Package 'mavevis'

April 26, 2018

Title Visualization for MaveDB

Version 0.0.0.9000

Description Query data from MaveDB and visualize as genophenograms with added tracks for structure information.

Depends R (>= 3.2.3)

License GPL

Encoding UTF-8

LazyData true

Imports rapimave, hgvsParseR, yogitools, httr, gdata, hash

Suggests testthat

RoxygenNote 6.0.1

Remotes github::jweile/rapimave, github::jweile/hgvsParseR, github::jweile/yogitools

SystemRequirements DSSP, OpenSASA, ClustalO

R topics documented:

`calc.conservation` *Calculate position-wise conservation from a Uniprot accession*

Description

Retrieves the 90 calculate a multiple sequence alignment and uses the AMAS algorithm to derive the position-wise sequence conservation. If the given accession has been used as an input before, a cached sequence alignment will be used instead.

Usage

```
calc.conservation(acc)
```

Arguments

`acc`	the Uniprot accession

Value

a numerical vector with the position-wise conservation.

`calc.strucfeats` *Calculate structural features*

Description

Calculates structural features of a protein from a given PDB entry

Usage

```
calc.strucfeats(pdb.acc, main.chain)
```

Arguments

`pdb.acc`	The PDB accession to use, e.g. "3UIP"
`main.chain`	The chain identifier in the PDB file that corresponds to the protein of interest. Should be a single uppercase letter, e.g. "A".

Value

a `data.frame` detailing secondary structure, solvent accessibility, and burial in interfaces.

Examples

```
## Not run:
sfeats <- calc.strucfeats("3UIP","A")

## End(Not run)
```

---

check.async.progress *Check progress on asynchronous job*

Description

Retrieves the standard out log of an asynchronously launched job, if it exists. If the job doesn't exist or hasn't written any log output, the function returns NULL.

Usage

```
check.async.progress(jobID)
```

Arguments

jobID	the ID of the job

Value

the contents of the stdout log for the given job, or NULL if none exists

---

dashboard *Draw dashboard for scoreset*

Description

Retrieves a scoreset entry from MaveDB and renders a dashboard plot consisting of a genophenogram heatmap, interface burial, solvent accessibility, secondary structure and conservation. The output is controlled by the outFormats parameter and can be either on an X11 device or in PDF or PNG format (or any combination thereof). Structural and conservation information is obtained from UniProtKB and PDB, thus the respective database accessions are required. Multiple PDB files can be used at once.

Usage

```
dashboard(ssid, uniprotId, pdbs, mainChains, wt.seq = NULL, seq.offset = 0,
  syn.med = NULL, stop.med = NULL, overrideCache = FALSE,
  outFormats = c("pdf", "png"), pngRes = 100, outID = ssid)
```

Arguments

ssid	The MaveDB ScoreSet ID of the dataset to be visualized
uniprotId	The UniprotKB accession for the corresponding protein
pdbs	A vector of PDB accessions to be used for the structure tracks
mainChains	the chain identifiers corresponding to the protein in question for each provided pdb accession.
wt.seq	An (optional) wild-type sequence for the given protein to use instead of the sequence found in UniprotKB. This can be either a nucleotide or amino acid sequence.
seq.offset	A parameter describing the start position of the map in the protein's amino acid sequence. For example, if only a domain was scanned.
syn.med	The median value of synonymous variants as an estimate of wild-type function. This is only necessary of no synonymous variants are present in the dataset.
stop.med	The median value of nonsense variants as an estimate of complete loss of function. This is only necessary if no nonsense variants are present in the dataset.
overrideCache	defaults to FALSE. If set to TRUE, data will always be re-downloaded from remote locations instead of using a local cache.
outFormats	a vector containing any of the following strings: x11, pdf, png . Using two or all three at once is allowed and will result in multiple output files.
pngRes	Resolution of PNG output in DPI. Defaults to 100.
outID	a name for the output file to which the plot will be written. Defaults to ssid.

Examples

```
## Not run:
dashboard(
 ssid="SCS000001A.1", uniprotId="P46937",
 pdbs=c("2LAY","2LTW"), mainChains=c("A","A"),
  wt.seq="GACGTTCCACTGCCGGCTGGTTGGGAAATGGCTAAAACTAGTTCTGGTCAGCGTTACTTCCTGAACCACATCGACCAGACCACCACGTGGCAGGACCCGC
   seqOffset=170, syn.med=0
)

## End(Not run)
```

---

dashboard.async.run *Launch asynchronous dashboard job*

Description

Launches a job running the dashboard function in a separate system process in the background. The job is given a unique ID, which is returned by this function.

Usage

```
dashboard.async.run(ssid, uniprotId, pdbs, mainChains, wt.seq = NULL,
  seq.offset = 0, syn.med = NULL, stop.med = NULL,
  overrideCache = FALSE, outFormats = c("pdf", "png"), pngRes = 100)
```

Arguments

ssid	The MaveDB ScoreSet ID of the dataset to be visualized
uniprotId	The UniprotKB accession for the corresponding protein
pdbs	A vector of PDB accessions to be used for the structure tracks
mainChains	the chain identifiers corresponding to the protein in question for each provided pdb accession.
wt.seq	An (optional) wild-type sequence for the given protein to use instead of the sequence found in UniprotKB. This can be either a nucleotide or amino acid sequence.
seq.offset	A parameter describing the start position of the map in the protein's amino acid sequence. For example, if only a domain was scanned.
syn.med	The median value of synonymous variants as an estimate of wild-type function. This is only necessary of no synonymous variants are present in the dataset.
stop.med	The median value of nonsense variants as an estimate of complete loss of function. This is only necessary if no nonsense variants are present in the dataset.
overrideCache	defaults to FALSE. If set to TRUE, data will always be re-downloaded from remote locations instead of using a local cache.
outFormats	a vector containing any of the following strings: x11, pdf, png . Using two or all three at once is allowed and will result in multiple output files.
pngRes	Resolution of PNG output in DPI. Defaults to 100.

Value

the job ID

---

find.pdbs *Find PDB structures for a Uniprot accession*

Description

Finds PDB structures that contain the protein indicated by the given Uniprot accession. Checks for a local cache file of previous results. If such a cache exists, the pre-calculated results will be returned, otherwise queries to Uniprot and PDB will be made.

Usage

```
find.pdbs(acc)
```

Arguments

acc	the Uniprot accession

Value

a data.frame with the following columns:

- pdb: the PDB accession of the structure
- method: the experimental method for this structure, e.g NMR, X-ray or Model
- resolution: the resolution of this structure, in Angstrom.
- mainChains: a /-separated list of chain IDs that correspond to the protein with the given Uniprot accession.
- start: the first amino acid of the protein represented in the structure
- end: the last amino acid of the protein represented in the structure
- partners: a comma-separated list of interaction partners if the structure is of a complex. Each item follows the syntax chainID=UniprotID/ProteinName

---

genophenogram *Draw genophenogram plot*

Description

Draws a genophenogram plot from given data

Usage

```
genophenogram(wt.aa, pos, mut.aa, score, syn.med, stop.med, error = NULL,
  a = 0, grayBack = FALSE, img.width = 12, tracks = NULL)
```

Arguments

wt.aa	wildtype amino acid sequence as a vector of single characters.
pos	vector of amino acid positions
mut.aa	vector of mutant AAs
score	vector of scores
error	vector of stderr values
a	bezier transformation intensity (with -0.5 <= a <= 0.5)
grayBack	draw a gray background for incomplete maps
img.width	optional parameter to inform the drawing function of the chosen image width, allowing it to adjust the size of the legend
tracks	an optional trackdrawer object to add structural information to the plot

getCacheFile *Find cache file location by name*

Description

Finds the location for a cache file. The file does not necessary need to exist yet, as this function is meant to be used determine to a location for both storage and retrieval.

Usage

```
getCacheFile(name)
```

Arguments

name	the name of the file, e.g. "P12456_alignment.fasta"

Details

Depending on the execution context, the storage location may differ. The cache location can be controlled with the environment variable $MAVECACHE. This will be made use of within the mavevis server docker container. If the variable is not set, a directory ".mavecache/" will be created in the user's home directory to be used as the storage location.

Value

the full path to the file

Examples

```
file <- getCacheFile("P12345_alignment.fasta")
```

getUniprotSeq *Retrieve Uniprot Sequence*

Description

Retrieves the amino acid sequence for the protein indicated by a Uniprot accession.

Usage

```
getUniprotSeq(uniprot.acc)
```

Arguments

uniprot.acc	the accession

Value

the amino acid sequence

`makeUUID` *Create universally unique ID (UUIDv4)*

Description

Creates a universally unique identifier compatible with the UUID v4.0 standard. See https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

Usage

```
makeUUID()
```

Value

the UUID as a character string

`new.amasLite` *Calculate AMAS conservation*

Description

Uses the AMAS method (Livingstone & Barton 1993) method to calculate position-wise conservation from a protein multiple sequence alignment. Creates an object of type amasLite, which features a single method: `run(alignmentFile)`.

Usage

```
new.amasLite()
```

Details

The `run(alignmentFile)` method takes the name of a file as its parameter. The file must be a FASTA formatted multiple sequence alignment. The method returns a numerical vector listing the position-wise conservation with respect to the first entry in the alignment.

Value

an object of type amasLite

new.structure *Create new Structure object*

Description

Constructor for a PDB structure object.

Usage

```
new.structure(pdb.file)
```

Arguments

pdb.file	The PDB structure file to use

Details

The resulting object offers the following methods:

- get.sequence(chain) Returns the amino acid sequence of the given chain
- get.aa(chain,pos) Returns a data.frame with the atomic information for the amino acid determined by the given chain and position.
- get.chains() Returns a vector detailing the chain indentifiers in the structure
- get.atom(chain,pos,name) Returns the atomic information associated with the given chain, amino acid position, and atom name.
- subcomplex.combos(chain) !Deprecated! Does not work with NMR structures
- get.chain.info() Returns a dataframe listing which chain corresponds to which protein.

Value

a PDB structure object

new.trackdrawer *New Trackdrawer object*

Description

This constructor creates a new Trackdrawer object. The object can be used to draw a plot containing amino-acid resolution structural information tracks, such as conservation, secondary structure, solvent accessibility or burial.

Usage

```
new.trackdrawer(l, nox = FALSE)
```

Arguments

l	the length of the tracks (i.e. the number of amino acids)
nox	no x-axis. Removes the x-axis and decreases the bottom margin to 0. This option is for use with the `layout()` function, to support adding tracks on top of an existing graph that uses amino acid postion as the x-axis.

Details

The object has the following methods:

- `add.track(values,label,col,minVal=0,maxVal=max(values))` adds a new numerical track, which will be visualized in heatmap style. `values` are the numerical values for the heatmap. `label` is the label that will be shown on the y-axis for this track. `col` is the color that will be used to generate the color ramp (from white to `col`). `minVal` is optional and is the minimum value that will be mapped to the color white (default 0). `maxVal` is also optional and is the maximum value in the scale, which will be mapped to `col`.
- `add.constrack(values)` adds a new conservation track, which will be visualized as a barplot. `values` are the conservation values to use.
- `add.ss.track(values)` adds a new secondary structure track, which will be visualized using spriral and arrow symbols. `values` are the character values representing the secondary structure type for each amino acid. Permissible values are: "AlphaHelix","H","G","I" for alpha helices, "Strand","E" for beta strands and "Disorder" for disordered regions. Any other string is interpreted as no specific secondary structure.
- `draw()` draws the plot
- `num.tracks()` returns the number of tracks currently added to the plot.

Value

an object of type TrackDrawer.

Examples

```
## Not run:
td <- new.trackdrawer(50)
td$add.track(runif(50,0,1),"SASA","blue")
td$add.constrack(runif(50,0,11))
td$add.ss.track(sample(c("AlphaHelix","Strand",""),50,replace=TRUE))
td$draw()

## End(Not run)
```

pdb.informative *Select smallest informative subset of PDB structures*

Description

Given the results of `find.pdbs()`, this function finds the smallest informative subset among them. That is, the smallest set of PDB structures, that still represent all available interaction partners.

Usage

```
pdb.informative(pdb.table)
```

Arguments

`pdb.table`	The result of `find.pdbs()`

Value

a vector with the IDs of the selected PDB structures.

query.pdb *Query PDB*

Description

Queries the Protein Data Bank (PDB) for protein structure data for a given database accession and writes the result to file

Usage

```
query.pdb(pdb.acc, pdb.file = paste0(pdb.acc, ".pdb"))
```

Arguments

`pdb.acc`	the PDB accession to query
`pdb.file`	the name of the file to which the output will be written.

Value

the name of the output file.

Examples

```
## Not run:
#with a pre-determined output file
outfile <- "sumo_conjugase_structure.pdb"
query.pdb("3UIP",outfile)

#with an autogenerated output file
outfile <- query.pdb("3UIP")

## End(Not run)
```

`retrieve.async.result.file` *Retrieve result file from asynchronous job*

Description

Retrieves the output file from an asynchronous job if it exists. The file is copied to the working directory, so that OpenCPU can expose it on the webservice.

Usage

```
retrieve.async.result.file(jobID, ext = "png")
```

Arguments

`jobID`	the ID of the job.
`ext`	the file extension. Defaults to "png".

Value

`TRUE` if successful, otherwise `FALSE`.

`run.dssp` *Run DSSP on a PDB structure*

Description

Calculates secondary structure information from a PDB file using the external software DSSP, which must be installed and available via $PATH

Usage

```
run.dssp(pdb.file)
```

Arguments

pdb.file	The PDB file on which to run DSSP

Value

a data.frame with the output of DSSP

Examples

```
## Not run:
dssp.out <- run.dssp(pdb.file)

## End(Not run)
```

---

run.sasa *Run FreeSASA on a PDB structure*

Description

Calculates solvent accessible surface area on a given structure using the external software FreeSASA, which must be installed and available via $PATH.

Usage

```
run.sasa(pdb.file)
```

Arguments

pdb.file	The PDB file on which to run FreeSASA

Value

a data.frame with the output of FreeSASA

Examples

```
## Not run:
sasa <- run.sasa("3UIP.pdb")

## End(Not run)
```

`subcomplex.combos` *Generate PDB files for complex components*

Description

This function finds the component chains of a given PDB structure and generates new PDB files for any desired set of combinations of these components

Usage

```
subcomplex.combos(pdb.file, chain.sets)
```

Arguments

`pdb.file`	the input PDB file for the complex
`chain.sets`	a list of character vectors, detailing the combinations of chains to produce. E.g. `list("A",c("A","B"),c("A","C"))` produces one file for chain A, one for the combination of chains A and B, and one for the combination of chains A and C.

Value

a vector of file names corresponding to the elements of `chain.sets`

Examples

```
## Not run:
subcplx.files <- sub.complex.combos("3UIP.pdb",chain.sets=list("A",c("A","B"),c("A","C")))

## End(Not run)
```

Index