Package 'biogram'

June 22, 2015

Type Package

Title N-Gram Analysis of Biological Sequences

Version 1.2

LazyData true

Date 2015-04-02

Description Tools for extraction and analysis of various
n-grams (k-mers) derived from biological sequences (proteins
or nucleic acids). Uses QuiPT (quick permutation test) for fast
feature-filtering to deal with the dimensionality of the n-gram data.

License GPL-3

URL https://github.com/michbur/biogram

BugReports https://github.com/michbur/biogram/issues

Depends R (>= 3.0.0), slam

Imports bit, entropy

Suggests ggplot2, testthat

NeedsCompilation no

Repository CRAN

Author Michal Burdukiewicz [cre, aut],
Piotr Sobczyk [aut],
Chris Lauber [aut]

Maintainer Michal Burdukiewicz <michalburdukiewicz@gmail.com>

Date/Publication 2015-06-22 12:59:21

R topics documented:

`biogram-package` *biogram - analysis of biological sequences using n-grams*

Description

`biogram` package for the analysis of nucleic acid and protein sequences using n-grams. Possible applications include motif discovery, feature selection, clustering, and classification.

n-grams

n-grams (k-tuples) are sets of `n` characters derived from the input sequence(s). They may form continuous sub-sequences or be discontinuous. For example, from the sequence of nucleotides `AATA` one can extract the following continuous 2-grams (bigrams): `AA`, `AT` and `TA`. Moreover, there are two possible bigrams separated by a single space: `A_T` and `A_A`, and one bigram separated by two spaces: `A__A`.

Another important n-gram parameter is its position. Instead of just counting n-grams, one may want to count how many n-grams occur at a given position in multiple (e.g. related) sequences. For example, in the sequences AATA and AACA there is only one bigram at position 1: AA, but there are two bigrams at position two: AT and AC. The following notation is used for position-specific n-grams: 1_AA, 2_AT, 2_AC.

In the biogram package, the count_ngrams function is used for counting and extracting n-grams. Using the d argument the user can specify the distance between elements of the n-grams. The pos argument can be used to enable position specificity.

n-gram data dimensionality

We note that n-grams suffer from the curse of dimensionality. For example, for a peptide of length 6 20^n n-grams and 6×20^n positioned n-grams are possible. Data sets of such an enormous size are hard to manage and analyze in R.

The biogram package deals with both of the abovementioned problems. It uses innate properties of the n-gram data which usually can be represented by sparse matrices. Data storage is done using functionalities from the slam package. To ease the selection of significant features, biogram provides the user with QuiPT, a very fast permutation test for binary data (see test_features).

Another way of reducing dimensionality is the aggregation of sequence residues into more general groups. For example, all positively-charged amino acids may be aggregated into one group. This action can be performed using the degenerate function.

Author(s)

Michal Burdukiewicz, Piotr Sobczyk, Chris Lauber

Examples

```
#use data set from package
data(human_cleave)
#first nine columns represent subsequent nine amino acids from cleavage sites
#degenerate the sequence to reduce the dimensionality of the problem
#(use five groups instead of 20 amino acids)
deg_seqs <- degenerate(human_cleave[, 1L:9],
                       list(`a` = c(1, 6, 8, 10, 11, 18),
                            `b` = c(2, 13, 14, 16, 17),
                            `c` = c(5, 19, 20),
                            `d` = c(7, 9, 12, 15),
                            'e' = c(3, 4)))
#EXAMPLE 1 - extract significant trigrams
#extract trigrams
trigrams <- count_ngrams(deg_seqs, 3, letters[1L:5], pos = TRUE)
#select features that differ between the two target groups using QuiPT
test1 <- test_features(human_cleave[, "tar"], trigrams)
#see a summary of the results
summary(test1)
#aggregate features in groups based on their p-value
gr <- cut(test1)
#get position map of the most significant n-grams
position_ngrams(gr[[1]])
```

```
#transform the most significant n-grams to more readable form
decode_ngrams(gr[[1]])

#EXAMPLE 2 - search for specific n-grams
#the n-grams of the interest are a_a (a-gap-a) and e_e (e-gap-e) on the
#3rd and 4th position
#firstly code n-grams in biogram notation and add position information
coded <- code_ngrams(c("a_a", "c_c"))
#add position information
coded <- c(paste0("3_", coded), paste0("4_", coded))
#count only the features of the interest
bigrams <- count_specified(deg_seqs, coded)
#test which of the features of the interest is significant
test2 <- test_features(human_cleave[, "tar"], bigrams)
cut(test2)
```

`add_1grams` *Add 1-grams*

Description

Builds (n+1)-grams from n-grams.

Usage

```
add_1grams(ngrams)
```

Arguments

`ngrams`	a vector of positioned n-grams (as created by `count_ngrams`).

Details

n-grams are built by pasting existing n-grams with unigrams extracted from them.

Value

a vector of n-grams (where `n` is equal to the `n` of the input plus one) with position information.

Note

All n-grams must have the same length (`n`).

See Also

Function used by `add_1grams` to extract unigrams: `position_ngrams`.

Examples

```
add_1grams(c("1_1_0", "2_1_0", "5_1_0", "7_1_0", "4_2_0",
"5_2_0", "7_2_0", "8_5_0"))
add_1grams(c("1_2.3.4_0.0", "4_1.1.1_0.0"))
```

---

calc_criterion *Calculate criterion*

Description

Calculates independently chosen statistical criterion for each feature versus target vector.

Usage

```
calc_criterion(target, features, criterion_function)
```

Arguments

`target`	`integer` vector with target information (e.g. class labels).
`features`	`integer` matrix of features with number of rows equal to the length of the target vector.
`criterion_function`	a function calculating criterion. For a full list, see See also.

Details

Permutation test implemented in `biogram` uses several criterions to filter important features. Each can be used by `test_features` by specifying `criterion` parameter.

Possible criterions are:

ig Information Gain. Calculated using `calc_ig`.

kl Kullback-Leibler divergence. Calculated using `calc_kl`.

Value

a `integer` vector of length equal to the number of features containing computed information gain values.

Note

Both `target` and `features` must be binary, i.e. contain only 0 and 1 values.

See Also

`test_features`.

Examples

```
calc_criterion(sample(0L:1, 100, replace = TRUE),
               matrix(sample(0L:1, 400, replace = TRUE), ncol = 4),
               calc_ig)
```

`calc_ig` *Calculate IG for single feature*

Description

Computes information gain of single feature and target vector.

Usage

```
calc_ig(feature, target_b, len_target, pos_target, ES)
```

Arguments

`feature`	feature vector.
`target_b`	target in bits (as per `as.bit`).
`len_target`	length of target vector.
`pos_target`	number of positive cases in target vector.
`ES`	numeric value of target entropy.

Details

The information gain term is used here (improperly) as a synonym of mutual information. It is defined as:

$$IG(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right)$$

In biogram package information gain is calculated using following relationship: $IG = E(S) - E(S|F)$

Value

a single numeric value - information gain in nats.

Note

During calculations $0 \log 0 = 0$. For justification see References.

Input looks strange, but the function was designed to be as fast as possible subroutine of `calc_ig` and generally should not be directly called by user.

References

Cover TM, Thomas JA *Elements of Information Theory, 2nd Edition* Wiley, 2006.

Examples

```
tar <- sample(0L:1, 100, replace = TRUE)
feat <- sample(0L:1, 100, replace = TRUE)
prop <- c(100 - sum(tar), sum(tar))/100
entr <- - sum(prop*log(prop))
library(bit) #used to code vector as bit
calc_ig(feat, as.bit(tar), 100, sum(tar), entr)
```

`calc_kl` *Calculate KL divergence of features*

Description

Computes Kullback-Leibler divergence between features and target vector.

Usage

```
calc_kl(feature, target_b, len_target, pos_target, ES)
```

Arguments

`feature`	feature vector.
`target_b`	target in bits (as per `as.bit`).
`len_target`	length of target vector.
`pos_target`	number of positive cases in target vector.
`ES`	numeric value of target entropy.

Value

a `integer` vector of length equal to the number of features containing computed information gain values.

Note

Both `target` and `features` must be binary, i.e. contain only 0 and 1 values.

References

STH here

See Also

`test_features`.

Kullback-Leibler divergence is calculated using `KL.plugin`.

Examples

```
tar <- sample(0L:1, 100, replace = TRUE)
feat <- sample(0L:1, 100, replace = TRUE)
prop <- c(100 - sum(tar), sum(tar))/100
entr <- - sum(prop*log(prop))
library(bit) #used to code vector as bit
calc_kl(feat, as.bit(tar), 100, sum(tar), entr)
```

---

code_ngrams *Code n-grams*

Description

Code human-friendly representation of n-grams into a biogram format.

Usage

```
code_ngrams(decoded_ngrams)
```

Arguments

decoded_ngrams	a character vector of decoded n-grams.

Value

a character vector of n-grams.

See Also

Inverse function: decode_ngrams.

Examples

```
code_ngrams(c("11_2", "1__12", "222"))
code_ngrams(c("aaa_b", "d__aa", "abd"))
```

`construct_ngrams` *Construct and filter n-grams*

Description

Step-by-step builds and selects important n-grams. `construct_ngrams` extracts unigrams from the sequences and filters significant features (with p-value below `conf_level`).

Usage

```
construct_ngrams(target, seq, u, n_max, conf_level = 0.95, gap = TRUE)
```

Arguments

`target`	`integer` vector with target information (e.g. class labels).
`seq`	a vector or matrix describing sequence(s).
`u`	`integer`, `numeric` or `character` vector of all possible unigrams.
`n_max`	size of constructed n-grams.
`conf_level`	confidence level.
`gap`	`logical`, if `TRUE` gap are used. See Details.

Details

Gap parameter determines if `construct_ngrams` performs feature selection on exact n-grams (`gap` equal to FALSE) or on all features in the Hamming distance 1 from the n-gram (`gap` equal to TRUE).

Value

a vector of n-grams.

See Also

Feature filtering method: `test_features`.

Examples

```
deg_seqs <- degenerate(human_cleave[, 1L:9],
list(`1` = c(1, 6, 8, 10, 11, 18),
     `2` = c(2, 13, 14, 16, 17),
     `3` = c(5, 19, 20),
     `4` = c(7, 9, 12, 15),
     '5' = c(3, 4)))
bigrams <- construct_ngrams(human_cleave[, "tar"], deg_seqs, 1L:5, 2)
```

`count_multigrams` *Detect and count multiple n-grams in sequences*

Description

A convinient wrapper around `count_ngrams` for counting multiple values of `n` and `d`.

Usage

```
count_multigrams(ns, ds = rep(0, length(ns)), seq, u, pos = FALSE,
  scale = FALSE, threshold = 0)
```

Arguments

`ns`	`numeric` vector of n-grams' sizes. See Details.
`ds`	`list` of distances between elements of n-grams. Each element of the list is a vector used as distance for the respective n-gram size given by the `ns` parameter.
`seq`	a vector or matrix describing sequence(s).
`u`	`integer`, `numeric` or `character` vector of all possible unigrams.
`pos`	`logical`, if `TRUE` position-specific n_grams are counted.
`scale`	`logical`, if `TRUE` output data is normalized. Should be used only for n-grams without position information. See `Details`.
`threshold`	`integer`, if not equal to 0, data is binarized into two groups (larger or equal to threshold vs. smaller than threshold).

Details

`ns` vector and `ds` vector must have equal length. Elements of `ds` vector are used as equivalents of `d` parameter for respective values of `ns`. For example, if `ns` is `c(4, 4, 4)`, the `ds` must be a list of length 3. Each element of the `ds` list must have length 3 or 1, as appropriate for a `d` parameter in `count_ngrams` function.

Value

a `integer` matrix with named columns. The naming conventions are the same as in `count_ngrams`.

Examples

```
seqs <- matrix(sample(1L:4, 600, replace = TRUE), ncol = 50)
count_multigrams(c(3, 1), list(c(1, 0), 0), seqs, 1L:4, pos = TRUE)
#if ds parameter is not present, n-grams are calculated for distance 0
count_multigrams(c(3, 1), seq = seqs, u = 1L:4)

#calculate three times n-gram with the same length, but different distances between
#elements
count_multigrams(c(4, 4, 4), list(c(2, 0, 1), c(2, 1, 0), c(0, 1, 2)),
                 seqs, 1L:4, pos = TRUE)
```

count_ngrams *Count n-grams in sequences*

Description

Counts all n-grams or position-specific n-grams present in the input sequence(s).

Usage

```
count_ngrams(seq, n, u, d = 0, pos = FALSE, scale = FALSE,
  threshold = 0)
```

Arguments

seq	a vector or matrix describing sequence(s).
n	integer size of n-gram.
u	integer, numeric or character vector of all possible unigrams.
d	integer vector of distances between elements of n-gram (0 means consecutive elements). See Details.
pos	logical, if TRUE position-specific n_grams are counted.
scale	logical, if TRUE output data is normalized. Should be used only for n-grams without position information. See Details.
threshold	integer, if not equal to 0, data is binarized into two groups (larger or equal to threshold vs. smaller than threshold).

Details

A distance vector should be always n - 1 in length. For example when n = 3, d = c(1,2) means A_A__A. For n = 4, d = c(2,0,1) means A__AA_A. If vector d has length 1, it is recycled to length n - 1.

n-gram names follow a specific convention and have three parts for position-specific n-grams and two parts otherwise. The parts are separated by _. The . symbol is used to separate elements within a part. The general naming scheme is POSITION_NGRAM_DISTANCE. The optional POSITION part of the name indicates the actual position of the n-gram in the sequence(s) and will be present only if pos = TRUE. This part is always a single integer. The NGRAM part of the name is a sequence of elements in the n-gram. For example, 4.2.2 indicates the n-gram 422 (e.g. TCC). The DISTANCE part of the name is a vector of distance(s). For example, 0.0 indicates zero distances (continuous n-grams), while 1.2 represents distances for the n-gram A_A__A.

Examples of n-gram names:

- 46_4.4.4_0.1 : trigram 44_4 on position 46
- 12_2.1_2 : bigram 2__1 on position 12
- 8_1.1.1_0.0 : continuous trigram 111 on position 8
- 1.1.1_0.0 : continuous trigram 111 without position information

Value

a simple_triplet_matrix where columns represent n-grams and rows sequences. See Details for specifics of the naming convention.

Note

By default, the counted n-gram data is stored in a memory-saving format. To convert an object to a 'classical' matrix use the as.matrix function. See examples for further information.

See Also

Create vector of possible n-grams: create_ngrams.

Extract n-grams from sequence(s): seq2ngrams.

Get indices of n-grams: get_ngrams_ind.

Count n-grams for multiple values of n: count_multigrams.

Count only specified n-grams: count_specified.

Examples

```
#count trigrams without position information for nucleotides
count_ngrams(sample(1L:4, 50, replace = TRUE), 3, 1L:4, pos = FALSE)
#count position-specific trigrams from multiple nucleotide sequences
seqs <- matrix(sample(1L:4, 600, replace = TRUE), ncol = 50)
ngrams <- count_ngrams(seqs, 3, 1L:4, pos = TRUE)
#output results of the n-gram counting to screen
as.matrix(ngrams)
```

---

count_specified *Count specified n-grams*

Description

Counts specified n-grams in the input sequence(s).

Usage

```
count_specified(seq, ngrams)
```

Arguments

seq	a vector or matrix describing sequence(s).
ngrams	a vector of n-grams. Must have the same n.

Details

count_specified counts only selected n-grams declared by user in the ngrams parameter. Declared n-grams must be written using the biogram notation.

Value

a simple_triplet_matrix where columns represent n-grams and rows sequences.

See Also

Count all possible n-grams: count_ngrams.

Examples

```
seqs <- matrix(c(1, 2, 2, 1, 2, 1, 2, 1, 1, 2, 3, 4, 1, 2, 2, 4), nrow = 2)
count_specified(seqs, ngrams = c("1.1.1_0.0", "2.2.2_0.0", "1.1.2_0.0"))

seqs <- matrix(sample(1L:5, 200, replace = TRUE), nrow = 20)
count_specified(seqs, ngrams = c("2_4.2_0", "2_1.4_0", "3_1.3_0",
                                 "2_4.2_1", "2_1.4_1", "3_1.3_1",
                                 "2_4.2_2", "2_1.4_2", "3_1.3_2"))
```

---

count_total *Count total number of n-grams*

Description

Computes total number of n-grams that can be extracted from sequences taking into account their length (even or uneven).

Usage

```
count_total(seq, n, d)
```

Arguments

seq	a vector or matrix describing sequence(s).
n	integer size of n-gram.
d	integer vector of distances between elements of n-gram (0 means consecutive elements). See Details.

Details

A format of d vector is discussed in Details of count_ngrams.

Value

A number of n-grams.

Examples

```
seqs <- matrix(sample(1L:4, 600, replace = TRUE), ncol = 50)
#make several sequences shorter by replacing them partially with NA
seqs[8L:11, 46L:50] <- NA
seqs[1L, 31L:50] <- NA
count_total(seqs, 3, c(1, 0))
```

---

create_feature_target *Create feature according to given contingency matrix*

Description

Creates a matrix of features and target based on the values from contingency matrix.

Usage

```
create_feature_target(n11, n01, n10, n00)
```

Arguments

n11	number of elements for which both target and feature equal 1.
n01	number of elements for which target and feature equal 1,0 respectively.
n10	number of elements for which target and feature equal 0,1 respectively.
n00	number of elements for which both target and feature equal 0.

Value

a matrix of 2 columns and n11+n10+n01+n00 rows. Columns represent target and feature vectors, respectively.

Examples

```
#equivalent of
#         target
#feature 10 375
#        15 600
target_feature <- create_feature_target(10, 375, 15, 600)
```

create_ngrams *Get all possible n-Grams*

Description

Creates vector of all posible n_grams (for given n).

Usage

```
create_ngrams(n, u, possible_grams = NULL)
```

Arguments

n	integer size of n-gram.
u	integer, numeric or character vector of all possible unigrams.
possible_grams	number of possible n-grams. If not NULL n-grams do not contain information about position

Details

See Details section of count_ngrams for more information about n-grams naming convention. The possible information about distance must be added by hand (see examples).

Value

a character vector. Elements of n-gram are separated by dot.

Note

Input data must be a matrix or data frame of numeric elements.

Examples

```
#bigrams for standard aminoacids
create_ngrams(2, 1L:20)
#bigrams for standard aminoacids with positions, 10 amino acid long sequence, so
#only 9 bigrams can be located in sequence
create_ngrams(2, 1L:20, 9)
#bigrams for DNA with positions, 10 nucleotide long sequence, distance 1, so only
#8 bigrams in sequence
#paste0 adds information about distance at the end of n-gram
paste0(create_ngrams(2, 1L:4, 8), "_0")
```

criterion_distribution *criterion_distribution class*

Description

A result of `distr_crit` function.

Details

An object of class `criterion_distribution` is a numeric matrix.

Data

1st column: possible values of criterion.

2nd column: probability density function.

3rd column: cumulative distribution function.

Attributes

plot_data A matrix with values of the criterion and their probabilities.

nice_name 'Nice' name of the criterion.

cut.feature_test *Categorize tested features*

Description

Categorizes results of `test_features` function into groups based on their significance.

Usage

```
## S3 method for class 'feature_test'
cut(x, split = "significances", breaks = c(0, 1e-04,
  0.01, 0.05, 1), ...)
```

Arguments

`x`	an object of class `feature_test`.
`split`	attribute along which output should be categorized. Possible values are `"significances"`, `"positives"` and `"negatives"`. See `Value`.
`breaks`	a vector of significances of frequencies along which n-grams are aggregated. See description of `cut` function and `Details`.
`...`	further parameters accepted by the `cut` function.

Value

the value of function depends on the `split` parameter. The function returns a named list of length equal to the length of `significances` (when `split` equals `"significances"`) or `frequencies` (when `split` equals `"positives"` or `"negatives"`) minus one. Each elements of the list contains names of the n-grams belonging to the given significance or frequency group.

---

decode_ngrams *Decode n-grams*

Description

Transforms a vector of n-grams into a human-friendly form.

Usage

```
decode_ngrams(ngrams)
```

Arguments

`ngrams`	a `character` vector of n-grams.

Value

a `character` vector of length equal to the number of n-grams.

Note

Decoded n-grams lose the position information.

See Also

Validate n-gram structure: `is_ngram`.

Inverse function: `code_ngrams`.

Examples

```
decode_ngrams(c("2_1.1.2_0.1", "3_1.1.2_2.0", "3_2.2.2_0.0"))
```

`degenerate` *Degenerate protein sequence*

Description

'Degenerates' amino acid or nucleic sequence by aggregating elements to bigger groups.

Usage

```
degenerate(seq, element_groups)
```

Arguments

`seq`	`character` vector or matrix representing single sequence.
`element_groups`	list of groups to which elements of sequence should be aggregated.

Value

a `character` vector or matrix (if input is a matrix) containing aggregated elements.

Note

Both sequence and `element_groups` should contain lower-case letters. Upper-case will be automatically converted without a message.

Characters not present in the `element_groups` will be converted to NA with a warning.

See Also

`l2n` to easily convert information stored in biological sequences from letters to numbers.

Examples

```
sample_seq <- c(1, 3, 1, 3, 4, 4, 3, 1, 2)
table(sample_seq)

#aggregate sequence to purins and pyrimidines
deg_seq <- degenerate(sample_seq, list(w = c(1, 4), s = c(2, 3)))
table(deg_seq)
```

distr_crit *Compute criterion distribution*

Description

Computes criterion distribution for feature, target under null hypothesis

Usage

```
distr_crit(target, feature, criterion = "ig")
```

Arguments

target	{0,1}-valued target vector. See Details.
feature	{0,1}-valued feature vector. See Details.
criterion	the criterion used for calculations of distribution. See calc_criterion.

Details

both target and feature vectors may contain only 0 and 1.

Value

An object of class criterion_distribution.

See Also

calc_criterion.

Examples

```
target_feature <- create_feature_target(10, 375, 15, 600)
distr_crit(target = target_feature[,1], feature = target_feature[,2])
```

fast_crosstable *Very fast 2d cross-tabulation*

Description

Quickly cross-tabulates two binary vectors.

Usage

```
fast_crosstable(target_b, len_target, pos_target, feature)
```

Arguments

target_b	target in bits (as per as.bit).
len_target	length of target vector.
pos_target	number of positive cases in target vector.
feature	feature vector.

Details

Input looks odd, but the function was build to be as fast as possible subroutine of calc_ig, which works on many features but only one target.

Value

a vector of length four:

1. target +, feature+
2. target +, feature-
3. target -, feature+
4. target -, feature-

Note

Binary vector means numeric vector with 0 or 1.

Examples

```
tar <- sample(0L:1, 100, replace = TRUE)
feat <- sample(0L:1, 100, replace = TRUE)
library(bit) #used to code vector as bit
fast_crosstable(as.bit(tar), length(tar), sum(tar),  feat)
```

---

feature_test *feature_test class*

Description

A result of test_features function.

Details

An object of class feature_test is a numeric vector of p-values. Additional attributes characterizes futher the details of test which returned these p-values.

Attributes

criterion the criterion used in permutation test.

adjust the name of p-value adjusting method

times the number of permutations. If QuiPT was chosen NA.

occ frequency of features splitted in subset based on the value of target.

---

gap_ngrams *Gap n-grams*

Description

Introduces gaps in the n-grams.

Usage

```
gap_ngrams(ngrams)
```

Arguments

ngrams	a vector of positioned n-grams (as created by count_ngrams).

Details

A single element of the input n-gram at a time will be replaced by a gap. For example, introducing gaps in n-gram 2_1.1.2_0.1 will results in three n-grams: 3_1.2_1 (where the 2_1_0 unigram was replaced by a gap), 2_1.2_2 and 2_1.1_0.

Value

A character vector of (n-1)-grams with introduced gaps.

Examples

```
gap_ngrams(c("2_1.1.2_0.1", "3_1.1.2_0.0", "3_2.2.2_0.0"))
```

get_ngrams_ind *Get indices of n-grams*

Description

Computes list of n-gram elements positions in sequence.

Usage

```
get_ngrams_ind(len_seq, n, d)
```

Arguments

len_seq	integer value describing sequence's length.
n	integer size of n-gram.
d	integer vector of distances between elements of n-gram (0 means consecutive elements). See Details.

Details

A format of d vector is discussed in Details of count_ngrams.

Value

A list with number of elements equal to n. Every element is a vector containing locations of given n-gram letter. For example, first element of list contain indices of first letter of all n-grams. The attribute d of output contains distances between letter used to compute locations (see Details).

Examples

```
#positions trigrams in sequence of length 10
get_ngrams_ind(10, 9, 0)
```

human_cleave *Human signal peptides cleavage sites*

Description

A set of 648 cleavage sites and 648 parts of mature proteins shortly after cleavage sites derived from human proteome.

Format

A data frame with 1296 observations on the following 10 variables. Columns from P1 to P9 describes positions in an extracted peptide. tar is a target vector. It has value 1 if a peptide is a cleavage site and 0 if not.

Details

Each peptide in the data set is nine amino acid residues long. In case of cleavage sites, the clevage is located between fifth and sixth peptide. The non-cleavage sites are parts of mature proteins starting five positions after cleavage site.

Note

Amino acid residues were recoded as integers.

Source

UniProt

Examples

```
data(human_cleave)
table(human_cleave[, 1])
```

`is_ngram` *Validate n-gram*

Description

Checks if the character string may be used as an n-gram and its notation follows specific convention of `biogram` package.

Usage

```
is_ngram(x)
```

Arguments

`x`	a `character` string representing single n-gram.

Value

`TRUE` if n-gram's notation is correct, `FALSE` if not.

Examples

```
print(is_ngram("1_1.1.1_0.0"))
print(is_ngram("not_ngram"))
```

l2n *Convert letters to numbers*

Description

Converts biological sequence from letter to number notation.

Usage

```
l2n(seq, seq_type)
```

Arguments

seq	character vector representing single sequence.
seq_type	the type of sequence. Can be rna, dna or prot.

Value

a numeric vector containing converted elements.

See Also

l2n is a wrapper around degenerate.

Inverse function: n2l.

Examples

```
sample_seq <- c("a", "d", "d", "g", "a", "g", "n", "a", "l")
l2n(sample_seq, "prot")
```

n2l *Convert numbers to letters*

Description

Converts biological sequence from number to letter notation.

Usage

```
n2l(seq, seq_type)
```

Arguments

seq	numeric vector representing single sequence.
seq_type	the type of sequence. Can be rna, dna or prot.

Value

a numeric vector containing converted elements.

See Also

n2l is a wrapper around degenerate.

Inverse function: l2n.

Examples

```
sample_seq <- c(1, 3, 3, 6, 1, 6, 12, 1, 10)
n2l(sample_seq, "prot")
```

---

ngrams2df *n-grams to data frame*

Description

Tranforms a vector of n-grams into a data frame.

Usage

```
ngrams2df(ngrams)
```

Arguments

ngrams	a character vector of n-grams.

Value

a data.frame with 2 (in case of n-grams without known position) or three columns (n-grams with position information).

See Also

Decode n-grams: decode_ngrams.

Examples

```
ngrams2df(c("2_1.1.2_0.0", "3_1.1.2_0.0", "3_2.2.2_0.0", "2_1.1_0"))
```

`plot.criterion_distribution` *Plot criterion distribution*

Description

Plots results of `distr_crit` function.

Usage

```
## S3 method for class 'criterion_distribution'
plot(x, ...)
```

Arguments

`x`	object of class `criterion_distribution`.
`...`	further arguments passed to `plot`.

Value

nothing.

Examples

```
target_feature <- create_feature_target(10, 375, 15, 600)
example_result <- distr_crit(target = target_feature[,1],
                             feature = target_feature[,2])
plot(example_result)

#a ggplot2 plot
library(ggplot2)
ggplot_distr <- function(x) {
b <- data.frame(cbind(x=as.numeric(rownames(attr(x, "plot_data"))),
                      attr(x, "plot_data")))
d1 <- cbind(b[,c(1,2)], attr(x, "nice_name"))
d2 <- cbind(b[,c(1,3)], "Probability")
colnames(d1) <- c("x", "y", "panel")
colnames(d2) <- c("x", "y", "panel")
d <- rbind(d1, d2)
p <- ggplot(data = d, mapping = aes(x = x, y = y)) +
  facet_grid(panel~., scale="free") +
  geom_freqpoly(data= d2, aes(color=y), stat = "identity") +
  scale_fill_brewer(palette = "Set1") +
  geom_point(data=d1, aes(size=y), stat = "identity") +
  guides(color = "none") +
  guides(size = "none") +
  xlab("Number of cases with feature=1 and target=1") + ylab("")
p
}
ggplot_distr(example_result)
```

position_ngrams *Position n-grams*

Description

Tranforms a vector of positioned n-grams into a list of positions filled with n-grams that start on them.

Usage

```
position_ngrams(ngrams, df = FALSE, unigrams_output = TRUE)
```

Arguments

ngrams	a vector of positioned n-grams (as created by count_ngrams).
df	logical, if TRUE returns a data frame, if FALSE returns a list.
unigrams_output	logical, if TRUE extracts unigrams from the data and returns information about their position.

Value

if df is FALSE, returns a list of length equal to the number of unique n-gram starts present in n-grams. Each element of the list contains n-grams that start on this position. If df is FALSE, returns a data frame where first column contains n-grams and the second column represent their start positions.

See Also

Transform n-gram name to human-friendly form: decode_ngrams.

Validate n-gram structure: is_ngram.

Examples

```
#position data in the list format
position_ngrams(c("2_1.1.2_0.1", "3_1.1.2_0.0", "3_2.2.2_0.0"))
#position data in the data frame format
position_ngrams(c("2_1.1.2_0.1", "3_1.1.2_0.0", "3_2.2.2_0.0"), df = TRUE)
```

print.feature_test *Print tested features*

Description

Prints results of test_features function.

Usage

```
## S3 method for class 'feature_test'
print(x, ...)
```

Arguments

x	object of class feature_test.
...	further arguments passed to print.default.

Value

nothing.

seq2ngrams *Extract n-grams from sequence*

Description

Extracts vector of n-grams present in sequence(s).

Usage

```
seq2ngrams(seq, n, u, d = 0, pos = FALSE)
```

Arguments

seq	a vector or matrix describing sequence(s).
n	integer size of n-gram.
u	integer, numeric or character vector of all possible unigrams.
d	integer vector of distances between elements of n-gram (0 means consecutive elements). See Details.
pos	logical, if TRUE position-specific n_grams are counted.

Details

A format of d vector is discussed in Details of count_ngrams.

Value

A `character` matrix of n-grams, where every row corresponds to a different sequence.

Examples

```
#trigrams from multiple sequences
seqs <- matrix(sample(1L:4, 600, replace = TRUE), ncol = 50)
seq2ngrams(seqs, 3, 1L:4)
```

---

summary.feature_test *Summarize tested features*

Description

Summarizes results of `test_features` function.

Usage

```
## S3 method for class 'feature_test'
summary(object, conf_level = 0.95, ...)
```

Arguments

`object`	of class `feature_test`.
`conf_level`	confidence level. A feature with p-value equal to or smaller than the confidence is considered significant.
`...`	ignored

Value

nothing.

---

table_ngrams *Tabulate n-grams*

Description

Builds a contingency table of the n-gram counts versus their class labels.

Usage

```
table_ngrams(seq, ngrams, target)
```

Arguments

seq	a matrix describing sequences.
ngrams	a vector of n-grams. Must have the same n.
target	integer vector with target information (e.g. class labels).

Value

a data frame with the number of columns equal to the length of the target plus 1. The first column contains names of the n-grams. Further columns represents counts of n-grams for respective value of the target.

Examples

```
seqs_pos <- matrix(sample(c("a", "c", "g", "t"), 100, replace = TRUE,
            prob = c(0.2, 0.4, 0.35, 0.05)), ncol = 5)
seqs_neg <- matrix(sample(c("a", "c", "g", "t"), 100, replace = TRUE),
            ncol = 5)
tab <- table_ngrams(seq = rbind(seqs_pos, seqs_neg),
                ngrams = c("1_c.t_0", "1_g.g_0", "2_t.c_0", "2_g.g_0", "3_c.c_0", "3_g.c_0"),
                    target = c(rep(1, 20), rep(0, 20)))
#see the results
print(tab)
#easily plot the results using ggplot2
```

---

test_features *Permutation test for feature selection*

Description

Performs a feature selection on positioned n-gram data using a Fisher's permutation test.

Usage

```
test_features(target, features, criterion = "ig", adjust = "BH",
  threshold = 1, quick = TRUE, times = 1e+05)
```

Arguments

target	integer vector with target information (e.g. class labels).
features	integer matrix of features with number of rows equal to the length of the target vector.
criterion	criterion used in permutation test. See calc_criterion for the list of possible criterions.
adjust	name of p-value adjustment method. See p.adjust for the list of possible values. If NULL, no adjustment is done.

threshold	integer. Features that occur less than threshold and more often than nrow(features)-threshold are discarded from the permutation test.
quick	logical, if TRUE Quick Permutation Test (QuiPT) is used.
times	number of times procedure should be repeated. Ignored if quick is TRUE.

Details

Since the procedure involves multiple testing, it is advisable to use one of the avaible p-value adjustment methods. Such methods can be used directly by specifying the adjust parameter.

Value

an object of class feature_test.

Note

Both target and features must be binary, i.e. contain only 0 and 1 values.

Features occuring too often and too rarely are considered not informative and may be removed using the threshold parameter.

References

Radivojac P, Obradovic Z, Dunker AK, Vucetic S, *Feature selection filters based on the permutation test* in Machine Learning: ECML 2004, 15th European Conference on Machine Learning, Springer, 2004.

See Also

calc_criterion - computes selected criterion.

distr_crit - distribution of criterion used in QuiPT.

summary.feature_test - summary of results.

cut.feature_test - aggregates test results in groups based on feature's p-value.

Examples

```
#significant feature
tar_feat1 <- create_feature_target(10, 390, 0, 600)
#significant feature
tar_feat2 <- create_feature_target(9, 391, 1, 599)
#insignificant feature
tar_feat3 <- create_feature_target(198, 202, 300, 300)
test_res <- test_features(tar_feat1[, 1], cbind(tar_feat1[, 2], tar_feat2[, 2],
                          tar_feat3[, 2]))
summary(test_res)
cut(test_res)
```

Index