Package 'Pbase'

October 11, 2016

Type Package

Title Manipulating and exploring protein and proteomics data

Version 0.12.2

Depends R (>= 2.10), methods, BiocGenerics, Rcpp, Gviz

Imports cleaver (>= 1.3.6), Biobase, Biostrings, IRanges, S4Vectors, mzID, mzR (>= 1.99.1), MSnbase (>= 1.15.5), Pviz, biomaRt, GenomicRanges, rtracklayer

Suggests testthat (>= 0.8), ggplot2, BSgenome.Hsapiens.NCBI.GRCh38, TxDb.Hsapiens.UCSC.hg38.knownGene, AnnotationHub, knitr, rmarkdown, BiocStyle

Description A set of classes and functions to investigate and understand protein sequence data in the context of a proteomics experiment.

License GPL-3

Author Laurent Gatto [aut], Sebastian Gibb [aut, cre]

Maintainer Sebastian Gibb <mail@sebastiangibb.de>, Laurent Gatto <lg390@cam.ac.uk>

VignetteBuilder knitr

URL https://github.com/ComputationalProteomicsUnit/Pbase

BugReports https://github.com/ComputationalProteomicsUnit/Pbase/issues

biocViews Infrastructure, Proteomics, MassSpectrometry, Visualization, DataImport, DataRepresentation

NeedsCompilation no

R topics documented:

`calculateHeavyLabels` *Calculate heavy labeled peptides*

Description

A function to calculate heavy labeled peptides for proteins stored in a `Proteins` object.

Usage

```
calculateHeavyLabels(proteins, peptides, maxN = 20L, nN = 4L, nC = 3L,
  endsWith = c("K", "R", "G"), ...)
```

Arguments

`proteins`	A `Proteins` object.
`peptides`	A named `character` vector containing the peptides of interest. The names must match the UniProt accession numbers of the proteins in `object`.
`maxN`	An `integer`, maximal length of the heavy labeled peptide.
`nN`	An `integer`, minimal number of amino acids at the N terminus.
`nC`	An `integer`, minimal number of amino acids at the C terminus.
`endsWith`	A `character` vector containing the allowed amino acids at the end of the resulting sequence (every peptide that doesn't end with one of these amino acids has to be one amino acid shorter as `maxN`).
`...`	Additional parameters passed to `.addOverhangs`.

Details

The digestion efficiency with enzymes like trypsin is below 100%. That's why spiked-in peptides for labeled quantitation have to follow the same digestion rules as the peptides of interest. Therefore it is necessary to extend the peptides of interest by a few amino acids on the N- and C-terminus. These extensions should not be a cleavage point of the used enzym. This methods provides an easy interface to find the sequences for heavy labeled peptides that could be used as spike-ins for the peptides of interest. Please see the references for a more detailed discussion.

TODO: There should be a function to find the best labels for a given protein automatically.

Value

A `data.frame` with 6 columns:

- ProteinThe Protein accession number.
- PeptideThe peptide of interest.
- N_overhangThe added sequence of the N-terminus.
- C_overhangThe added sequence of the C-terminus.
- spikeTideResultA short description of the used creation rule.
- spikeTideThe heavy labeled peptide that represents the peptide of interest best.

Author(s)

Sebastian Gibb <mail@sebastiangibb.de> and Pavel Shliaha

References

The complete description of the issue: https://github.com/sgibb/cleaver/issues/5

Kito, Keiji, et al. A synthetic protein approach toward accurate mass spectrometric quantification of component stoichiometry of multiprotein complexes. Journal of proteome research 6.2 (2007): 792-800. http://dx.doi.org/10.1021/pr060447s

Examples

```
## example protein database
data(p, package = "Pbase")

## digest proteins into peptides
cleavedProteins <- cleave(p)

## find spike-ins for the peptides of interest
calculateHeavyLabels(cleavedProteins,
                      peptides = c(A4UGR9 = "MEGFHIK",
                                   A4UGR9 = "QGNMYTLSK",
                                   A6H8Y1 = "GSTASNPQR"))
```

`etrid2grl` *From a transcript identifier to* `GRanges` *object*

Description

This function takes on or more Ensembl transcript identifiers, queries Biomart and constructs a `GRangesList` object as would `Gviz::BiomartGeneRegionTrack` for a genomic region (in fact, currently most of the code has been taken from `Gviz:::.fetchBMData` and `GViz:::.chrName` is used to validate chromosome names).

Usage

```
etrid2grl(etrid, ens, use.names = FALSE)
```

Arguments

`etrid`	A vector of Ensembl transcript identifiers.
`ens`	A instance of class `Mart` from biomaRt. If missing, `useMart("ensembl", "hsapiens_gene_ensembl")` is used.
`use.names`	If set to `TRUE` and `etrid` has names, then the latter are used to name the output.

Value

A `GRangesList` object of length `length(etrid)`.

Author(s)

Laurent Gatto

Examples

```
id <- c("ENST00000612959", "ENST00000317091")
grl1 <- etrid2grl(id[1])
grl1
grl <- etrid2grl(id)
stopifnot(all.equal(id, names(grl)))
```

`isReverse` *Are all the ranges on the same strand*

Description

Are all the ranges on the same strand

Usage

```
isReverse(gr)

isForward(gr)
```

Arguments

`gr`	A `GRanges` object.

Value

A logical if *all* the ranges in the `gr` object are on the `"-"` (or `"+"` for codeisForward) strand.

Author(s)

Laurent Gatto

`mapToGenome-methods` *Map range coordinates between proteins and genome space*

Description

Map range coordinates between peptide features along proteins and genome (reference) space.

Usage

```
## S4 method for signature 'Proteins,GRangesList'
mapToGenome(x, genome, drop.empty.ranges = TRUE, ...)
## S4 method for signature 'Proteins,GRangesList'
pmapToGenome(x, genome, drop.empty.ranges = TRUE, ...)
```

Arguments

`x`	`Proteins` object containing peptides `pranges` to be mapped.
`genome`	A `GRangesList` object used to map between `x` and the result. The ranges are typically created by the `etrid2grl` function.
`drop.empty.ranges`	`TRUE` (default) or `FALSE`. Should empty ranges be dropped?
`...`	Additional parameters passed to inner functions. Currently ignored.

Details

- `mapToGenome` maps the `pranges(x)` to the ranges of `genome`. Unless `x` and `genome` are of length 1, both must be named and items of `x` are matched to items of `genome` using their respective names. Names that do not co-occur in `x` and `genome` are ignored. If we have

  `seqnames(x)`: `"A"`, `"B"` and `"C"`

  and

  `names(genome)`: `"C"`, `"A"`, `"a"`, `"z"`, `"A"` and `"A"`.

  the names of the output will be

  `"A"`, `"A"`, `"A"` and `"C"`.

  The output is ordered by (1) `seqnames(x)` and (2) the order of the elements in `genome`.

  In case less than `length(x)` are mapped, as for `p["B"]` above, a message informs the user.
- `pmapToGenome` is the element-wise (aka 'parallel') version of `mapToGenome`. The i-th `pranges(x)` is mapped to the i-th range in `genome`. `x` and `genome` must have the same length and do not need to be named (names are ignored).

Value

A named `GRangesList` object, with names matching `names(genome)`. For `pmapToGenome`, the return value will have the same length as the inputs.

Author(s)

Laurent Gatto

See Also

- See ?mapToAlignments in the **GenomicAlignments** package for mapping coordinates between reads (local) and genome (reference) space using a CIGAR alignment.
- See ?mapToTranscripts in the **GenomicRanges** package for mapping coordinates between features in the transcriptome and genome space.
- The proteinCoding function to remove non-protein coding ranges before mapping peptides to their genomic coordinates.
- The mapping vignette for examples and visualisations.

See plotAsAnnotationTrack and plotAsAnnotationTrack for more details about the two plotting functions.

Examples

```
data(p)

grl <- etrid2grl(acols(p)$ENST)
pcgrl <- proteinCoding(grl)

plotAsGeneRegionTrack(grl[[1]],
                      pcgrl[[1]])

mp <- mapToGenome(p[4], pcgrl[4])

plotAsAnnotationTrack(mp[[1]], pcgrl[[4]])

pmapToGenome(p, pcgrl)
```

---

p *Data accompanying the* Pbase *package*

Description

A small example Proteins test instance. This object is likely to change on a regular basis. It will be described more thoroughly when it becomes stable. The MSMS spectra that were searched against the database are available in the pms MSnExp object.

Usage

```
data(p)
data(pms)
```

See Also

The `Pbase-data` vignette.

Examples

```
data(p)
p
data(pms)
pms
```

`plotAsAnnotationTrack` *Plot gene region and annotation tracks*

Description

These functions convert ranges of peptides or exons to `AnnotationTrack` or `GeneRegionTrack` objects from the `Gviz` package and produces the corresponding plot. The genome argument controls whether additional ideogram and axis tracks are to be plotted. `plotAsAnnotationTrack` plots peptides that span multiple exons in red and connects them with a grey line. See `pmapToGenome` for example code.

Usage

```
plotAsAnnotationTrack(x, ..., genome = "hg38", plot = TRUE)

plotAsGeneRegionTrack(..., genome = "hg38", plot = TRUE)
```

Arguments

`x`	A `Granges` object containing peptides genomics coordinates, typically generated by `pmapToGenome`. These ranges are converted to a `AnnotationTrack`.
`...`	One or more `GRanges` instances, typically resulting from calling `etrid2grl`, or, a single `GRangesList`. These ranges are converted to `GeneRegionTrack` instances.
`genome`	A `character` of length 1, giving the name of the genome. Default is `"hg38"`. If `NULL`, no chromosome and axis tracks are displayed.
`plot`	A `logical` defining if the figure should be plotted. Default is `TRUE`.

Value

Used for its plotting side effects. Invisible returns a list of tracks.

Author(s)

Laurent Gatto

Pparams-class *Class* "Pparams"

Description

Pbase parametrisation infrastructure.

Objects from the Class

New Pbase parameters can be generated with the Pparams() constructor. Pparams instances control various aspects of Pbase functions, as described in the *Slots* section below. If no parameters are passed to the respective functions, default values from Pparams() are used.

Slots

DbFormat**:** The format of the protein sequence fasta database used to generate the Proteins object. Currently only "UniProt" is supported. "RefSeq" will be added as well as a mechanism to support arbitrary and custom fasta header.

IdFormat**:** The format of the identification data files used to add pfeatures to Protein instances. Currently, mzIdentML is supported.

IdReader**:** Package to be used to load the identification data. Currently one of mzR (via the openIDfile and psms functions) or mzID (via the mzID and flatten functions). Differences between these two architectures include the metadata available in the Proteins' pfeatures, speed and stability (mzR is much faster but less mature and currently susceptible to crashes).

verbose**:** A logical defining if the various functions display messages (default) or remain silent.

Methods

show signature(object = "Pparams"): ...

Author(s)

Laurent Gatto <lg390@cam.ac.uk>

Examples

```
Pparams()
Pparams(IdReader = "mzID")

try(Pparams(IdReader = "mzid"))
```

`proteinCoding-methods` *Only keep protein coding ranges*

Description

Removed all the ranges that are not protein coding. Typically used on the output of `etrid2grl` before `mapToGenome`.

Methods

`signature(object="GRanges", mcol="character", coding="character")` Removes all the ranges that are not annotated as protein coding ranges, i.e. ranges whose `mcols()$mcol` is different from `coding`. The default values are `mcols()feature` and `"protein_coding"`. The method return the `GRanges` trimmed from all non-matching ranges.

`signature(object="GRangesList", mcol="character", coding="character")` As above but for `GRanges` in a `GrangesList`.

`Proteins-class` *The* `Proteins` *Class for Proteomics Data And Meta-Data*

Description

The `Proteins` class encapsulates data and meta-data for proteomics experiments. The class stores the protein sequences as well as specific subsets of interest, typically peptides, as ranges. The `Proteins` instances, the sequence and peptide slots are described by their respective metadata attributes.

Objects from the Class

Objects can be created using its constructor `Proteins`. The constructor either takes a `fasta` file name as first argument or, alternatively a named `uniprotIds` argument with valid UniProt accession numbers (not yet implemented).

Details

An instance of class `Proteins` is characterised by one or multiple protein sequences that can be accessed as `AAStringSet` with the `aa` accessor. Sequence-specific annotation, such as accession numbers, protein and gene names, ... is available with the `ametadata` or `acols` methods. General metadata such as the data of creation of the instance are stored as a `list` returned by the `metadata` accessor, which would typically contain a `created` character that documents when the object was created, a reference `genome` descriptor, a `UniProtRelease` with the release data of the UniProt database and possibly others.

Each sequence of a `Proteins` instance can also be characterised by a set of specific ranges describing peptides of interest. These *peptide features* can be extracted as an `AAStringSetList`, where each protein sequence contains 0 or more peptide features. These peptides features are encode as ranges along the original proteins sequences (a `list` of `IRanges`) that can be extracted with the `pranges` function. These peptide features have their own metadata describing for example peptide identification scores, number of missed cleavages, ... available with the `pmetadata` or `pcols` methods.

See also the `Pbase-data` vignette.

Developement notes

Since version 0.2.0, `addIdentificationData` supports multiple identification file names to be added to a `Proteins` instance (argument renamed `filenames`) using either `mzID` or `mzR`. Added new `Pparams` parametrisation infrastructure.

See `news(package = "Pbase")` for a description of all changes.

Other possible metadata fields: `Uniprot.sw`, `biomaRt` instances.

Slots

`metadata`**:** Object of class `"list"` containing global metadata, accessed with `metadata`.

`aa`**:** Object of class `"AAStringSet"` storing the protein sequences, accessed with `aa`.

`pranges`**:** Object of class `"CompressedIRangesList"` containg protein feature ranges such as theoretical (obtain by in silico cleavage) or observed peptides. Accessed as an `IRangesList` with `pranges` or and `AAStringSetList` with `pfeatures`.

`.__classVersion__`**:** Object of class `"Versions"` documenting the class verions. Intended for developer use and debugging.

Extends

Class `"Versioned"`, directly.

Methods

aa `signature(x = "Proteins")`: Returns an `AAStringSet` instance representing the sequences of the proteins.

pfeatures `signature(x = "Proteins")`: ...

pranges `signature(x = "Proteins")`: ...

metadata `signature(x = "Proteins")`: Returns a `list` of global metadata of the instance `x`, including data of instance creation or, if created from a set of UnitProt identifiers (see constructors above), the UniProt version and `UnitProt.WS` version number.

ametadata `signature(x = "Proteins")`: Returns a `DataFrame` of protein metadata.

acols `signature(x = "Proteins")`: See `ametadata`.

pmetadata `signature(x = "Proteins")`: Returns a `list` of feature metadata.

pcols `signature(x = "Proteins")`: See `pmetadata`.

avarLabels `signature(x = "Proteins")`: Returns the names of the sequences metadata.

pvarLabels `signature(x = "Proteins")`: Returns the names of the peptide feature metadata.

seqnames `signature(x = "Proteins")`: Returns the protein sequence names defined as UniProt accession numbers.

length `signature(x = "Proteins")`: Returns the number of proteins.

[`signature(x = "Proteins", i = "ANY", j = "missing")`: Creates a subset of the `Proteins` insance.

[[`signature(x = "Proteins", i = "ANY", j = "missing")`: Returns an `AAString` instance representing the sequence of the selected protein.

pfilter `signature(x = "Proteins", mass = "numeric", len = "numeric", ...)`: ...

cleave `signature(x = "Proteins", enzym = "character", missedCleavages = "numeric")`: Cleaves all proteins using the `enzym` rule while allowing `missedCleavages` missing cleavages. Please see `cleave` for details.

addIdentificationData `signature(object = "Proteins",id = "character", rmEmptyRanges = "logical", par = "Pparams")`: Adds identification data from an IdentMzMl file (`id`) to the Proteins `object`. If `rmEmptyRanges` is `TRUE` proteins without any identification data are removed. See `Pparams` for further settings.

addPeptideFragments `signature(object = "Proteins",filenames = "character", rmEmptyRanges = "logical", "Pparams")`: Adds identification data from a fasta file (`filenames`) to the Proteins `object`. Please note that both fasta files (the origin of the Proteins `object` and the ones given in `filenames`) must share the same Uniprot accession numbers. If `rmEmptyRanges` is `TRUE` proteins without any identification data are removed. See `Pparams` for further settings.

plot `signature(x = "Proteins", y = "missing")`: Plots all proteins and associated peptides using the `Gviz`/`Pviz` infrastructure.

show `signature(object = "Proteins")`: Displays object summary as text.

Functions

rmEmptyRanges `signature(x = "Proteins")`: removes proteins with empty peptide ranges.

proteotypic `signature(x = "Proteins")`: returns a modified Proteins object. `pcols(x)` gains a "Proteotypic" logical column, indicating of the peptide is proteotypic or now.

proteinCoverage `signature(pattern = "Proteins")`: calulates the coverage of proteins. `pcols(x)` gains a "Coverage" numeric column.

isCleaver `signature(x = "Proteins", missedCleavages = "numeric")`: Tests whether a Protein object was cleaved already.

Author(s)

Laurent Gatto <lg390@cam.ac.uk> and Sebastian Gibb <mail@sebastiangibb.de>

References

Definition of the UniProt fasta comment format: `http://www.uniprot.org/help/fasta-headers`

See Also

`calculateHeavyLabels`

Examples

```
## Create a Protein object reading all proteins from a fasta file.
fastaFiles <- list.files(system.file("extdata", package = "Pbase"),
                         pattern = "fasta", full.names = TRUE)
p <- Proteins(fastaFiles)
p
metadata(p)

## Adding custom metadata
metadata(p, "Comment") <- "I love R"
metadata(p)

## Plotting
plot(p[1:5], from = 1, to = 30)

## Cleaving
pp <- cleave(p[1:100])
pp <- proteotypic(pp)
pp
pcols(pp[1:2])

plot(pp[1:2], from = 20, to = 30)

## Protein coverage
pp <- proteinCoverage(pp)
avarLabels(pp)
acols(pp)$Coverage
pp


## Add indentification data
idfile <- system.file("extdata/Thermo_Hela_PRTC_selected.mzid",
                      package = "Pbase")
p <- addIdentificationData(p, idfile)
pranges(p)
pfeatures(p)

plot(p[1])
plot(p[1], # the first protein has 36 peptides
     fill = c(rep("orange", 13), rep("steelblue", 13)))
```

Index