Package ‘Pbase’

October 16, 2014

Type Package

Title Manipulating and exploring protein and proteomics data

Version 0.4.0

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

Imports
cleaver (>= 1.3.6), Biobase, Biostrings, IRanges, S4Vectors,mzID, mzR (>= 1.99.1), MSnbase (>= 1.13.5), Pviz

Suggests testthat (>= 0.8), ggplot2, BSgenome.Hsapiens.UCSC.hg19,biomaRt, TxDb.Hsapiens.UCSC.hg19.knownGene, knitr, 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

R topics documented:

p — *An* `Proteins` *object and its MSMS spectra*

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
```

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"))
```

`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 of 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.

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", ...)`: ...

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

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

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

addIdentificationData `signature(object = "Proteins",filenames = "character", rmEmptyRanges = "logical", "Pparams")`: ...

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") ...

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`

Examples

```
fastaFiles <- list.files(system.file("extdata", package = "Pbase"),
                         pattern = "fasta", full.names = TRUE)
p <- Proteins(fastaFiles)
p

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


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

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

pp <- proteinCoverage(pp)
avarLabels(pp)
pp
```

```
idfile <- system.file("extdata/Thermo_Hela_PRTC_1_selected.mzid",
                      package = "Pbase")
p <- addIdentificationData(p, idfile)
pranges(p)
pfeatures(p)

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

Index