Package 'MSnID'

December 31, 2024

Type Package

Title Utilities for Exploration and Assessment of Confidence of LC-MSn Proteomics Identifications

Version 1.40.0

Author Vlad Petyuk with contributions from Laurent Gatto

Maintainer Vlad Petyuk <petyuk@gmail.com>

Description Extracts MS/MS ID data from mzIdentML (leveraging mzID package) or text files. After collating the search results from multiple datasets it assesses their identification quality and optimize filtering criteria to achieve the maximum number of identifications while not exceeding a specified false discovery rate. Also contains a number of utilities to explore the MS/MS results and assess missed and irregular enzymatic cleavages, mass measurement accuracy, etc.

License Artistic-2.0

Depends R (>= 2.10), Rcpp

Imports MSnbase (>= 1.12.1), mzID (>= 1.3.5), R.cache, foreach, doParallel, parallel, methods, iterators, data.table, Biobase, ProtGenerics, reshape2, dplyr, mzR, BiocStyle, msmsTests, ggplot2, RUnit, BiocGenerics, Biostrings, purrr, rlang, stringr, tibble, AnnotationHub, AnnotationDbi, xtable

LazyData yes

biocViews Proteomics, MassSpectrometry, ImmunoOncology

git_url https://git.bioconductor.org/packages/MSnID

git_branch RELEASE_3_20

git_last_commit 8588a13

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2024-12-30

Contents

`MSnID-package` *MSnID: Utilities for Handling MS/MS Identifications*

Description

Extracts MS/MS ID data from mzIdentML (leveraging mzID package) or text files. After collating the search results from multiple datasets it assesses their identification quality and optimize filtering criteria to achieve the maximal identifications at a user specified false discovery rate. Additional utilities include:

1. post-experimental recalibration of mass measurement accuracy
2. assessment of irregular and missed cleavages given the enzyme cleavage pattern
3. assessment of false discovery rates at peptide-to-spectrum match, unique peptide and protein levels
4. leverages brute-force and sophisticated optimization routines (Nelder-Mead and simulated annealing) for finding the filtering criteria that provide the maximum spectrum, peptide or protein identifications while not exceeding a corresponding preset threshold of false discovery rate
5. converts the results into MSnSet class object as spectral counting data

Details

Package:	MSnID
Type:	Package
Version:	0.1.0
Date:	2014-04-02

License: Artistic-2.0

Author(s)

Vladislav A. Petyuk (<vladislav.petyuk@pnnl.gov>)

`accessions` *Non-redundant list of accession (protein) identifiers*

Description

Returns the non-redundant list of accession (protein) identifiers from the *MSnID* object. Most of the times accessions and proteins have the same meaning. However, there are cases, for example use of 6-frame stop-to-stop translation as FASTA file, where the entries are called with general term accessions rather then proteins. Currently, accessions and proteins have the same meaning in `MSnID`.

Usage

```
accessions(object, ...)
proteins(object, ...)
```

Arguments

`object`	An instance of class "MSnID".
`...`	ignored parameters

Value

Non-redundant list of accession (protein) identifiers.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

`peptides`

Examples

```
data(c_elegans)
head(accessions(msnidObj))
head(proteins(msnidObj))
```

add_mod_symbol *Annotates peptide sequences with modification symbols*

Description

Given the provided modification mass, annotates its position within the peptide sequence with provided symbol.

Usage

```
add_mod_symbol(object, mod_mass, symbol)
```

Arguments

object	An instance of class "MSnID".
mod_mass	(string) modification mass. Must match exactly one of the masses in report_mods.
symbol	(string) character that annotates the position of the modification

Value

Returns MSnID object with new or modified peptide_mod column. The column contains peptide sequence with amino acid modifications annotated with symbols.

Note

In current implementation the method can not distinguish modifications with the same mass, but different amino acid specificity as different modifications.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

report_mods map_mod_sites

Examples

```
m <- MSnID(".")
mzids <- system.file("extdata","phospho.mzid.gz",package="MSnID")
m <- read_mzIDs(m, mzids)

# to know the present mod masses
report_mods(m)

# TMT modification
m <- add_mod_symbol(m, mod_mass="229.1629", symbol="#")
# alkylation
m <- add_mod_symbol(m, mod_mass="57.021463735", symbol="^")
# phosphorylation
m <- add_mod_symbol(m, mod_mass="79.966330925", symbol="*")

# show the mapping
```

```
head(unique(subset(psms(m), select=c("modification", "peptide_mod"))))

# clean-up cache
unlink(".Rcache", recursive=TRUE)
```

`apply_filter` *Filters the MS/MS identifications*

Description

Filter out peptide-to-spectrum MS/MS identifications.

Usage

```
apply_filter(msnidObj, filterObj)
```

Arguments

`msnidObj`	An instance of class "MSnID".
`filterObj`	Either an instance of MSnIDFilter class or a `"character"`.

Details

`filterObj` argument evaluated to a `"logical"` for each entry of the MS/MS results table.

Value

Returns an instance of "MSnID" class with with peptide-to-spectrum matches that pass criteria defined in `filterObj` argument.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnID `evaluate_filter`

Examples

```
data(c_elegans)

## Filtering using string:
msnidObj <- assess_termini(msnidObj, validCleavagePattern="[KR]\\.[^P]")
table(msnidObj$numIrregCleavages)
# getting rid of any other peptides except fully tryptic
msnidObj <- apply_filter(msnidObj, "numIrregCleavages == 0")
show(msnidObj)

## Filtering using filter object:
# first adding columns that will be used as filters
msnidObj$msmsScore <- -log10(msnidObj$`MS-GF:SpecEValue`)
msnidObj$mzError <- abs(msnidObj$experimentalMassToCharge -
```

```
                        msnidObj$calculatedMassToCharge)
# setting up filter object
filtObj <- MSnIDFilter(msnidObj)
filtObj$msmsScore <- list(comparison=">", threshold=10.0)
filtObj$mzError <- list(comparison="<", threshold=0.1) # 0.1 Thomson
show(filtObj)
# applying filter and comparing MSnID object before and after
show(msnidObj)
msnidObj <- apply_filter(msnidObj, filtObj)
show(msnidObj)
```

`assess_missed_cleavages` *Counts the missing cleavage sites within the peptides sequence*

Description

Bottom-up proteomics approaches utilize endoproteases or chemical agents to digest proteins into smaller fragments called peptides. The enzymes recognize short amino acid motifs and cleave along the peptide bonds. Chemical agents such as `CNBr` also possess amino acid cleavage specificity. In real-world not every cleavage site get cleaved during the sample processing. Therefore settings of MS/MS search engines quite often explictly allow up to a certain number missed clevage sites per peptide sequence.

This function counts the number of missed cleavages in peptide sequence given the endoprotease cleavage motif in the form of regular expression. The default value for `missedCleavagePattern` is `[KR](?=[^P$])`, which corresponds to trypsin.

Usage

```
assess_missed_cleavages(object, missedCleavagePattern="[KR](?=[^P$])")
```

Arguments

`object`	An instance of class "MSnID".
`missedCleavagePattern`	Cleavage pattern in the form of regular expression.

Value

Returns an instance of "MSnID" class with additional column "numMissCleavages"

Warning

If the "MSnID" instance does not contain "peptide" column in MS/MS results table then there will be an error. E.g. you can check this by
`"peptide" %in% names(msnid)` where `msnid` is your "MSnID" instance.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

assess_termini

Examples

```
data(c_elegans)
# adding column numMissCleavages containing count of missed cleavages
msnidObj <- assess_missed_cleavages(msnidObj,
                                    missedCleavagePattern="[KR](?=[^P$])")
# check the distribution
table(msnidObj$numMissCleavages)
```

---

assess_termini *Checks if the peptide termini conforms with cleavage specificity*

Description

Bottom-up proteomics approaches utilize endoproteases or chemical agents to digest proteins into smaller fragments called peptides. The enzymes recognize short amino acid motifs and cleave along the peptide bonds. Chemical agents such as CNBr also possesses amino acid cleavage specificity.

This function checks if peptide termini are as expected given the enzymatic/chemical cleavage specificity. The default value for validCleavagePattern is [KR]\.[^P], which corresponds to trypsin.

Usage

```
assess_termini(object, validCleavagePattern="[KR]\\.[^P]")
```

Arguments

object	An instance of class "MSnID".
validCleavagePattern	Cleavage pattern in the form of regular expression.

Details

N- or C- protein termini are not considered as irregular clevages sites.

Value

Returns an instance of "MSnID" class with additional column "numIrregCleavages". If both termini conforms with cleavage specificity, then value is 0, if one or two termini are irregular then the values are 1 and 2, correspondingly.

Warning

If the "MSnID" instance does not contain "peptide" column in MS/MS results table then there will be an error. E.g. you can check this by
"peptide" %in% names(msnid) where msnid is your "MSnID" instance.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

assess_missed_cleavages

Examples

```
data(c_elegans)
# adding column numIrregCleavages
# containing count of irregularly cleaved termini
msnidObj <- assess_termini(msnidObj, validCleavagePattern="[KR]\\.[^P]")
# check the distribution
table(msnidObj$numIrregCleavages)
```

---

correct_peak_selection

Corrects wrong selection of monoisotopic peak

Description

In a typical setting instruments select ions for fragmentation primarily based on ion intensity. For low molecular weight peptides the most intense peak usually corresponds to monoisotopic peak (that is only C12 carbon isotopes). With increase of molecular weight, instensity of monoisotopic peak becomes smaller relatively to heavier peptide isotopes (that is containing one or a few C13 isotopes).

The function subtracts or adds the mass difference between C13 and C12 isotopes (1.0033548378 Da) if that reduces the mass error. Such a mass error arises from the fact that instrument may peak non-monoisotopic peak for fragmentation and thus report the mass that is different by ~ 1 Da.

Usage

```
correct_peak_selection(object)
```

Arguments

object	An instance of class "MSnID".

Value

Returns an instance of "MSnID" class with updated experimentalMassToCharge value.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnID recalibrate mass_measurement_error

Examples

```
data(c_elegans)
# plot original mass error
massErr <- (msnidObj$experimentalMassToCharge -
            msnidObj$calculatedMassToCharge) *
            msnidObj$chargeState
hist(massErr,xlim=c(-1,+1), breaks=seq(-1.5,+1.5,0.01))
# fixing the problem of picking wrong monoisotopic peak
msnidObj <- correct_peak_selection(msnidObj)
# plot fixed mass error
massErr <- (msnidObj$experimentalMassToCharge -
            msnidObj$calculatedMassToCharge) *
            msnidObj$chargeState
hist(massErr,xlim=c(-1,+1), breaks=seq(-1.5,+1.5,0.01))
```

`evaluate_filter` *Filters the MS/MS identifications*

Description

Filter out peptide-to-spectrum MS/MS identifications.

Usage

```
evaluate_filter(object, filter, level=c("PSM", "peptide", "accession"))
```

Arguments

`object`	An instance of class "MSnID".
`filter`	Either an instance of MSnIDFilter class or a `"character"`.
`level`	Level at which the filter will be evaluated. Possible values are "PSM", "peptide" and "accession". Multiple are OK. Default is all of them.

Value

Returns a matrix with with column names "fdr" and "n". Column "n" contains the number of features (spectra, peptides or proteins/accessions) passing the filter. Column "fdr" is the false discovery rate (i.e. identification confidence) for the corresponding features. Row names correspond to the provided levels.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

`MSnID` `id_quality`

Examples

```
data(c_elegans)

## Filtering using string:
msnidObj <- assess_termini(msnidObj, validCleavagePattern="[KR]\\.[^P]")
table(msnidObj$numIrregCleavages)
evaluate_filter(msnidObj, "numIrregCleavages == 0")

## Filtering using filter object:
# first adding columns that will be used as filters
msnidObj$msmsScore <- -log10(msnidObj$`MS-GF:SpecEValue`)
msnidObj$mzError <- abs(msnidObj$experimentalMassToCharge -
                        msnidObj$calculatedMassToCharge)
# setting up filter object
filtObj <- MSnIDFilter(msnidObj)
filtObj$msmsScore <- list(comparison=">", threshold=10.0)
filtObj$mzError <- list(comparison="<", threshold=0.1) # 0.1 Thomson
show(filtObj)
evaluate_filter(msnidObj, filtObj)
```

---

fetch_conversion_table

Fetches conversion table form one type of identifiers to another

---

Description

A wrapper function over **AnnotationHub** that helps to convert from one protein identifiers to another.

Usage

```
fetch_conversion_table(organism_name,
                       from, to,
                       backend="AnnotationHub",
                       snapshot_date=NULL)
```

Arguments

organism_name	(string) official organism name. E.g. "Homo sapiens", "Mus musculus" or "Rattus norvegicus".
from, to	(string) identifier names. Recommended names are "SYMBOL", "UNIPROT", "REFSEQ" and "ENSEMBLPROT". Other identifiers are possible, but use them at your own risk.
backend	(string) currently only **AnnotationHub**
snapshot_date	(string) snapshot date for **AnnotationHub**. Default is NULL meaning latest date.

Value

data.frame with first column name of from identifier, the second name of to identifier

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

Examples

```
conv_tbl <- fetch_conversion_table("Rattus norvegicus", "REFSEQ", "SYMBOL")
head(conv_tbl)
conv_tab <- fetch_conversion_table("Homo sapiens", "UNIPROT", "SYMBOL")
head(conv_tbl)
```

---

id_quality *Identification quality*

Description

Reports quality for a given level of identification (spectra, peptide or protein).

Usage

```
id_quality(object, filter=NULL, level=c("PSM", "peptide", "accession"))
```

Arguments

object	An instance of class "MSnID".
filter	Optional argument. Either an instance of MSnIDFilter class or a "character".
level	Level at which the filter will be evaluated. Possible values are "PSM", "peptide" and "accession". Multiple are OK. Default is all of them.

Value

Returns a matrix with with column names "fdr" and "n". Column "n" contains the number of features (spectra, peptides or proteins/accessions) passing the filter. Column "fdr" is the false discovery rate (i.e. identification confidence) for the corresponding features. Row names correspond to the provided levels.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnID evaluate_filter

Examples

```
data(c_elegans)
id_quality(msnidObj, level="peptide")
id_quality(msnidObj, filter="`MS-GF:PepQValue` < 0.01", level="peptide")
```

`infer_parsimonious_accessions` *Eliminates Redundancy in Peptide-to-Protein Mapping*

Description

Infer parsimonious set of accessions (e.g. proteins) that explains all the peptide sequences. The algorithm is a simple loop that looks for the accession explaining most peptides, records the peptide-to-accession mapping for this accession, removes those peptides, and then looks for next best accession. The loop continues until no peptides left. The method does not accept any arguments at this point (except the MSnID object itself).

Usage

```
infer_parsimonious_accessions(object, unique_only=FALSE, prior=character(0))
```

Arguments

`object`	An instance of class "MSnID".
`unique_only`	If TRUE, peptides mapping to multiple accessions are dropped and only unique are retained. If FALSE, then shared peptides assigned according to Occam's razor rule. That is a shared peptide is assigned to a protein with larger number of unique peptides. If the number of unique peptides is the same, then to the first accession. Default is FALSE.
`prior`	(character) character vector with prior justified proteins/accessions. If `unique_only == TRUE`, then `prior` argument is ignored. Essentially evidence by presense of unique peptide supercedes any prior. Default is `character(0)`, that is none.

Details

Although the algorithm is rather simple it is THE algorithm used for inferring maximal matching in bipartate graphs and is used in the IDPicker software.

Value

Returns an instance of "MSnID" with minimal set of proteins necessary to explain all the peptide sequences.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnID

Examples

```
data(c_elegans)

# explicitely adding parameters that will be used for data filtering
msnidObj$msmsScore <- -log10(msnidObj$`MS-GF:SpecEValue`)
msnidObj$absParentMassErrorPPM <- abs(mass_measurement_error(msnidObj))

# quick-and-dirty filter. The filter is too strong for the sake of saving time
# at the minimal set of proteins inference step.
msnidObj <- apply_filter(msnidObj, 'msmsScore > 12 & absParentMassErrorPPM < 2')

show(msnidObj)
msnidObj2 <- infer_parsimonious_accessions(msnidObj)
show(msnidObj2)
```

`map_mod_sites` *Maps the modifications to protein sequence*

Description

Given the peptide sequence with modification X.XXXX*XXXX.X and provided protein sequence FASTA, the method maps the location of the modification resulting in {protein ID}-{aa}{aa position}".

Usage

```
map_mod_sites(object,
              fasta,
              accession_col = "accession",
              peptide_mod_col = "peptide_mod",
              mod_char = "*",
              site_delimiter = "lower")
```

Arguments

`object`	An instance of class MSnID.
`fasta`	(AAStringSet object) Protein sequences read from a FASTA file. Names must match protein/accesison IDs in the accesson column of the MSnID object.
`accession_col`	(string) Name of the column with accession/protein IDs in the MSnID object. Default is "accession".
`peptide_mod_col`	(string) Name of the column with modified peptide sequences in the MSnID object. Default is "peptide_mod".
`mod_char`	(string) character that annotates the position of the modification. Default is "*".
`site_delimiter`	(string) either a single character or "lower" (default) meaning it will be the same amino acid symbol, but in lower case

Value

MSnID object with extra columns regarting the modification mapping. Most likely, what you need is `SiteID`.

`PepLoc`	(list of ints) position of the starting amino acid within protein sequence. It is a list, because there may be multiple occurences of the same sequence matching the peptide's sequence.
`PepLocFirst`	(int) position of the first occurence of the matching sequence
`ProtLength`	(int) protein length
`ModShift`	(vector of ints) positions of modified amino acids within peptide
`ModAAs`	(vector of characters) single-letter amino acid codes of the modified residues
`SiteLoc`	(list of vectors of ints) positions of the modified amino acids within protein for each occurence of the peptide
`Site`	(list of vectors of characters) modified sites encoded as amino acid symbol followed by position for each occurence of the peptide
`SiteCollapsed`	(list of characters) same as `Site`, but modified sites for each peptide occurence collapsed into one string
`SiteCollapsedFirst`	(character) first element of the `SiteCollapsed` list
`SiteID`	(character) accession ID concatenated with `SiteCollapsedFirst`

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

Examples

```
m <- MSnID(".")
mzids <- system.file("extdata","phospho.mzid.gz",package="MSnID")
m <- read_mzIDs(m, mzids)

# to know the present mod masses
report_mods(m)

# TMT modification
m <- add_mod_symbol(m, mod_mass="229.1629", symbol="#")
# alkylation
m <- add_mod_symbol(m, mod_mass="57.021463735", symbol="^")
# phosphorylation
m <- add_mod_symbol(m, mod_mass="79.966330925", symbol="*")

# show the mapping
head(unique(subset(psms(m), select=c("modification", "peptide_mod"))))

# read fasta for mapping modifications
fst_path <- system.file("extdata","for_phospho.fasta.gz",package="MSnID")
library(Biostrings)
fst <- readAAStringSet(fst_path)
# to ensure names are the same as in accessions(m)
names(fst) <- sub("(^[^ ]*) .*$", "\1", names(fst))
# # mapping phosphosites
m <- map_mod_sites(m, fst, "accession", "peptide_mod", "*", "lower")
```

```
head(unique(subset(psms(m), select=c("accession", "peptide_mod", "SiteID"))))

# clean-up cache
unlink(".Rcache", recursive=TRUE)
```

`mass_measurement_error` *Computes error of the parent ion mass to charge measurement*

Description

Computes error of the parent ion mass to charge measurement from `experimentalMassToCharge` and `calculatedMassToCharge`. The returned value is in points per million (ppm).

Usage

```
mass_measurement_error(object)
```

Arguments

`object`	An instance of class "MSnID".

Details

It may be more common to compute "mass measurement error". However, the practical difference in "mass measurement error" and "mass to charge measurement error" is negligible. Moreover, the instruments measure mass/charge ratio, not mass *per se*.

Value

Returns mass to charge measurement error in "ppm" units.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

`MSnID` `recalibrate` `correct_peak_selection`

Examples

```
data(c_elegans)
ppm <- mass_measurement_error(msnidObj)
hist(ppm, 100)
```

MSnID-class *The "MSnID" Class for Mass Spectrometry Based Proteomics Identification Data*

Description

The MSnID is a convenience class for manipulating the MS/MS search results.

Objects from the Class

The way to create objects is to call MSnID constructor function that takes as an input the project working directory workDir and the second argument if the cache from previous analysis should be cleaned cleanCache.

Slots

workDir: Object of class "character". containing working directory for the project. The .Rcache subdirectory stores the cached resuls form the previous analyses. The mechanism of caching relies on R.cache package.

psms: Object of class data.table that contains all the MS/MS identification results in the form of peptide(or protein)-spectrum-matches.

Methods

read_mzIDs signature(object, mzids):
Reads mzIdentML files into psms data.table slot of object MSnID instance. The functionality leverage mzID package facility. Note, the calls are memoised using R.cache facility. So if the call with the same list of files issues again, the results will be read from cache instead of re-parsing the mzIdentML files. See read_mzIDs

psms(object), psms(object)<-value: Gets and sets MS/MS search results as data.frame. See psms

dim signature(x = "MSnID"):
Returns the dimensions of the table with MS/MS identification data.

peptides signature(object = "MSnID"):
Returns unique peptide list. See peptides

accessions signature(object = "MSnID"):
Returns unique accessions (typically proteins) list. See accessions

proteins signature(object = "MSnID"):
Returns unique proteins list. See proteins

assess_termini Checks the agreement of peptide termini with enzymes cleavage specificity. The return value is theMSnID object with extra variable numIrregCleavages. See assess_termini

assess_missed_cleavages Checks if the peptide sequence contains the sites that were not cleaved by the enzyme. For details see assess_missed_cleavages

mass_measurement_error Returns parent ion mass measurement error in parts per million (ppm) units. Note, it requires experimentalMassToCharge and calculatedMassToCharge variables to be set. See mass_measurement_error

recalibrate Recalibrates, that is removes systematic error from experimentalMassToCharge measurements. See recalibrate

correct_peak_selection Subtracts or adds the mass difference between C13 and C12 isotopes (1.0033548378 Da) if that reduces the mass error. Such a mass error arises from the fact that instrument may peak non-monoisotopic peak for fragmentation and thus report the mass that is different by ~ 1 Da. See `correct_peak_selection`

apply_filter `signature(msnidObj="MSnID", filterObj="character")`
`signature(msnidObj="MSnID", filterObj="MSnIDFilter")`
The `filterObj` argument is a `"character"` or converted to a `"character"` text string that is evaluated to a `"logical"` for each entry of the MS/MS results table. Return value is a filtered `MSnID` object with entries that pass the applied filter. See `apply_filter`

evaluate_filter `evaluate_filter(object, filter, level = c("PSM","peptide", "accession")`
Returns a list with `fdr` and `n` elements. Argument `filter` is either `"character"` or `"MSnIDFilter"` object. Argument `level` can take one of the values `c("PSM","peptide","accession")` and controls the level filter is evaluated. See `evaluate_filter`

`id_quality signature(object="MSnID", ...)`
Other optional ... arguments are `filter` is an "MSnIDFilter" instance and `level`. The `level` values are one of "PSM", "peptide", "accession". The method returns FDR for given level depending of type of identifications. See `id_quality`

as(,"MSnSet") `signature(x = "MSnID")`: Coerce object from `MSnID` to `MSnSet`.

names signature(x="MSnID")
Returns the column names in the MS/MS results table.

`object$name`**,** `object$name<-value` Access and set `name` column in MS/MS search results table.

`object[[i]]`**,** `object[[i]]<-value` Access and set column `i` (character or numeric index) in MS/MS search results table.

as(,"MSnSet") `signature(from = "MSnID")`: Coerce object from `MSnID` to `MSnSet`.

as(,"data.table") `signature(from = "MSnID")`: Coerce object from `MSnID` to `data.table`.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

`MSnSet`, `mzID`.

Examples

```
## Not run:
msnidObj <- MSnID(".")
mzids <- system.file("extdata","c_elegans.mzid.gz",package="MSnID")
msnidObj <- read_mzIDs(msnidObj, mzids)
# clean up the cache directory
unlink(".Rcache", recursive=TRUE)

## End(Not run)
```

MSnIDFilter-class *The "MSnIDFilter" Class for Handling MS/MS Criteria, Relationships and Thresholds for Data Filtration.*

Description

The MSnIDFilter is a convenience class for manipulating the MS/MS filter for MS/MS results.

Objects from the Class

The way to create objects is to call MSnIDFilter constructor function that takes as input the MSnID class instance and (optionally) filterList.

Slots

MSnIDObj: An instance of class "MSnID".

filterList: An optional argument. A list with element names corresponding to column names avalable in MSnID instance. Each element contains sub-elements "comparison" and "threshold". "Comparison" is one of the relationship operators (e.g. ">") see Comparison for details. "Threshold" is the corresponding parameter value the identification has to be more or less (depending on comparison) to pass the filter.

Methods

show signature(object="MSnIDFilter"):
Prints MSnIDFilter object.

object$name, object$name<-value Access and set filterList elements.

names signature(x="MSnIDFilter")
Returns the names of the criteria.

as.numeric signature(x="MSnIDFilter")
Converts filterList into "numeric" vector. Vector names are the list element names. Vector values are threshold values. Comparison operators are lost.

length signature(x="MSnIDFilter")
Returns the number of criteria set in the "MSnIDFilter" object.

update signature(object="MSnIDFilter", ...)
The additional ... argument is numeric vector of the same length as the number of criteria in MSnIDFilter object. The method update the corresponding thresholds to new provided values.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnSet evaluate_filter apply_filter optimize_filter

Examples

```
data(c_elegans)

## Filtering using filter object:
# first adding columns that will be used as filters
msnidObj$msmsScore <- -log10(msnidObj$`MS-GF:SpecEValue`)
msnidObj$mzError <- abs(msnidObj$experimentalMassToCharge -
                        msnidObj$calculatedMassToCharge)
# setting up filter object
filtObj <- MSnIDFilter(msnidObj)
filtObj$msmsScore <- list(comparison=">", threshold=10.0)
filtObj$mzError <- list(comparison="<", threshold=0.1) # 0.1 Thomson
show(filtObj)
# applying filter and comparing MSnID object before and after
show(msnidObj)
msnidObj <- apply_filter(msnidObj, filtObj)
show(msnidObj)
```

---

msnidObj *Example mzIdenML File and MSnID Object*

Description

MSnID object from c_elegans_A_3_1_21Apr10_Draco_10-03-04_msgfplus.mzid.gz dataset from PeptideAltas repository id PASS00308.

Usage

```
data(c_elegans)
```

Examples

```
data(c_elegans)
msnidObj


## Not run:
## How to download the example mzID file from PeptideAltas:
try(setInternet2(FALSE),silent=TRUE)
ftp.loc <- "ftp://PASS00308:PJ5348t@ftp.peptideatlas.org/MSGFPlus_Results/MZID_Files/c_elegans_A_3_1_21Apr1
download.file(ftp.loc, "c_elegans.mzid.gz")
## End(Not run)

## Not run:
## Script for generation of the C. elegans example data:
msnidObj <- MSnID(".")
mzids <- system.file("extdata","c_elegans.mzid.gz",package="MSnID")
msnidObj <- read_mzIDs(msnidObj, mzids)
save(msnidObj, file='c_elegans.RData', compress='xz', compression_level=9)
# MD5 sum for the file is: a7c511a6502a6419127f1e46db48ed92
digest::digest(msnidObj)
# clean up the cache directory
unlink(".Rcache", recursive=TRUE)
```

```
## End(Not run)
```

`optimize_filter` *Filter criteria optimization to maximize the number of identifications given the FDR upper threshold*

Description

Adjusts parameters in the "MSnIDFilter" instance to achieve the most number of spectra, peptides or proteins/accessions within pre-set FDR upper limit.

Usage

```
optimize_filter(filterObj, msnidObj, fdr.max, method,
                level, n.iter, mc.cores=NULL)
```

Arguments

`filterObj`	An instance of class "MSnIDFilter".
`msnidObj`	An instance of class "MSnID".
`fdr.max`	Upper limit on acceptable false discovery rate (FDR).
`method`	Optimization method. Possible values are "Grid" or same values as for the `method` argument of the optim function. Tested and recommended arguments (besides "Grid") of `method` are "Nelder-Mead" or "SANN".
`level`	Determines at what level to perform optimization. Possible values are "PSM", "peptides" or "accession".
`n.iter`	For method "Grid" is approxomate number of evaluation point. For "Nelder-Mean" and "SANN" methods see optim.
`mc.cores`	Controls the number of enabled cores for parallel processing. Make sense only for "Grid" optimizaton. Default is `getOption("mc.cores", 2L)`. See mclapply for details.

Details

The "Grid" method is brute-force optimization through evaluation of approximately *n.iter* combinations of the parameters set in the "MSnIDFilter" object. The enumeration of the parameter combinations proceeds as follows. The *n.iter* number is getting split given the dimensionality of the problem (that is the number of filter parameters in the "MSnIDFilter" object. For each parameter the evaluation points are equally spaced according to quantiles of the parameter distribution. This way we enumerate the grid that has more evaluation points in relatively more dense areas.

Note, optimization is computationally expensive. Thus, the `optimize_filter` call is memoised using facilities from the R.cache package. Once the same call of `optime_filter` function issued second time the results will be retrieved from cache.

Value

Returns an instance of "MSnIDFilter" that is maximized to provide the most number of identifications while maintaining a pre-set confidence (FDR).

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnID evaluate_filter apply_filter

Examples

```
data(c_elegans)

# explicitely adding parameters that will be used for data filtering
msnidObj$msmsScore <- -log10(msnidObj$`MS-GF:SpecEValue`)
msnidObj$absParentMassErrorPPM <- abs(mass_measurement_error(msnidObj))

# Setting up filter object
filtObj <- MSnIDFilter(msnidObj)
filtObj$absParentMassErrorPPM <- list(comparison="<", threshold=10.0)
filtObj$msmsScore <- list(comparison=">", threshold=10.0)

system.time({
    filtObj.grid <- optimize_filter(filtObj, msnidObj, fdr.max=0.01,
                                    method="Grid", level="peptide", n.iter=50)})
show(filtObj.grid)

# Fine tuning. Nelder-Mead optimization.
system.time({
    filtObj.nm <- optimize_filter(filtObj.grid, msnidObj, fdr.max=0.01,
                                    method="Nelder-Mead", level="peptide",
                                    n.iter=50)})
show(filtObj.nm)

# Fine tuning. Simulated Annealing optimization.
system.time({
    filtObj.sann <- optimize_filter(filtObj.grid, msnidObj, fdr.max=0.01,
                                    method="SANN", level="peptide", n.iter=50)})
show(filtObj.sann)
```

---

peptides *Non-redundant list of peptides*

Description

Returns the non-redundant list of peptides from the *MSnID* object

Usage

```
peptides(object)
```

Arguments

object	An instance of class "MSnID".

Value

Non-redundant list of peptides.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

accessions proteins

Examples

```
data(c_elegans)
head(peptides(msnidObj))
```

---

psms *Peptide-to-spectrum matches*

Description

Returns results of MS/MS search (peptide-to-spectrum) matches in the form of data.frame.

Usage

```
psms(object, ...)
psms(object) <- value
```

Arguments

object	An instance of class "MSnID".
value	Value is a data.frame with MS/MS search results
...	ignored for now

Value

data.frame

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnID

Examples

```
data(c_elegans)
msnidDF <- psms(msnidObj)
head(msnidDF)
```

read_mzIDs *Populates MS/MS results table from mzIdentML files*

Description

Reads mzIdentML files into psms data.table slot of object MSnID instance. The functionality leverage mzID package facility. Note, the calls are memoised using R.cache facility. So if the call with the same list of files issues again, the results will be read from cache instead of re-parsing the mzIdentML files.

Usage

```
read_mzIDs(object, mzids, backend)
```

Arguments

object	An instance of class "MSnID"
mzids	paths to mzIdentML (mzid) files
backend	Package that is leveraged for parsing. Either 'mzID' or 'mzR' corresponding to mzID-package and mzR-class respectively. The 'mzR' parser is much faster, since it is based on C++ code. 'mzID' will be kept in the package for leagacy reasons. Note, the default is 'mzID'.

Details

mzIdentML files can be either as is or in gzip compressed form (*.mzid.gz).

Value

Returns an instance of "MSnID" class with @psms data.table slot populated with MS/MS identifications.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

flatten mzID

Examples

```
## Not run:
msnidObj <- MSnID(".")
mzids <- system.file("extdata","c_elegans.mzid.gz",package="MSnID")
msnidObj <- read_mzIDs(msnidObj, mzids)
# clean up the cache directory
unlink(".Rcache", recursive=TRUE)

## End(Not run)
```

recalibrate *Post-experimental recalibration of observed mass to charge ratios*

Description

Mass spectrometry measurements like any other real-worls measurements are prone to systematic errors. Typically they are minimized by instrument calibration prior the analysis. Nonetheless, the calibration may drift over time or be affected by some adverse factors (temperature or space charge fluctuations).

This function estimates and removes the systematic error from the datasets. The side effect is the recalibrated `experimentalMassToCharge` values.

Usage

```
recalibrate(object)
```

Arguments

`object`	An instance of class "MSnID".

Details

Currently it employs a very simple method of zero-centering the histogram of mass measurement errors. In the future it will contain more sophisticated recalibration routines.

Value

"MSnID" class instance with updated `experimentalMassToCharge`.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

MSnID mass_measurement_error correct_peak_selection

Examples

```
data(c_elegans)

# first let's fix the error of picking wrong monoisotopic peak
# otherwise the mass error range will be very large
msnidObj <- correct_peak_selection(msnidObj)

# original mass error in ppm
ppm <- mass_measurement_error(msnidObj)
hist(ppm, 200, xlim=c(-20,+20))

# The dataset is well calibrated. So let's introduce
# some mass measurement error.
msnidObj$experimentalMassToCharge <-
    msnidObj$experimentalMassToCharge * (1+0.00001)
```

```
# mass error (in ppm) after artificial de-calibration
ppm <- mass_measurement_error(msnidObj)
hist(ppm, 200, xlim=c(-20,+20))

# recalibration
msnidObj <- recalibrate(msnidObj)
ppm <- mass_measurement_error(msnidObj)
hist(ppm, 200, xlim=c(-20,+20))
```

`remap_accessions` *Changes accessions from one protein id to another*

Description

Changes accessions from one protein id to another.

Usage

```
remap_accessions(object,
                 conversion_table,
                 extraction_pttrn=c("\\|([^|-]+)(-\\d+)?\\|",
                                    "([A-Z]P_\\d+)",
                                    "(ENS[A-Z0-9]+)"),
                 path_to_FASTA=NULL)
```

Arguments

`object`	An instance of class "MSnID".
`conversion_table`	(data.frame) first column in the data frame corresponds to identifiers in the FASTA file. Second column is the new identifier.
`extraction_pttrn`	(string) regex pattern that extract protein identifier from FASTA entry name as first group (that is `"\\1"`). The most common patterns are the one corresponding to UniProt `"\\|([^|-]+)(-\\d+)?\\|"`, RefSeq `"^([A-Z]P_\\d+)"` and ENSEMBL `"^(ENS[A-Z0-9]+)"`. Other regex patterns can be accepted as well. Defaul is UniProt pattern.
`path_to_FASTA`	(string) path to FASTA file. If provided only accessions present in the given FASTA file will be retained.

Value

Returns an instance of "MSnID" with updated `accessions`.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

`remap_fasta_entry_names`

Examples

```
m <- MSnID(".")
mzids <- system.file("extdata","phospho.mzid.gz",package="MSnID")
m <- read_mzIDs(m, mzids)
head(m$accessions)
conv_tab <- fetch_conversion_table("Homo sapiens", "UNIPROT", "SYMBOL")
m2 <- remap_accessions(m, conv_tab, "\\|([^|-]+)(-\\d+)?\\|")
head(m2$accessions)
unlink(".Rcache", recursive=TRUE)
```

---

remap_fasta_entry_names

Remapping entries in FASTA file

Description

Remaps entries in the FASTA file from one protein identifier to another according to provided conersion table. Input is a path to FASTA file. Output is also a path to a new FASTA file with updated entry names.

Usage

```
remap_fasta_entry_names(path_to_FASTA,
                        conversion_table,
                        extraction_pttrn=c("\\|([^|-]+)(-\\d+)?\\|",
                                           "([A-Z]P_\\d+)",
                                           "(ENS[A-Z0-9]+)"))
```

Arguments

`path_to_FASTA`	(string) path to FASTA file
`conversion_table`	(data.frame) first column in the data frame corresponds to identifiers in the FASTA file. Second column is the new identifier.
`extraction_pttrn`	(string) regex pattern that extract protein identifier from FASTA entry name as first group (that is `"\\1"`). The most common patterns are the one corresponding to UniProt `"\\|([^|-]+)(-\\d+)?\\|"`, RefSeq `"^([A-Z]P_\\d+)"` and ENSEMBL `"^(ENS[A-Z0-9]+)"`. Other regex patterns can be accepted as well. Defaul is UniProt pattern.

Value

path to new FASTA file

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

Examples

```
library(Biostrings)
fst_path <- system.file("extdata","for_phospho.fasta.gz",package="MSnID")
readAAStringSet(fst_path)
conv_tab <- fetch_conversion_table("Homo sapiens", "UNIPROT", "SYMBOL")
fst_path_2 <- remap_fasta_entry_names(fst_path, conv_tab, "\\|([^|-]+)(-\\d+)?\\|")
readAAStringSet(fst_path_2)
file.remove(fst_path_2)
```

`report_mods` *Lists modification masses in the MSnID object*

Description

Parses out masses from the `modification` column and return them as `table` with counts.

Usage

```
report_mods(object, ...)
```

Arguments

`object`	An instance of class "MSnID".
`...`	currently not used

Value

Counts of each modification mass listed in `modification column`.

Author(s)

Vladislav A Petyuk <vladislav.petyuk@pnnl.gov>

See Also

`add_mod_symbol` `map_mod_sites`

Examples

```
msnidObj <- MSnID(".")
mzids <- system.file("extdata","phospho.mzid.gz",package="MSnID")
msnidObj <- read_mzIDs(msnidObj, mzids)
# reports the masses and number of their occurences
report_mods(msnidObj)
# clean-up cache
unlink(".Rcache", recursive=TRUE)
```

Index