Package 'wspaces'

September 20, 2017

Type Package

Title Build and mnipulate word spaces (a.k.a. word embedding models).

Version 0.1.0

Author J. T. Atria

Maintainer J. T. Atria <jtatria@gmail.com>

Description wspaces packages a number of useful functions for distributional semantic modelling, a set of interface functions for a Lucene index backend and a set of fuctions to enforce some reasonable conventions for lexical datasets and the necessary file formats to store and read them.

License GPL v3.0

Encoding UTF-8

LazyData true

Imports Rcpp (>= 0.12.11),
RcppParallel,
Matrix,
magrittr,
data.table,
rJava,
igraph

LinkingTo Rcpp,
RcppEigen,
RcppParallel

RoxygenNote 6.0.1

SystemRequirements Java, GNU make

R topics documented:

classify 2
class_mass 3
class_rank 4
coef_bhattacharyya 4
dist_hellinger 5

div_bhattacharyya 6
div_jensen_shannon 6
div_kulback_leibler 7
div_setratio 8
graph_connected 9
graph_prune_connected 9
graph_weight_cv 10
graph_weight_vc 11
idf 11
is.real 12
lexical_dataset 12
lexical_join 13
load_corpus 13
load_spm 14
obo_all_pos 15
obo_count_cooc 15
obo_get_fields 16
obo_get_terms 16
obo_lexicon 17
obo_lex_pos 17
obo_mkconf 18
obo_mkdocset 18
obo_new 19
obo_rebuild_corpus 19
read_cooccur 20
read_dataset 21
read_frequencies 21
read_lexicon 22
read_pos_counts 23
save_spm 23
str_normalize 24
term_data 25
term_idx 25
tf 26
tfidf 26
weight_cooc 28
%.% 29

Index **30**

`classify` *Wrapper for cut.*

Description

Wrapper for cut(x, k) that accepts vectors or a generator function for labels, and optionally returns a numeric vector instead of a factor.

Usage

```
classify(x, k, factor = TRUE, labels = NA)
```

Arguments

x	Passed to cut as 'x'
k	Passed to cut as 'breaks'
factor	Logical. Coerce to numeric if FALSE.
labels	A vector of length equal to `length( k ) + 1` or a function to generate a vector of length `length( k ) + 1` from `1:lenght( k )` used as factor labels. Ignored if factor is FALSE.

Value

A vector of length equal to `length( x )` with the interval in k for each value of x

---

class_mass *Generate classes with equal mass.*

Description

Generates classes from a vector of masses (frequencies) s.t. that each generated class has equivalent total mass (and most likely very different sizes for highly skewed distributions like e.g. a lexicon's tf).

Usage

```
class_mass(x, k = 100, factor = TRUE, labels = NA, log = FALSE,
  sort = TRUE, desc = TRUE)
```

Arguments

x	A vector of massese or frequencies.
k	The desired number of classes.
factor	If false, return a numerci vector instead of a factor.
labels	A vector of length k or a function to be applied over 1:k to be used as factor labels. Ignored if factor==FALSE.
log	Logical. Compute mass using log1p(x) instead of x.
sort	Logical. Sort x before calculating mass and class; ensures members in each class have individual masses in the same order of magnitude.

Value

A vector of length equal to k with the class for each x.

class_rank *Generate rank classes with equal population size.*

Description

Generates classes of approximately equal size over intervals of the rankings in x. i.e. splits the vector of ranks in x in classes s.t. each class contains a similar number of observations.

Usage

```
class_rank(x, k = 100, factor = TRUE, labels = NA, desc = TRUE,
  no.sort = FALSE)
```

Arguments

x	A sortable vector.
k	The desired number of classes.
factor	If false, return a numeric vector instead of a factor.
labels	A vector of length k or a function to be applied over 1:k to be used as factor labels. Ignored if factor==FALSE
desc	Logical. Sort in descending order. TRUE by default.
no.sort	Logical. Don't sort x, for cases in which x is already sorted (or results will be wrong).

Details

This is similar to the "Jenks" classification strategy.

Value

A vector of length equal to k with the class for each x.

coef_bhattacharyya *Bhattacharyya coefficient*

Description

Computes the Bhattacharyya overlap coefficient between the probability distributions represented by the given frequency or probability vectors. The Bhattacharyya coefficient is equal to $\sum_i^n \sqrt{p_i * q_i}$.

Usage

```
coef_bhattacharyya(p, q)
```

Arguments

p	A frequency or probability vector. Coerced to prob by p / sum(p)
q	A frequency or probability vector. Coerced to prob by q / sum(q)

Details

The given vectors will be coerced to probabilities by `x = x / sum( x )`.

Value

a scalar value equal to the Bhattacharyya overlap coefficient between p and q.

`dist_hellinger` *Hellinger distance*

Description

Computes the Hellinger distance between the probability distributions represented by the given frequency or probability vectors p and q. The Hellinger distance is equal to the square root of the 1-complement of the Bhattacharyya coefficient between p and q: $\sqrt{1 - \sum_i^n \sqrt{p_i * q_i}}$.

Usage

```
dist_hellinger(p, q)
```

Arguments

p	A frequency or probability vector. Coerced to prob by p / sum(p)
q	A frequency or probability vector. Coerced to prob by q / sum(q)

Details

This function only computes the square root of the 1-complement. The coefficient is computed by `coef_bhattacharyya(p, q)`.

Unlike the closely related Bhattacharyya divergence, this measure is a proper distance as it respects the triangle inequality.

Value

a scalar value equal to the Hellinger divergence between p and q.

div_bhattacharyya *Bhattacharyya divergence*

Description

Computes the Bhattacharyya divergence between the probability distributions represented by the given frequency or probabality vectors. The Bhattacharyya divergence is equal to the reciprocal of the natural logarithm of the Bhattacharyya coefficient between p and q: $-log(\sum_i^n \sqrt{p_i * q_i})$

Usage

```
div_bhattacharyya(p, q)
```

Arguments

p	A frequency or probability vector. Coerced to prob by p / sum(p)
q	A frequency or probability vector. Coerced to prob by q / sum(q)

Details

This function just computes the natural logarithm reciprocal. The coefficient is computed by `bha_coef( p, q )`.

This measure is not a proper distance because it does not respect the triangle inequality. If a proper distance is needed, use the Hellinger distance (`dist_hellinger( p, q )`), also based on the Bhattacharyya coefficient.

Value

a scalar value equal to the Bhattacharyya divergence between p and q.

div_jensen_shannon *Jensen-Shannon divergence*

Description

Computes the Jensen-Shannon smoothing reflection of the Kullback-Leibler divergence.

Usage

```
div_jensen_shannon(p, q, ...)
```

Arguments

p	A frequency or probability vector. Coerced to prob by p / sum(p)
q	A frequency or probability vector. Coerced to prob by q / sum(q)
...	further arguments passed to the underlying Kullback-Leibler implementation (@seealso div_kulback_leibler)
base	Logarithm base. Uses e by default, yielding measure in nats. Use 2 for measure in bits/shannons.

Details

The Jensen-Shannon divergence is equal to the arithmetic mean between the Kullback-Leibler divergences of p from m and q from m, where m is equal to the elemen-wise mean between p and q.

This function only computes the element-wise mean between p and q, and the mean between the KL of p and from m. The actual Kullback-Leibler divergences are computed by `div_kulback_leibler(p,q)`.

Unlike the underlying Kullback-Leibler, this measure is reciprocal.

Value

a scalar value equal to the Jensen-Shannon divergence between p and q.

`div_kulback_leibler` *Kullback-Leibler divergence*

Description

The Kullback-Leibler divergence of p from q is equal to $\sum_i P(i)\ log\ \frac{P(i)}{Q(i)}$. KL is valid only when q_i = 0 iff p_i = 0. If p or q contain any zeros, a correction will be applied, as per the 'zeros' parameter.

Usage

```
div_kulback_leibler(p, q, base = exp(1), zeros = c("drop", "smooth",
  "convex", "bail"), conexd = rexp)
```

Arguments

p	A frequency or probability vector. Coerced to prob by p / sum(p)
q	A frequency or probability vector. Coerced to prob by q / sum(q)
base	Logarithm base. Uses e by default, yielding measure in nats. Use 2 for measure in bits/shannons.
zeros	Strategy for dealing with zeros in p or q. See details.

Details

If either p or q contain any zeros, the value of the 'zeros' parameter determines what action to take, before computing x / sum(x): 'drop' will drop all with zero values in p or q: `x <- x[ p != 0 & q !=0 ]`. 'smooth' will smooth both vectors with a Dirichlet prior: `x <- ( x + 1 ) / ( sum( x ) + length( x ) )`. 'convex' will apply a convex combination of x with a uniform distribution: `x <- x + ( 1 / length( x ) )`. 'bail' will error out.

Value

A scalar value equal to the Kullback-Leibler divergence of p from q.

`div_setratio` *Ratio between sets.*

Description

Computes a ratio of the given aggregation function over the elements of the given vector for members of the given numerator set and the elements of the given vector for the given denominator set.

Usage

```
div_setratio(x, nset, dset, nfunc = length, dfunc = nfunc)
```

Arguments

`x`	A vector from where to select inputs to the given aggregation function.
`nset`	A vector representing a subset of x over which to apply the given function for the numerator value.
`dset`	A vector representing a subset of x over which to apply the given function for the denominator value.
`nfunc`	An aggregation function like sum or length for the numerator set. Defaults to length.
`dfunc`	An aggregation function like sum or length for the denominator set. Defaults to nfunc.

Details

This function can be used to implement many different measures of similarity/divergence not included in this package. It is also used for community/vertex weighting function in the wspaces' graph module.

Value

A scalar value equal to the value of the given function applied to the elements of nset over the value of the given funciton applied to the elements of dset.

graph_connected *Determine graph connectivity up to the given tolerance*

Description

This function will return true if the maximum component size of all components detached from the largest component is greater or equal to the given tolerance. E.g. a tolerance of one implies that a graph will still be considered connected if the minor components are at most orphan nodes, a tolerance of 2 implies the same if the minor components are at most only dyads, etc.

Usage

```
graph_connected(g, tol = 1, cmps = components(g))
```

Arguments

g	An igraph graph.
tol	An integer indicating the maximum size of detached components. Defaults to 1.
cmps	An igraph components object. Will be computed over g if none given.

Details

This function will call `igraph::components( g )` if no value is given for the cmps parameter. *This can be extremely slow*, but so far I have not been able to find or produce a faster component determination strategy. If you know of one, please contact me (e.g. if you know how to compute matrix kernels in less than O(n^3))

Value

`TRUE` if the given graph has no detached component larger than tol, `FALSE` otherwise.

graph_prune_connected *Prune a graph maintaining connectivity*

Description

Removes edges from a graph such that the resulting graph maintains connectivity up to the given tolerance.

Usage

```
graph_prune_connected(g, edges, tol = 1, dropV = TRUE, attr = "weight",
  desc = TRUE, verbose = FALSE)
```

Arguments

g	An igraph graph.
edges	A vector of edges, sorted according to their significance.
tol	An integer indicating the maximum allowed disconnected component size.
dropV	A logical indicating whether vertices in disconnected components below the given tolerance tol should be dropped from the graph. Defaults to TRUE.
attr	An edge attribute name, used to sort edges if no edge list provided. Ignored if edges is not NULL. Deafults to 'weight' if it is.
desc	Logical, indicating whether edges should be sorted in descending order according to attr. Ignored if edges is not NULL, defaults to TRUE if it is.

Details

The given tolerance value indicates the maximum size of detached components that will be tolerated when determining connectivity. i.e. a tolerance of one implies the graph will still be considered connected if the size of any detached component after edge removal is no greater than one.

Edges are removed according to the significance order provided in the given edges vector or, if none is provided, by the order induced by the given attribute.

Value

A subgraph of g with the least significant edges removed.

---

graph_weight_cv *Weight of ego's internal incident edges over all internal edges.*

Description

Computes the ratio between ego's non-community crossing incident edges and all of its community's internal edges. This measure is also known as ego's neighbourhood's "contribution" to its community.

Usage

```
graph_weight_cv(comm, g, v = V(g), attr = "weigth", aggr = sum)
```

Arguments

comm	A communities object. The result of a cluster-finding function on g.
g	The graph.
v	A vector of vertices in g. Defualts to all vertices in g.
attr	An edge attribute to sum over the two sets. Set to NA to use cardinalities instead.
aggr	A function to combine values for the two sets. Defaults to sum.

Details

This can be interpreted as the 'weight' of ego's inmediate neighbourhood on its community.

`graph_weight_vc` *Weight of ego's internal incident edges over all of its incident edges.*

Description

Computes the ratio between ego's non-community crossing incident edges and all of its incident edges. This measure is also known as ego's community's "contribution" to ego's neighbourhood.

Usage

```
graph_weight_vc(comm, g, v = V(g), attr = "weight", aggr = sum)
```

Arguments

`comm`	A communities object. The result of a cluster-finding function on g.
`g`	The graph.
`v`	A vector of vertices in g. Defualts to all vertices in g.
`attr`	An edge attribute to sum over the two sets. Set to NA to use cardinalities instead.
`aggr`	A function to combine values for the two sets. Defaults to sum.

Details

This can be interpreted as the 'weight' of ego's community on its inmediate neighbourhood.

`idf` *Compute IDF weights for the given DF vector.*

Description

Compute IDF weigghts for the given DF vector over the given D document sample size, using the given weighting mode.

Usage

```
idf(dfs, D, mode = 2L)
```

Arguments

`dfs`	A vector with raw document frequencies.
`D`	Numeric vector of length one indicating the document sample size.
`mode`	Numeric vector of length one indicating the mode used for IDF calculation. See details.

Details

- IDF modes
  - 0: Unary: 1 if df > 0, but terms with 0 DF are by definition excluded of the lexicon, so 1.
  - 1: Plain: log of total number of documents, D, over the term's df.
  - 2: Smooth: (default) log of total number of documents, D, over the term's df, plus 1.
  - 3: Max: log of maximum df, over the term's df.
  - 4: Probabilistic: log of the total number of documents minus the term's df over the term's df.

Value

A vector of the same length as dfs with IDF weights.

`is.real` *Real-valued vectors*

Description

wrapper for (is.numeric(x) || is.integer(x))

Usage

```
is.real(x)
```

Arguments

`x`	a vector.

Value

TRUE if x can be interpreted as a real number, FALSE otherwise.

`lexical_dataset` *Enforce internal conventions for lexical datasets.*

Description

Lexical datasests created by upstream in memory or dumped to files always: 1) are sorted in descending term frequency order, 2) have as many rows as there are terms in a corpus' lexicon, even if the requested dataset contains no observations for some terms (values will be NA) and 3) contain a column indicating the row's term, named as `conf()$termId`.

Usage

```
lexical_dataset(d)
```

Arguments

d	A dataset created by an obo object or read from disk.

Value

d with internal conventions enforced: a 'key' column preserving original sort order and a `conf()$termId` column indicating each's row term, with all data columns following.

---

lexical_join *Join lexical datasets.*

Description

wspaces lexical datasets maintain row identity in rownames. This function wraps `dplyr::left_join` to join the given datasets on rownames.

Usage

```
lexical_join(d1, d2)
```

Arguments

d1	A lexical dataset
d2	A lexical dataset

Value

the left join between d1 and d2, on rownames.

---

load_corpus *Load corpus data from the given directory.*

Description

Loads corpus data from the files found in the given directory. Corpus data includes a lexicon file, frequency counts for all corpus partitions (if any), POS counts for every word (lemma) in the lexicon and a cooccurrence matrix.

Usage

```
load_corpus(dir = getwd(), lexicon = "lxcn.dsv", frequencies = "freq.dsv",
  pos_counts = "posc.dsv", cooccur = "cooc.bin", quiet = FALSE,
  attach = FALSE, env = .GlobalEnv)
```

Arguments

dir	Directory to find corpus data files in.
lexicon	Lexicon file name, default 'lexicon.tsv'
frequencies	Frequencies file name, default 'frequencies.tsv'
cooccur	Cooccurrence matrix file name, default 'cooccur.bin'
quiet	Logical indicating whether progress and general stats messages should be silenced.
pos_table	POS counts file name, default 'pos_counts.tsv'

Details

See details below for the different file formats used. DSV files are used for lexicon, frequencies and POS counts data. Cooccurrence counts are saved as a sparse matrix stored in triplet format as a plain array of (int, int, float) triplets. Each data file's format is documented in its respective loading function read_*.

Value

A list of length four containing the loaded corpus datasets.

See Also

read_lexicon, read_frequencies, read_pos_counts and read_cooccur.

---

load_spm *Load sparse matrices from disk.*

Description

Reads a sparse matrix from a binary file containing a sequence of (int,int,double) triplets.

Usage

```
load_spm(file)
```

Arguments

file	A character vector indicating the path to the file to be read.

Value

A Matrix::sparseMatrix built from the triplets read from the given binary file.

obo_all_pos *Get all defined POS classes in the OBO corpus.*

Description

Get all defined POS classes in the OBO corpus.

Usage

```
obo_all_pos()
```

Value

Character vector containing all POS class names in OBO.

obo_count_cooc *Count cooccurrences over the given document set.*

Description

Produces a sparse matrix containing cooccurrence counts for all terms in the analysis field over all documents in the given document set.

Usage

```
obo_count_cooc(obo, lxcn = obo_lexicon(obo), ds = obo$docSample(),
  shrink = TRUE)
```

Arguments

obo	An OBO interface object.
ds	A DocSet, built from obo_mkdocset.

Value

A sparse matrix containing cooccurrence counts for all terms in the given field in the given sample of documents.

obo_get_fields *Get all fields contained in the index.*

Description

Obtains a character vector with all fields contained in the corpus index.

Usage

```
obo_get_fields(obo)
```

Arguments

obo	An OBO interface object.

Details

The returned values may be used to e.g. list all terms in any field, construct document sets over any one of the field's terms or combinations thereof, etc.

Value

A character vector with the names of all fields present in the index.

obo_get_terms *Get all terms in the given field.*

Description

Obtains a character vector with all the terms found in the given field.

Usage

```
obo_get_terms(obo, field)
```

Arguments

obo	An OBO interface object.
field	A field name, e.g. an element of obo_get_fields.

Details

Elements of the returned vector may be used to construct document sets over the same field using obo_mkdocset.

Value

A character vector containing all the terms found in the given field.

obo_lexicon *Get a copy of the lexicon.*

Description

Get a copy of the lexicon.

Usage

```
obo_lexicon(obo)
```

Arguments

obo	An OBO interface object.

Value

A lexicon data frame, with terms as row names and tf and df as columns.

obo_lex_pos *Get lexical POS classes in the OBO corpus.*

Description

Get lexical POS classes in the OBO corpus.

Usage

```
obo_lex_pos()
```

Value

Character vector containing lexical POS class names in OBO.

obo_mkconf *Create a new OBO conf object*

Description

The returned object may be used to modify corpus analysis parameters by using the native java method 'set'. See examples.

Usage

```
obo_mkconf(...)
```

Value

An OBOConf instance.

Examples

```
conf <- obo_mkconf()
conf@set( 'wPre', 10 ) # Set cooccurrence window leading offset to 10.
conf$set( 'wPos', 10 ) # Set cooccurrence window trailing offset to 10.
obo <- obo_new( conf ) # obtain an OBO index interface object.
```

obo_mkdocset *Build document sets.*

Description

Document sets define the sample of corpus segments that will be considered for all counting functions, particularly get_ cooccurrences.

Usage

```
obo_mkdocset(obo, field, terms)
```

Arguments

obo	An OBO interface object.
field	A field over which to construct a document set.
terms	A character vector containing terms to select documents for the DocSet.

Details

If the given vector's length is greater than one, the filter will be constructed as the union of all individual terms. If it is equal to one, it will be interpreted as a regular expression. If it's 0, the returned document set will be the empty set.

Value

A DocSet instance that can be used to select observations from a corpus.

`obo_new` *Create OBO interface object.*

Description

The returned OBO instance object will take its parameters from the given conf object. If no conf is given, one will be created with default parameters.

Usage

```
obo_new(..., conf = obo_mkconf(...))
```

Arguments

`conf`	An OBOConf instance, e.g. the value of `obo_mkconf`.

Details

All index access functions require an OBO interface object as first parameter.

Value

An OBO index interface object.

`obo_rebuild_corpus` *Rebuild corpus datasets.*

Description

This function will rebuild all corpus datasets using the parameters currently found in the given obo object's configuration.

Usage

```
obo_rebuild_corpus(obo, reload = TRUE, ...)
```

Arguments

`obo`	An OBO interface object.
`...`	Further arguments passed to load_corpus. Ignored if reload is FALSE.
`realod`	Logical indicating whether the new corpus files should be reloaded.

Details

The produced data sets will be dumped into the current configuration's data directory, using the configured filenames.

Value

If reload, the value of load_corpus. NULL otherwise.

`read_cooccur` *Read cooccurrence matrix from disk.*

Description

Reads cooccurrence data from a file on disk, stored as a sparse tensor saved in tuple format; i.e. a plain array of (coord, value) structs, where coord is itself an array of integer coordinates for each dimension in the represented tensor and value is a floating point number.

Usage

```
read_cooccur(file, lxcn = NULL, shrink = is.null(lxcn))
```

Arguments

`file`	Location of the cooccurrence matrix file on disk.
`lxcn`	A corpus lexicon. Optional in most cases, but necessary for shrinking partial sets.
`shrink`	Logical. Reduce matrix by removing empty rows and columns. Only useful with partial sets.

Details

The default corpus backend implementation uses a term-term context definition and stores cooccurrences in a rank-2 tensor, using long integers for coordinates and double precision floats for values.

The specific relationship between values in the returned tensor and the actual distributional patterns for each term depends on the corpus backend cooccurrence counting strategy.

The default implementation uses a sliding window with harmonic weights equal to the inverse of the distance between focus and context terms.

This is similar to the strategy used by the Glove model for the computation of cooccurrence counts. This format is for now hardcoded, but this is subject to change in future versions, including both the addition of higher ranks for storing partial sum components for each term-context tensor as well as summary statistics that may be useful for additional transformations of the cooccurrence data.

For now, the stored data consists of scalar values for the weighted sum of all term-context observations, with no data retained about partial results.

Value

A Matrix::sparseMatrix object containing occurrence counts for terms in contexts.

`read_dataset` *Wrapper for* `data.table::fread` *for lexical datasets.*

Description

This function is used to load all lexical datasets in order to enforce all conventions assumed in the rest of the lexical dataset manipulations in this package. See `lexical_dataset` for details.

Usage

```
read_dataset(file, sep = "@", header = TRUE, nas = 0)
```

Arguments

`file`	Location of the file containing the lexical dataset to load.
`sep`	Column separator. Defaults to '@'.
`header`	Logical. Take variable names from the first row. Defaults to TRUE.
`nas`	Value to replace NA's in loaded dataset. Defaults to 0. Set to NA to keep NA's.

Value

a data.frame constructed from the loaded file, with columns and keys set in the appropriate way for further lexical analysis.

`read_frequencies` *Read term frequency data from DSV file.*

Description

Reads term frequency data for all corpus partitions from a DSV file. Frequency data is recorded on disk as a series of records containing each term's string form and an array of frequency counts for all corpus partitions. Terms themselves are *not* stored as data frame columns, but are instead used as *row names*.

Usage

```
read_frequencies(file, header = TRUE, sep = "@", lxcn = NULL)
```

Arguments

file	Location of the frequencies file on disk.
header	Logical indicating whether the given file contains a header as first row. Default TRUE.
sep	Character indicating the field separator value. Deafault '@'.
lxcn	(optional) A lexicon data frame used for consistency checks.

Value

A data frame containing the list of terms as its row names, and as many columns as there are corpus partitions containing each term's frequency counts for the respective corpus partition.

---

read_lexicon *Read lexicon data from DSV file.*

Description

Reads lexicon data from a DSV file. Lexicon data is recorded on disk as a series of records containings each term's string form, its term total frequency and its total document frequency. Additional columns may be used for additional corpus-wide term statistics. Terms themselves are *not* stored as data frame columns, but are instead used as row names.

Usage

```
read_lexicon(file, header = TRUE, sep = "@")
```

Arguments

file	Location of the lexicon file on disk.
header	Logical indicating whether the given file contains a header as first row. Default TRUE.
sep	Character indicating the field separator value. Deafault '@'.

Details

The character vector terms <- rownames(lexicon) should be considered the canonical list of terms in a given corpus, and all other lexical data sets are guaranteed to have no more elements than its length (though they may be shorter if any filters are in effect)

Value

A data frame containing the list of terms as its row names, a tf column for total term frequencies and a df column for total document frequency for each term. The returned data frame may contain additional columns for extra corpus-wide statistics depending on the corpus backend implementation.

read_pos_counts *Read POS count data from DSV file.*

Description

Reads POS count data from a DSV file. POS count data is recorded on disk as a series of records containings each term's string form, followed by one column for each major POS group in the tagset used by the corpus backend for its NLP pipeline. Terms themselves are *not* stored as data frame columns, but are instead used as *row names*.

Usage

```
read_pos_counts(file, header = TRUE, sep = "@", drop = TRUE)
```

Arguments

file	Location of the POS count data file on disk.
header	Logical indicating whether the given file contains a header as first row. Default TRUE.
sep	Character indicating the field separator value. Deafault '@'.
drop	Logical. Drop POS columns with 0 occurrences (e.g. PUNCT in most cases). Default TRUE.

Value

A data frame containing the list of terms as its row names, and as many columns as there are POS groups in the corresponding tagset, with each term's counts on each POS group.

save_spm *Save sparse matrices to disk.*

Description

Writes a sparse matrix object to a binary file as a sequence of (int,int,double) triplets.

Usage

```
save_spm(m, file)
```

Arguments

m	A sparseMatrix instance
file	A file name

str_normalize *Normalize strings.*

Description

This function offers a consistent procedure for string normalization combining several common operations into one call. Term transofrmations across a corpus should always use a consistent normalization function, e.g. currying this function with a fixed set of parameters.

Usage

```
str_normalize(x, trim = TRUE, lower = TRUE, white = TRUE, nl = FALSE,
  punct = FALSE)
```

Arguments

x	A string (i.e. a character vector of length 1).
trim	Logical. Remove trailing and leading whitespace.
lower	Logical. Reduce everything to lower case.
white	Logical. Eliminate duplicate whitespace.
nl	Loigcal. Replace newlines with spaces.
punct	Logicel. Remove punctuation.
word	Logical. Remove non-word characters.

Value

a reasonably normalized string.

Examples

```
\code{
    x <- "some Ugly    dirty\nstring!"
    cat( x )
    cat( normalizeString( x ) )
}
\code{
    std_str_normalize <- function( x ) {
        str_normalize( x, trim=TRUE, lower=TRUE, white=TRUE, nl=TRUE, punct=FALSE )
    }
    normals <- std_str_normalize( ugly_strings )
}
```

term_data — *Extract term data from the given lexical dataset.*

Description

Extracts term data from the given term character vector by matching the given terms against the rownames in the given dataset. All lexical datasets constructed by wspaces keep their term identifier as rownames.

Usage

```
term_data(d, terms)
```

Arguments

d	A lexical dataset, i.e. a data frame or matrix with terms as row names.
terms	A character vector with terms to get data for.

Value

A subset of d containing only entries that are in terms.

term_idx — *Get index vector into the given data for the given terms.*

Description

Generates an index vector for each entry in the given terms character vector into the elements or rows of the given named vector, matrix or lexical dataset.

Usage

```
term_idx(d, terms)
```

Arguments

d	Data into which t compute index vector. Can be a named vector, named matrix or lexical dataset.
terms	A character vector with the terms for which to compute indices.

Details

Use this method to consistently extract data from different sources for a given subset of terms.

Value

An index vector with the location of data entries in d for the terms in terms.

tf — *Compute TF weights for the given raw TF vector.*

Description

Compute TF weights for the given raw TF vector over the given L document length, using the given weighting mode.

Usage

```
tf(tfs, L, mode = 2L)
```

Arguments

tfs	A vector with raw term frequencies.
L	Numeric vector of length one, indicating the document's total term length.
mode	Numeric vector of length one indicating the mode used for TF calculation. See details.

Details

- TF modes
  - 0: Boolean: 1 if tf > 0; 0 otherwise.
  - 1: Raw: Raw term frequency.
  - 2: Normalized: (default) Term frequency divided by the total number of terms in document (the 'length').
  - 3: Log-normlized: Natural log of the term frequency over total document terms, + 1.
  - 4: 0.5 normalized: K*(1-K) * (tf / max(tf)), with K set to 0.5.

Value

A vector of the same size as tfs with TF weights.

tfidf — *TF-IDF weighting.*

Description

Weigth the given frequency matrix using the TF-IDF stategy.

Usage

```
tfidf(tf_, df_, tf_mode = 2L, idf_mode = 2L, ow = FALSE)
```

Arguments

tf_	A matrix with one row for each term, and as many columns as documents or corpus segments there are frequencies for.
df_	A vector of length equal the number of rows in tf_, containing document frequencies, to compute the IDF component.
tf_mode	A term frequency weigthing strategy. See details.
idf_mode	An inverse document frequency weigthing strategy. See details.
ow	A logical vector indicating whether the result should be destructively copied over the input matrix.

Details

TF-IDF weights attempt to moderate the effect of very common terms, by dividing the total frequency of a term within a context (i.e. a document, but in general any corpus segment), by the number of contexts in which the term appears.

The idea behind this approach is that terms that appear in every possible context do not provide any additional information to the contexts in which they appear. Hence, all TF-IDF strategies compute weights as some variation of $TF_{t,c}/IDF_t$, where $TF_{t,c}$ is a monotonic function of a term t's prevalence within a specific context c and IDF_t is an inversely monotonic function of the term t's prevalence in all contexts across the entire corpus.

Note that the TF term is valid for a term in a context, while the IDF term is valid for a term across the entire corpus.

Values of tf_mode and idf_mode indicate how the TF and IDF terms are computed, as indicated below.

- TF modes
  - 0: Boolean: 1 if tf > 0; 0 otherwise.
  - 1: Raw: Raw term frequency.
  - 2: Normalized: (default) Term frequency divided by the total number of terms in document (the 'length').
  - 3: Log-normlized: Natural log of the term frequency over total document terms, + 1.
  - 4: 0.5 normalized: K*(1-K) * (tf / max(tf)), with K set to 0.5.
- IDF modes
  - 0: Unary: 1 if df > 0, but terms with 0 DF are by definition excluded of the lexicon, so 1.
  - 1: Plain: log of total number of documents, D, over the term's df.
  - 2: Smooth: (default) log of total number of documents, D, over the term's df, plus 1.
  - 3: Max: log of maximum df, over the term's df.
  - 4: Probabilistic: log of the total number of documents minus the term's df over the term's df.

Value

An isomorphic matrix to tf_, with entries weighted by the given strategy. If ow == TRUE, tf_ is replaced with this value.

`weight_cooc` *Weight the given cooccurrence matrix.*

Description

Computes one of several weighting and extent functions on the non-zero entries of the given (sparse) cooccurrence matrix and returns the transformed (sparse) matrix.

Usage

```
weight_cooc(m, rowm = NULL, colm = NULL, positive = TRUE, ow = FALSE,
  mode = 2L)
```

Arguments

`m`	A sparse matrix with raw cooccurrence counts or some function thereof.
`rowm`	A vector of length == nrow(m) with focal (i.e. row) term probabilities.
`colm`	A vector of length == ncol(m) with context (i.e. column) term probabilities.
`positive`	Logical. Truncate negative values to zero if reasonable (i.e. use log(p + 1) internally).
`ow`	Logical. Operate destructively on m by overwriting it with the result. Defaults to FALSE.
`mode`	Weighting function to apply on m. See details.

Details

Available weighting functions:

- 0: Type weight: $P(c|W) > 0?1 : 0$
- 1: Token weight: $P(c|W) > 0$
- 2: PMI: $log(P(c, W)/P(c) * P(W))$
- 3: Weighted PMI: $P(c, W) * log(P(c, W)/P(c) * P(W))$
- 4: t-Test: $P(c, W) - P(c) * P(W)/\sqrt{P(c, W)/N}$
- 5: z-Test: $P(c, W) - P(c) * P(W)/\sqrt{(P(c) * P(W)/N)}$
- 6: Log-likelihood approximation: See below.

Mode 6 uses [citation needed] 'allr' log-likelihood approximation: $-2\frac{L(F(w,c),F(w),F(c))}{L(F(w,c),F(w),\frac{F(W,c)}{F(W)})}$ with $L(k, n, x) = x^k * (1 - x)^{n-k}$

Value

a sparse matrix with similar structure to m with the results of the weighting function.

`%.%` *Perl-style concatenation operator.*

Description

Wrapper for paste(..., sep=")

Usage

```
... %.% NA
```

Index

%.%, 29

class_mass, 3
class_rank, 4
classify, 2
coef_bhattacharyya, 4

data.table::fread, *21*
dist_hellinger, 5
div_bhattacharyya, 6
div_jensen_shannon, 6
div_kulback_leibler, 7
div_setratio, 8
dplyr::left_join, *13*

graph_connected, 9
graph_prune_connected, 9
graph_weight_cv, 10
graph_weight_vc, 11

idf, 11
is.real, 12

lexical_dataset, 12, *21*
lexical_join, 13
load_corpus, 13
load_spm, 14

obo_all_pos, 15
obo_count_cooc, 15
obo_get_fields, *16*, 16
obo_get_terms, 16
obo_lex_pos, 17
obo_lexicon, 17
obo_mkconf, 18, *19*
obo_mkdocset, *15, 16*, 18
obo_new, 19
obo_rebuild_corpus, 19

read_cooccur, *14*, 20
read_dataset, 21
read_frequencies, *14*, 21
read_lexicon, *14*, 22
read_pos_counts, *14*, 23

save_spm, 23
str_normalize, 24

term_data, 25
term_idx, 25
tf, 26
tfidf, 26

weight_cooc, 28