combinations (if 'metap' is available) and a majority-vote fraction across models.

Usage

```
PredictLogReg(
  peaks,
  model,
  moz,
  tolerance = 6,
  toleranceStep = 2,
  normalizeFun = TRUE,
  noMatch = 0,
  Reference = NULL,
  chunk_size = 10000L,
```

```
  ncores = 1L,
  verbose = FALSE
)
```

Arguments

peaks	a list of MALDIquant::MassPeaks objects (one per spectrum).
model	a model or a list of models estimated from a shortlist of m/z (e.g., the output of LogReg). Each model must support predict(..., type = "prob"). If a single model is supplied, it is wrapped into a one-element list.
moz	a numeric vector of shortlisted m/z values used for prediction (typically the selection from SelectionVar/SelectionVarStat_fast).
tolerance	numeric; accepted m/z tolerance (in Da) for matching peaks to moz. Default 6.
toleranceStep	numeric; if a spectrum yields no matches at the initial tolerance, the function retries by increasing tolerance in steps of toleranceStep until at least one match is found (bounded internally). Default 2.
normalizeFun	logical; if TRUE (default), per-spectrum max normalization is applied after matching (row is divided by its maximum).
noMatch	numeric; intensity used when no peak matches a given m/z. Default 0.
Reference	optional factor of true categories, length equal to length(peaks). If provided and has at least two distinct levels, the function returns, in addition to the predictions, per-model confusion matrices (caret::confusionMatrix).
chunk_size	integer; number of spectra per prediction batch (rows of X). Large datasets can be processed in chunks to limit memory. Default 10000.
ncores	integer; number of cores used while building X from peaks on the R side (the C++ matching itself is single-threaded here). Default 1.
verbose	logical; print progress messages. Default FALSE.

Details

- Matching and normalization: peak-to-moz matching is performed by build_X_from_peaks_fast (C++-backed; nearest-within-tolerance). If no m/z from a spectrum match the shortlist initially, the tolerance is increased by toleranceStep in a small number of attempts until at least one match is found. If normalizeFun = TRUE, each row is divided by its maximum (guarded to avoid divide-by-zero).
- Multiple models: when several models are supplied, the output contains one set of probabilities per model (with method column identifying it). Two additional rows per spectrum can be appended:
  - comb_fisher: per-class Fisher combined p-values computed via metap::sumlog (if available).
  - max_vote: per-class fraction of models casting the top-probability vote for that class.
- Models: this function is agnostic of the modeling engine as long as predict(type = "prob") is implemented (e.g., caret multinom/nnet/ranger/xgb, glmnet, etc.).

Value

If Reference is missing (or has < 2 levels), a data.frame with:

- name: spectrum name (from MassPeaks metaData fullName/file when available)
- method: model identifier (from model$method; suffixed with "_i" if needed)
- one column per class with predicted probabilities
- pred_max_p: predicted class (argmax of probabilities)

If Reference is provided with at least two levels, a list with:

- Prob.results: the predictions data.frame as above
- Confusion.Matrix: a list of caret::confusionMatrix objects (one per method)

References

Kuhn, M. (2008). Building predictive models in R using the caret package. Journal of Statistical Software, 28(1), 1–26.

Alexandre Godmer, Yahia Benzerara, Emmanuelle Varon, Nicolas Veziris, Karen Druart, Renaud Mozet, Mariette Matondo, Alexandra Aubry, Quentin Giai Gianetto (2025). MSclassifR: An R package for supervised classification of mass spectra with machine learning methods. Expert Systems with Applications, 294, 128796. doi:10.1016/j.eswa.2025.128796

See Also

LogReg; SelectionVar; SelectionVarStat_fast; build_X_from_peaks_fast

Examples

```
library(MSclassifR)
library(MALDIquant)

## 1) Preprocess and detect peaks
data("CitrobacterRKIspectra", "CitrobacterRKImetadata", package = "MSclassifR")
spectra <- SignalProcessing(CitrobacterRKIspectra)
peaks   <- MSclassifR::PeakDetection(x = spectra, averageMassSpec = FALSE)

## 2) Build X and Y (sample-by-peak intensities + labels)
##    Option A: if you prefer the helper and a sparse return:
Y <- factor(CitrobacterRKImetadata$Species)
xy <- build_XY_from_peaks(peaks, labels = Y, normalize = "max", sparse = FALSE)
X <- xy$X
Y <- xy$Y

##    Option B: via MALDIquant::intensityMatrix (as in the original examples)
##IntMat <- MALDIquant::intensityMatrix(peaks)
##rownames(IntMat) <- paste(CitrobacterRKImetadata$Strain_name_spot)
##IntMat[is.na(IntMat)] <- 0
##IntMat <- t(apply(IntMat, 1, function(x) x / max(x)))  # per-spectrum max norm
##X <- t(IntMat)                                         # features in columns
##Y <- factor(CitrobacterRKImetadata$Species)
```

```
## 3) Select discriminant m/z with "cvp" method
a <- MSclassifR::SelectionVar(
  X, Y,
  MethodSelection = "cvp",
  MethodValidation = "cv",
  PreProcessing = c("center","scale","nzv","corr"),
  NumberCV = 2,
  Metric = "Kappa"
)
sel_moz <- a$sel_moz

## 4) Train several models on the shortlisted m/z
model_lm  <- MSclassifR::LogReg(X = X, moz = sel_moz, Y = Y, number = 2,
 repeats = 2, Metric = "Kappa", kind = "linear")
model_nn  <- MSclassifR::LogReg(X = X, moz = sel_moz, Y = Y, number = 2,
 repeats = 2, Metric = "Kappa", kind = "nnet", Sampling = "up")
model_rf  <- MSclassifR::LogReg(X = X, moz = sel_moz, Y = Y, number = 2,
 repeats = 2, Metric = "Kappa", kind = "rf",  Sampling = "down")
model_svm <- MSclassifR::LogReg(X = X, moz = sel_moz, Y = Y, number = 2,
 repeats = 2, Metric = "Kappa", kind = "svm", Sampling = "up")

Models <- list(
  model_lm$train_mod,
  model_nn$train_mod,
  model_rf$train_mod,
  model_svm$train_mod
)

## 5) Predict classes for a subset of peaks; 6 Da tolerance for matching
prob_cat <- MSclassifR::PredictLogReg(
  peaks     = peaks[1:5],
  model     = Models,
  moz       = sel_moz,
  tolerance = 6,
  Reference = Y[1:5]
)
prob_cat

## 6) Meta-classifier strategy (several RF models + SMOTE + Fisher combine)
a2 <- MSclassifR::SelectionVar(X, Y, MethodSelection = "mda", Ntree = 5 * ncol(X))
sel_moz2 <- a2$sel_moz
models2 <- vector("list", 4L)
for (i in seq_along(models2)) {
  models2[[i]] <- MSclassifR::LogReg(
    X = X, moz = sel_moz2, Y = Y,
    number = 5, repeats = 5,
    kind = "rf", Metric = "Kappa",
    Sampling = "smote"
  )$train_mod
}
prob_cat2 <- MSclassifR::PredictLogReg(
  peaks = peaks,
```

```
  model = models2,
  moz   = sel_moz2,
  tolerance = 6,
  Reference = Y
)
```

SelectionVar *Variable selection using methods based on random forests and others.*

Description

This function performs variable selection (i.e. selection of discriminant mass-to-charge values) using several selection methods (see MethodSelection argument).

Usage

```
SelectionVar(X,
             Y,
            MethodSelection = c("RFERF", "RFEGlmnet", "VSURF", "sPLSDA", "mda",
                                "cvp", "boruta"),
             MethodValidation = c("cv", "repeatedcv", "LOOCV"),
             PreProcessing = c("center", "scale", "nzv", "corr"),
             Metric = c("Kappa", "Accuracy"),
             Sampling = c("no", "up","down", "smote"),
             NumberCV = NULL,
             RepeatsCV = NULL,
             Sizes,
             Ntree = 1000,
             ncores = 2,
             threshold = 0.01,
             ncomp.max = 10,
             nbf=0)
```

Arguments

X	a numeric matrix corresponding to a library of mass spectra. Each row of X is the intensities of a mass spectrum measured on mass-to-charge values. The columns are assumed to be mass-to-charge values.
Y	a factor with a length equal to the number of rows in X and containing the categories of each mass spectrum in X.
MethodSelection	a character indicating the method used for variables selection. Methods available: (1) "RFERF" for recursive feature elimination (RFE) coupled with random forests (see rfe in the caret R package); (2) "RFEGlmnet" for RFE with coupled with logistic regression; (3) "VSURF" for a method using random forests

(see VSURF in the VSURF R package); (4) "sPLSDA" for a method based on sparse partial least squares discriminant analysis (see splsda in the mixOmics); (5) "mda" for a method selecting variables from the distribution of the "mean decrease in accuracy" variables importances of a random forest (see importance function in the randomForest R package); (6) "cvp" for a method selecting variables from the distribution of the cross-validated permutation variables importances of a random forest (see CVPVI function in the vita R package); (7) "boruta" for a method selecting variables using the Boruta algorithm that iteratively compares importances of variables with importances of shadow variables, created by shuffling original ones (see Boruta function in the Boruta R package). Additional explanations are available in the Details section.

MethodValidation
: a character indicating the resampling method:"cv" for cross-validation; "repeatedcv" for repeated cross-validation; and "LOOCV" for leave-one-out cross-validation. Only used for the "RFERF", "RFEGlmnet" and "sPLSDA" methods.

NumberCV
: a numeric value indicating the number of K-folds for cross-validation. Only used for the "RFERF", "RFEGlmnet", "sPLSDA" and "cvp" methods.

RepeatsCV
: a numeric value indication the number of repeat(s) for K-folds for cross-validation or repeated cross-validation. Only used for the "RFERF", "RFEGlmnet" and "sPLSDA" methods.

PreProcessing
: a vector indicating the method(s) used to pre-process the mass spectra in X: centering ("center"), scaling ("scale"), eliminating near zero variance predictors ("nzv"), or correlated predictors ("corr"). Only used for the "RFERF", "RFEGlmnet" and "sPLSDA" methods.

Metric
: a character indicating the metric used to select the optimal model for the RFE algorithms. Possible metrics are the "Kappa" coefficient or the "Accuracy". This argument is not used for the "VSURF", "cvp", "mda" and the "sPLSDA" methods of MethodSelection. See details of the "SelectionVar" function.

Sampling
: a character indicating an optional subsampling method to handle imbalanced datasets: subsampling methods are either "no" (no subsampling), "up", "down" or "smote". "no" by default.

Sizes
: a numeric vector indicating the number of variables to select. Only used for the "RFERF", "RFEGlmnet" and "sPLSDA" methods. For the "RFERF" and "RFEGlmnet" methods, the final number of selected variables is the one giving the highest average "Metric" ("Accuracy" or "Kappa") on the folds used for cross-validation. It is thus bounded by NumberCV*max(Sizes). For the "sPLSDA" method, Sizes corresponds to the number of variables to test from the X dataset when estimating the sparse PLS-DA model (see test.keepX argument in the mixOmics R package).

Ntree
: a numeric value indicating the number of trees in random forests, only used if MethodSelection = "VSURF" or "mda" or "cvp". Note we advise to select a number highly superior to the total number of variables for a robust selection (to not miss some features in the subspaces used to build trees). It is 1000 by default.

ncores
: a positive integer only used for the cvp method. The number of cores to use, i.e. at most how many child processes will be run simultaneously. Must be at

least one, and parallelization requires at least two cores. If ncores=0, then the half of CPU cores on the current host are used.

ncomp.max — a positive integer indicating the maximum number of components that can be included in the sPLS-DA model (10 by default).

threshold — a numeric value corresponding to a threshold used for the optimal selection of the number of components included in the sPLS-DA model (0.01 by default). When the number of components increases and the balanced classification error rate (BER) does not change anymore, we keep the minimal number where the BER reaches a plateau (i.e. when BER(N)-BER(N+1)<threshold, we keep N). If a plateau is not reached, ncomp.max components are selected.

nbf — a numeric value corresponding to a number of simulated non discriminant features. This is used to improve the robustness of the estimation of the distribution of the variable importances for non discriminant features. Only used for the "mda" and "cvp" methods. 0 by default: no additional non discriminant feature is created.

Details

The selection of variables can be carried out with two different objectives: either to find a minimum number of variables allowing to obtain the highest possible accuracy (or Kappa coefficient), which involves the possible elimination of variables correlated between them (i.e. not bringing any additional predictive power with respect to some other variables); or to find all the variables in the dataset with a potential predictive power ("discriminant" variables).

The VSURF method attempts to accomplish only the first objective. The mda and cvp methods attempt to accomplish the second objective, as do the methods available in the SelectionVarStat function of our MSclassifR R package. The RFERF, RFEGlmnet and sPLSDA methods take as input a number of variables to be selected(Sizes argument), and can therefore be used with both objectives.

Within the framework of the second objective, either the mda or cvp methods can be used to estimate a number of discriminant variables from the importances of variables. The SelectionVarStat function can also be used to estimate this number from distributions of p-values. Of note, be sure that the Ntree argument is high enough to get a robust estimation with the mda or cvp methods.

The "RFEGlmnet" and "RFERF" methods are based on recursive feature elimination and can either optimize the kappa coefficient or the accuracy as metrics when selecting variables.

The "sPLSDA" method selects variables from the ones kept in latent components of the sparse PLS-DA model using an automatic choice of the number of components (when the balanced classification error rate (BER) reaches a plateau - see argument threshold).

The "mda" and "cvp" methods use the distribution of variable importances to estimate the number of discriminant features (mass-to-charge values). Briefly, the distribution of variable importances for useless (not discriminant) features is firstly estimated from negative importance variables by the method proposed in section 2.6 of Janitza et al.(2018). Next, the following mixture model is assumed: $F(x) = \pi \times F_u(x) + (1 - \pi) \times F_d(x)$ where F is the empirical cumulative distribution of variable importances of all the features, F_u the one of the useless features, F_d the one of the discriminative features, and π is the proportion of useless features in the dataset. From the estimated distribution of useless features, we can estimate quantile values x_q and compute $\epsilon_q = min(F(x_q)/q; 1)$ for each quantile q. The minimum of the ϵ_q corresponds to the estimated

proportion of useless features in the dataset, what allows estimating the number of discriminant features by $N_d = floor(N \times (1 - \pi))$ where N is the total number of features. Next, the N_d features with the highest variable importances are selected.

The "VSURF" and "sPLSDA" methods use the minimum mean out-of-bag (OOB) and balanced classification error rate (BER) metrics respectively.

The "boruta" method selects variables from the Boruta algorithm (see Kursa and Rudnicki (2010)). The maxRuns argument of the Boruta function is fixed to 3*ncol(X) to perform the selection of variables.

For Sampling methods available for unbalanced data: "up" corresponds to the up-sampling method which consists of random sampling (with replacement) so that the minority class is the same size as the majority class; "down" corresponds to the down-sampling method randomly which consists of random sampling (without replacement) of the majority class so that their class frequencies match the minority class; "smote" corresponds to the Synthetic Minority Over sampling Technique (SMOTE) specific algorithm for data augmentation which consist of creates new data from minority class using the K Nearest Neighbor algorithm.

See rfe in the caret R package, VSURF in the VSURF R package, splsda in the mixOmics R package, importance function in the randomForest R package, CVPVI function in the vita R package, and Boruta function in the Boruta R package for more details.

Godmer et al. (2025) presents a comparison of different pipelines using SelectionVar that can help you to optimize your workflow. For a comprehensive guide, additional applications, and detailed examples of using this package, please visit our GitHub repository: here.

Value

A list composed of:

sel_moz — a vector with discriminant mass-over-chage values.

For the "RFERF" and "RFEGlmnet" methods, it also returns the results of the rfe function of the caret R package.

For the "VSURF" method, it also returns the results of the results of the VSURF function of the VSURF R package.

For the "sPLSDA" method, it also returns the following items:

Raw_data — a horizontal bar plot and containing the contribution of features on each component.

selected_variables — data frame with uniques features (selected variables to keep and containing the contribution of features in order to class samples).See plotLoadings in the mixOmics R package for details.

For the "mda" and "cvp" methods, it also returns the following items:

nb_to_sel — a numeric value corresponding to an estimated number of mass-over-chage values where the intensities are significantly different between categories (see details).

imp_sel — a vector containing the variable importances for the selected features.

References

Kuhn, Max. (2012). The caret Package. Journal of Statistical Software. 28.

Genuer, Robin, Jean-Michel Poggi and Christine Tuleau-Malot. VSURF : An R Package for Variable Selection Using Random Forests. R J. 7 (2015): 19.

Friedman J, Hastie T, Tibshirani R (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software, 33(1), 1-22.

Kim-Anh Le Cao, Florian Rohart, Ignacio Gonzalez, Sebastien Dejean with key contributors Benoit Gautier, Francois, Bartolo, contributions from Pierre Monget, Jeff Coquery, FangZou Yao and Benoit Liquet. (2016). mixOmics: Omics. Data Integration Project. R package version 6.1.1. https://CRAN.R-project.org/package=mixOmics

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. J. Artif. Int. Res. 16, 1 (January 2002), 321–357.

Branco P, Ribeiro R, Torgo L (2016). "UBL: an R Package for Utility-Based Learning." CoRR, abs/1604.08079.

Janitza, S., Celik, E., Boulesteix, A. L. (2018). A computationally fast variable importance test for random forests for high-dimensional data. Advances in Data Analysis and Classification, 12, 885-915.

Miron B. Kursa, Witold R. Rudnicki (2010). Feature Selection with the Boruta Package. Journal of Statistical Software, 36(11), p. 1-13.

Alexandre Godmer, Yahia Benzerara, Emmanuelle Varon, Nicolas Veziris, Karen Druart, Renaud Mozet, Mariette Matondo, Alexandra Aubry, Quentin Giai Gianetto, MSclassifR: An R package for supervised classification of mass spectra with machine learning methods, Expert Systems with Applications, Volume 294, 2025, 128796, ISSN 0957-4174, doi:10.1016/j.eswa.2025.128796.

Examples

```
library("MSclassifR")
library("MALDIquant")

################################################################################
## 1. Pre-processing of mass spectra

# load mass spectra and their metadata
data("CitrobacterRKIspectra","CitrobacterRKImetadata", package = "MSclassifR")
# standard pre-processing of mass spectra
spectra <- MSclassifR::SignalProcessing(CitrobacterRKIspectra)
# detection of peaks in pre-processed mass spectra
peaks <- MSclassifR::PeakDetection(x = spectra, averageMassSpec=FALSE)
# matrix with intensities of peaks arranged in rows (each column is a mass-to-charge value)
IntMat <- MALDIquant::intensityMatrix(peaks)
rownames(IntMat) <- paste(CitrobacterRKImetadata$Strain_name_spot)
# remove missing values in the matrix
IntMat[is.na(IntMat)] <- 0
# normalize peaks according to the maximum intensity value for each mass spectrum
IntMat <- apply(IntMat,1,function(x) x/(max(x)))
```

```
# transpose the matrix for statistical analysis
X <- t(IntMat)
# define the known categories of mass spectra for the classification
Y <- factor(CitrobacterRKImetadata$Species)

##############################################################################
## 2. Perform variables selection using SelectionVar with RFE and random forest
# with 5 to 10 variables,
# up sampling method and trained with the Kappa coefficient metric
a <- SelectionVar(X,
                  Y,
                  MethodSelection = c("RFERF"),
                  MethodValidation = c("cv"),
                  PreProcessing = c("center","scale","nzv","corr"),
                  NumberCV = 2,
                  Metric = "Kappa",
                  Sizes = c(5:10),
                  Sampling = "up")

# Plotting peaks on the first pre-processed mass spectrum and highlighting the
# discriminant mass-to-charge values with red lines
PlotSpectra(SpectralData=spectra[[1]],Peaks=peaks[[1]],
            Peaks2=a$sel_moz,col_spec="blue",col_peak="black")

##############################################################################
## 3. Perform variables selection using SelectionVar with VSURF
# This function can last a few minutes
b <- SelectionVar(X, Y, MethodSelection = c("VSURF"))
summary(b$result)

##############################################################################
## 4. Perform variables selection using SelectionVar with "mda" or "cvp"
# option 1: Using mean decrease in accuracy
# with no sampling method
c <- SelectionVar(X,Y,MethodSelection="mda",Ntree=10*ncol(X))

# Estimation of the number of peaks to discriminate species
c$nb_to_sel

# Discriminant mass-to-charge values
c$sel_moz

# Plotting peaks on the first pre-processed mass spectrum and highlighting the
# discriminant mass-to-charge values with red lines
PlotSpectra(SpectralData=spectra[[1]],Peaks=peaks[[1]],
            Peaks2=c$sel_moz,col_spec="blue",col_peak="black")

# option 2: Using cross-validated permutation variable importance measures (more "time-consuming")
# with no sampling method
d <- SelectionVar(X,Y,MethodSelection="cvp",NumberCV=2,ncores=2,Ntree=1000)

# Estimation of the number of peaks to discriminate species
d$nb_to_sel
```

```
# Discriminant mass-to-charge values
d$sel_moz

# Plotting peaks on the first pre-processed mass spectrum and highlighting the
# discriminant mass-to-charge values with red lines
PlotSpectra(SpectralData=spectra[[1]],Peaks=peaks[[1]],
            Peaks2=d$sel_moz,col_spec="blue",col_peak="black")

# Mass-over charge values found with both methods ("mda" and "cvp")
intersect(c$sel_moz,d$sel_moz)
```

`SelectionVarStat` *Fast feature (m/z) selection using multiple hypothesis testing (LIMMA/ANOVA/Kruskal) with optional class balancing (no/up/down/SMOTE)*

Description

Runs univariate statistical tests across all features (columns) to identify discriminant m/z values. Supports three tests:

- "Limma": moderated F-test via the limma R package on k-1 contrasts between groups
- "anova": classical one-way ANOVA (implemented in C++ for speed)
- "kruskal": Kruskal–Wallis rank-sum test (implemented in C++ with tie correction)

Usage

```
SelectionVarStat(
  X,
  Y,
  stat.test = c("Limma", "anova", "kruskal"),
  pi0.method = "abh",
  fdr = 0.05,
  Sampling = c("no", "up", "down", "smote"),
  seed = NULL
)
```

Arguments

`X`	Numeric matrix with samples in rows and features (peaks) in columns. If a sparse Matrix is provided, it is coerced to a base R dense matrix. Infinities are set to NA; NAs are allowed.

Y	Class labels. A factor (or coercible to factor) of length nrow(X). Must contain at least two levels.
stat.test	Character string; which univariate test to use. One of "Limma", "anova", or "kruskal". Default is "Limma". • Limma uses limma::lmFit on ~ 0 + Y, applies k-1 contrasts versus a reference group with limma::makeContrasts and then limma::eBayes to compute a moderated F p-value testing overall between-group differences. • anova uses a fast C++ one-way ANOVA (upper-tail F p-value). • kruskal uses a fast C++ Kruskal–Wallis test with tie correction (upper-tail chi-square p-value).
pi0.method	Character; method for cp4p::estim.pi0 and cp4p::adjust.p. Default "abh". See cp4p documentation for options.
fdr	Numeric in (0, 1]; FDR threshold used to select features after p-value adjustment. Default 0.05.
Sampling	Character string; optional class balancing applied before testing. One of "no", "up", "down", "smote". Default "no". • "up": up-samples minority classes to the majority size (base R implementation). • "down": down-samples majority classes to the minority size (base R). • "smote": uses the package's internal smote_classif() on data.frame(Y, X). Column names of X are preserved; if SMOTE fails or yields <2 classes, the function falls back to up-sampling.
seed	Optional integer. If provided, sets the random seed for up/down sampling and SMOTE to ensure reproducibility.

Details

Optional class balancing can be applied before testing: no sampling, up-sampling, down-sampling, or SMOTE using the internal smote_classif() function. P-values are adjusted with the cp4p R package, and features below the FDR threshold are returned, together with an estimated proportion of nulls, pi0.

Missing values are preserved as NA: for ANOVA/Kruskal, the C++ backends skip NAs per feature; for Limma, missing values are handled by limma internally. Non-finite values (Inf/-Inf) are coerced to NA.

- Limma design: model.matrix(~ 0 + Y) is used; a full-rank set of k-1 contrasts vs the first level is constructed with limma::makeContrasts. The returned p-values are moderated F-test p-values for overall group differences.
- ANOVA/Kruskal implementations are in C++ (see anova_cols_cpp and kruskal_cols_cpp); they process all features in one pass and skip NAs per feature.
- Non-finite p-values (if any) are set to 1 before pi0/adjustment.
- SMOTE requires at least two observations per class; otherwise the function automatically falls back to up-sampling.
- For limma, if residual degrees of freedom are not positive (nrow(X) - nlevels(Y) <= 0), p-values are set to 1 with a warning.

Value

A list with:

- nb_to_sel: integer, floor(ncol(X) * (1 - pi0)) using cp4p::estim.pi0.
- n_selected_fdr: integer, number of features with adjusted p-value < fdr.
- sel_moz: character vector of selected feature names (columns of X) with adjusted p-value < fdr.
- ap: the full object returned by cp4p::adjust.p (contains adjusted p-values).

See Also

limma::lmFit, limma::contrasts.fit, limma::eBayes, limma::makeContrasts; cp4p::estim.pi0, cp4p::adjust.p

Examples

```
##############################################################################
## 1. Pre-processing of mass spectra

# load mass spectra and their metadata
data("CitrobacterRKIspectra","CitrobacterRKImetadata", package = "MSclassifR")
# standard pre-processing of mass spectra
spectra <- MSclassifR::SignalProcessing(CitrobacterRKIspectra)
# detection of peaks in pre-processed mass spectra
peaks <- MSclassifR::PeakDetection(x = spectra, labels = CitrobacterRKImetadata$Strain_name_spot)
# build matrix with intensities of peaks (rows = samples, columns = m/z)
Y <- factor(CitrobacterRKImetadata$Species)
xy <- build_XY_from_peaks(peaks, labels = Y, normalize = "max", sparse = FALSE)
X <- xy$X
Y <- xy$Y

##############################################################################
## 2. Estimate the optimal number of peaks to discriminate the different species

OptiPeaks <- MSclassifR::SelectionVarStat(X,
                             Y,
                             stat.test = "Limma",
                             pi0.method = "abh",
                             fdr = 0.05,
                             Sampling = "smote",
                             seed = 1)

## Estimation of the optimal number of peaks to discriminate species (from the pi0 parameter)
OptiPeaks$nb_to_sel

## discriminant mass-to-charge values estimated using a 5 per cent false discovery rate
OptiPeaks$sel_moz

## p-values and adjusted p-values estimated for all the tested mass-to-charge values
OptiPeaks$ap$adjp
```

SignalProcessing *Signal processing for MALDI-TOF spectra (wrapper to SignalProcessingUltra)*

Description

Backward-compatible wrapper that delegates to SignalProcessingUltra. Keeps the original argument names/signature so existing code continues to work.

Usage

```
SignalProcessing(
  x,
  transformIntensity_method = "sqrt",
  smoothing_method = "Wavelet",
  removeBaseline_method = "SNIP",
  removeBaseline_iterations = 25,
  calibrateIntensity_method = "TIC",
  alignSpectra_NoiseMethod = "MAD",
  alignSpectra_method = "lowess",
  alignSpectra_halfWs = 11,
  alignSpectra_SN = 3,
  tolerance_align = 0.002,
  referenceSpectra = NULL,
  minFrequency = 0.5,
  binPeaks_method = "strict",
  keepReferenceSpectra = FALSE,
  ...
)
```

Arguments

x	list of MALDIquant MassSpectrum objects.
transformIntensity_method	character, intensity transform (default "log").
smoothing_method	character, smoothing method ("Wavelet" UDWT).
removeBaseline_method	character, baseline method ("TopHat" default; "SNIP","ConvexHull" supported).
removeBaseline_iterations	integer, SNIP iterations if removeBaseline_method = "SNIP".
calibrateIntensity_method	character, intensity calibration ("PQN" default, or "TIC","median").
alignSpectra_NoiseMethod	character, noise estimator for peak finding pre-alignment ("MAD").

`alignSpectra_method`
: character, alignment engine: "cubic" (default), "lowess", or "landmark_cpp".

`alignSpectra_halfWs`
: integer, half window size for peak detection.

`alignSpectra_SN`
: numeric, SNR for peak detection.

`tolerance_align`
: numeric, tolerance for matching anchors to the reference during alignment. Use consistent units across your pipeline (Da by default here).

`referenceSpectra`
: optional MALDIquant MassPeaks object to use as alignment reference.

`minFrequency`
: numeric, minimum peak frequency to build reference if not provided (default 0.7).

`binPeaks_method`
: character, "strict" (default) or "relaxed" for reference peak binning.

`keepReferenceSpectra`
: logical, if TRUE and no reference provided, returns list(spectra=..., RefS=...).

`...`
: additional arguments passed to SignalProcessingUltra (e.g., n_workers, ref_sample_n).

Value

A list of processed MassSpectrum objects, or list(spectra, RefS) if keepReferenceSpectra = TRUE.

SignalProcessingUltra *Optimized signal processing for MALDI-TOF spectra (parallel + optional C++ alignment)*

Description

This function performs post-acquisition processing for lists of MALDIquant MassSpectrum objects. It parallelizes per-spectrum steps and can align with either MALDIquant's warping (cubic/lowess) or a fast landmark-based C++ algorithm ("landmark_cpp").

Usage

```
SignalProcessingUltra(
  x,
  transformIntensity_method = "log",
  smoothing_method = "Wavelet",
  removeBaseline_method = "TopHat",
  removeBaseline_iterations = 25,
  calibrateIntensity_method = "PQN",
  alignSpectra_NoiseMethod = "MAD",
  alignSpectra_method = c("cubic", "lowess", "landmark_cpp"),
  alignSpectra_halfWs = 11,
```

```
  alignSpectra_SN = 3,
  tolerance_align = 0.002,
  ppm_align = FALSE,
  referenceSpectra = NULL,
  minFrequency = 0.7,
  binPeaks_method = "strict",
  keepReferenceSpectra = FALSE,
  n_workers = NULL,
  ref_sample_n = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

x	list of MALDIquant MassSpectrum objects.
transformIntensity_method	character, intensity transform (default "log").
smoothing_method	character, smoothing method ("Wavelet" UDWT).
removeBaseline_method	character, baseline method ("TopHat" default; "SNIP","ConvexHull" supported).
removeBaseline_iterations	integer, SNIP iterations if removeBaseline_method = "SNIP".
calibrateIntensity_method	character, intensity calibration ("PQN" default, or "TIC","median").
alignSpectra_NoiseMethod	character, noise estimator for peak finding pre-alignment ("MAD").
alignSpectra_method	character, alignment engine: "cubic" (default), "lowess", or "landmark_cpp".
alignSpectra_halfWs	integer, half window size for peak detection.
alignSpectra_SN	numeric, SNR for peak detection.
tolerance_align	numeric, tolerance for matching anchors to the reference during alignment. Use consistent units across your pipeline (Da by default here).
ppm_align	logical, set TRUE if tolerance_align is in ppm (then interpreted as ppm).
referenceSpectra	optional MALDIquant MassPeaks object to use as alignment reference.
minFrequency	numeric, minimum peak frequency to build reference if not provided (default 0.7).
binPeaks_method	character, "strict" (default) or "relaxed" for reference peak binning.
keepReferenceSpectra	logical, if TRUE and no reference provided, returns list(spectra=..., RefS=...).

n_workers	integer, number of parallel workers (default: all cores minus one).
ref_sample_n	integer or NULL, if set, build the reference from a random subset of this many spectra.
verbose	logical, print progress.
...	passed to MALDIrppa::wavSmoothing (e.g., n.levels).

Value

A list of MassSpectrum objects, or a list with $spectra and $RefS if keepReferenceSpectra = TRUE.

`smote_classif` *SMOTE for classification datasets*

Description

Generate synthetic examples for minority classes using the SMOTE idea, to balance a classification dataset.

Usage

```
smote_classif(
  formula,
  data,
  k = 5,
  strategy = c("balance", "perc"),
  perc = NULL,
 metric = c("euclidean", "manhattan", "chebyshev", "canberra", "overlap", "heom",
    "hvdm", "pnorm"),
  p = 2,
  seed = NULL,
  C.perc = NULL
)
```

Arguments

formula	A model formula target ~ predictors indicating the response and predictors.
data	A data.frame containing the variables in the model.
k	Integer, number of nearest neighbors used by SMOTE (default 5).
strategy	One of "balance" (oversample to the max class size) or "perc" (oversample each class by a percentage). Default "balance".
perc	Numeric percentage used when strategy = "perc" (e.g., 100 means generate as many synthetic examples as existing in the class). Ignored for "balance".
metric	Distance metric for neighbor search: one of "euclidean", "manhattan", "chebyshev", "canberra", "overlap", "heom", "hvdm", "pnorm". Default "euclidean".

p	Numeric p for the p-norm when metric = "pnorm"; also used implicitly for "euclidean" (p=2) and "manhattan" (p=1). Default 2.
seed	Optional integer seed for reproducibility.
C.perc	Deprecated. Backward-compatibility alias for oversampling control. If character "balance", mapped to strategy = "balance". If a single numeric, mapped to strategy = "perc" and perc = C.perc. Other forms are ignored with a warning.

Details

The function supports multi-class data. With strategy = "balance" (default), each class is oversampled up to the size of the largest class. With strategy = "perc", each class c is oversampled by round(n_c * perc/100). Neighbors are computed within each class.

Value

A data.frame with synthetic rows appended, same columns and types as input.

Examples

```
data(iris)
imbal_iris <- iris[c(1:40, 51:100, 101:110), ]
table(imbal_iris$Species)
balanced_iris <- smote_classif(Species ~ ., imbal_iris)
table(balanced_iris$Species)
```

Index