Package: wordspace (via r-universe)

November 2, 2026

Type Package

Title Distributional Semantic Models in R

Version 0.2-9

Date 2025-08-17

Depends R (>= 3.3.0), Matrix (>= 1.3.0)

Imports Rcpp (>= 0.11.0), sparsesvd, iotools, methods, stats, utils, graphics, grDevices, cluster, MASS

LinkingTo Rcpp

Description An interactive laboratory for research on distributional semantic models ('DSM', see <https://en.wikipedia.org/wiki/Distributional_semantics> for more information).

License GPL-3

URL http://wordspace.r-forge.r-project.org/

LazyData yes

LazyDataCompression xz

Suggests knitr, markdown, tm, testthat (>= 3.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation yes

Author Stephanie Evert [cre, aut] (ORCID: <https://orcid.org/0000-0002-4192-2437>)

Maintainer Stephanie Evert <stephanie.evert@fau.de>

Repository https://cran.r-universe.dev

Date/Publication 2025-08-18 08:50:02 UTC

RemoteUrl https://github.com/cran/wordspace

RemoteRef HEAD

RemoteSha eb983c7e58d09e4db900dc9affb12697f6b532d6

Contents

wordspace-package . . . 3
as.distmat . . . 4
as.dsm . . . 5
as.dsm.tm . . . 6
as.matrix.dsm . . . 7
check.dsm . . . 8
context.vectors . . . 9
convert.lemma . . . 11
dim.dsm . . . 13
dimnames.dsm . . . 14
dist.matrix . . . 15
dsm . . . 18
dsm.canonical.matrix . . . 21
dsm.projection . . . 23
dsm.score . . . 25
DSM_GoodsMatrix . . . 31
DSM_HieroglyphsMatrix . . . 32
DSM_SingularValues . . . 33
DSM_TermContextMatrix . . . 34
DSM_TermTermMatrix . . . 35
DSM_Vectors . . . 36
DSM_VerbNounTriples_BNC . . . 37
ESSLLI08_Nouns . . . 38
eval.clustering . . . 39
eval.multiple.choice . . . 41
eval.similarity.correlation . . . 43
head.dist.matrix . . . 45
head.dsm . . . 46
match.split . . . 47
merge.dsm . . . 48
nearest.neighbours . . . 49
normalize.rows . . . 51
pair.distances . . . 52
plot.dist.matrix . . . 54
plot.eval.similarity.correlation . . . 55
print.dsm . . . 57
rbind.dsm . . . 58
read.dsm.matrix . . . 59
read.dsm.triplet . . . 60
read.dsm.ucs . . . 64
RG65 . . . 65
rowNorms . . . 66
rsvd . . . 68
scaleMargins . . . 69
SemCorWSD . . . 70
signcount . . . 71

subset.dsm 72
t.dsm 74
WordSim353 75
wordspace.openmp 76
write.dsm.matrix 77

Index **79**

`wordspace-package` *Distributional Semantic Models in R (wordspace)*

Description

This package aims to provide a toy laboratory for research and experimentation in distributional semantics as well as a user-friendly environment for building and applying distributional semantic models (DSM) in R

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Please cite this package as

Evert, Stefan (2014). Distributional semantics in R with the wordspace package. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 110–114, Dublin, Ireland.

and link to the package homepage at http://wordspace.r-forge.r-project.org/.

Tutorial materials using the package for examples and exercises are available from http://wordspace.collocations.de/ under an open-source license. You will also find some pre-compiled distributional semantic models (DSM) there.

See Also

If you are new to the **wordspace** package, you should start by reading the package vignette and trying out the code examples shown there.

Good starting points into the package documentation are `dsm`, `read.dsm.triplet`, `dist.matrix` and `nearest.neighbours`.

`as.distmat` *Mark an arbitrary matrix as a pre-computed* `dist.matrix` *object (wordspace)*

Description

Mark an arbitrary dense or sparse matrix as a pre-computed `dist.matrix` object, so it can be used with `nearest.neighbours` and `pair.distances`. Default methods are provided for a regular dense `matrix`, any type of `sparseMatrix` from the **Matrix** package, as well as a `dsm` object (from which the raw or scored co-occurrence matrix is extracted).

Usage

```
as.distmat(x, ...)

## S3 method for class 'matrix'
as.distmat(x, similarity=FALSE, symmetric=FALSE, ...)
## S3 method for class 'sparseMatrix'
as.distmat(x, similarity=FALSE, symmetric=FALSE, force.dense=FALSE, ...)
## S3 method for class 'dsm'
as.distmat(x, similarity=FALSE, symmetric=FALSE, force.dense=FALSE, ...)
```

Arguments

`x`	a matrix-like object of a suitable class (for which a method implementation is available) or a DSM object of class `dsm`
`similarity`	whether the matrix contains similarity or distance values. Note that sparse distance matrices (`similarity=FALSE`) are not supported.
`symmetric`	whether the distance or similarity is symmetric (i.e. it has the same rows and columns in the same order and $d(x, y) = d(y, x)$). Methods trust the specified value and do not check whether this is actually true.
`force.dense`	whether to convert a sparse distance matrix into a dense `matrix` object. Keep in mind that the resulting matrix may be extremely large.
`...`	additional arguments passed on to the method implementations (see respective manpages for details)

Details

This method is called `as.distmat` because the regular name `as.dist.matrix` would collide with the `as.dist` method for `matrix` objects.

The method has two main purposes:

1. enable the use of pre-computed distance information from external sources in **wordspace**;
2. disguise a (scored) co-occurrence matrix as a similarity matrix so that `nearest.neighbours` and `pair.distances` can be used for lookup of first-order co-occurrence data.

Value

If x is a dense matrix or force.dense=TRUE, it is assigned to class dist.matrix so it can be used with nearest.neighbours and pair.distances as well as the plot and head methods.

If x is a sparse matrix, it is marked with an attribute dist.matrix recognized by nearest.neighbours and pair.distances; however, method implementations for dist.matrix objects will not apply. **Important note:** In this case, x must be a non-negative similarity matrix and empty cells are treated as zeroes.

In either case, attributes similarity and symmetric are set as specified.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

plot and head methods for distances matrices; nearest.neighbours and pair.distances

Examples

```
# interpret co-occurrence frequency as similarity measure
M <- as.distmat(DSM_HieroglyphsMatrix, similarity=TRUE)
nearest.neighbours(M, "cat")
nearest.neighbours(M, "hear", byrow=FALSE)
```

---

as.dsm *Create DSM Object From Various R Data Structures (wordspace)*

Description

Convert co-occurrence data from various in-memory formats to DSM object.

Usage

```
as.dsm(obj, ...)
```

Arguments

obj	an object of a suitable class (for which a method implementation is available)
...	additional arguments passed on to the method implementation (see respective manpages for details)

Value

An object of class dsm.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

Currently available implementations: as.dsm.TermDocumentMatrix, as.dsm.DocumentTermMatrix

---

as.dsm.tm *Create DSM Object From* tm *Package (wordspace)*

Description

Convert a **tm** term-document or document-term matrix into a wordspace DSM object.

Usage

```
## S3 method for class 'TermDocumentMatrix'
as.dsm(obj, ..., verbose=FALSE)
## S3 method for class 'DocumentTermMatrix'
as.dsm(obj, ..., verbose=FALSE)
```

Arguments

obj	an term-document or document-term matrix from the **tm** package, i.e. an object of a class TermDocumentMatrix or DocumentTermMatrix.
...	additional arguments are ignored
verbose	if TRUE, a few progress and information messages are shown

Value

An object of class dsm.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

as.dsm and the documentation of the **tm** package

Examples

```
## Not run:
library(tm) # tm package needs to be installed
data(crude) # news messages on crude oil from Reuters corpus

cat(as.character(crude[[1]]), "\n") # a text example

corpus <- tm_map(crude, stripWhitespace) # some pre-processing
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeWords, stopwords("english"))

cat(as.character(corpus[[1]]), "\n") # pre-processed text

dtm <- DocumentTermMatrix(corpus) # document-term matrix
inspect(dtm[1:5, 90:99])   # rows = documents

wordspace_dtm <- as.dsm(dtm, verbose=TRUE) # convert to DSM
print(wordspace_dtm$S[1:5, 90:99]) # same part of dtm as above

wordspace_tdm <- t(wordspace_dtm) # convert to term-document matrix
print(wordspace_tdm)

## End(Not run)
```

`as.matrix.dsm` *Extract Matrix from DSM Object (wordspace)*

Description

Extract the co-occurrence or score matrix from a DSM object.

Usage

```
## S3 method for class 'dsm'
as.matrix(x, what = c("auto", "M", "S"), ...)
```

Arguments

`x`	an object of class `dsm`
`what`	whether to extract the raw co-occurrence matrix (`M`) or the score matrix (`S`). The default option `auto` prefers the score matrix if both are available.
`...`	any additional arguments are ignored

Details

This function ensures that the row and column names of the matrix are consistent with the row/column information tables of the DSM. For faster access to the matrix, simply use `x$M` or `x$S` directly.

Value

Either the raw co-occurrence matrix or the score matrix of the DSM `x`.

Note that unlike other `as.matrix` methods, a sparse matrix in canonical DSM format may be returned.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`dsm`, `dim.dsm`, `dimnames.dsm`, `dsm.is.canonical`

Examples

```
as.matrix(DSM_TermTerm)
as.matrix(DSM_TermContext)
```

---

check.dsm *Validate Internal Structure of DSM Object (wordspace)*

Description

Validate the internal structure of a DSM object and return a list with information about the object.

Usage

```
check.dsm(model, validate = FALSE, nonneg.check = FALSE)
```

Arguments

`model`	an object of class `dsm`
`validate`	carry out extended validation of internal consistency? (may be expensive)
`nonneg.check`	if `TRUE`, check the co-occurrence (M) and/or score (S) matrix for non-negativity (may be expensive)

Value

Aborts with error message if any inconsistency is detected. Otherwise a list with the following items is returned:

`nrow`	number of rows (target terms) of the DSM
`ncol`	number of columns (features) of the DSM
`N`	sample size of the underlying data set (may be `NA`)
`M$ok`	whether co-occurrence frequency matrix M is available
`M$sparse`	whether M is sparse or dense (only present if `M$ok`)
`M$canonical`	whether M is in canonical DSM format (only present if `M$ok`)
`M$nonneg`	whether M is non-negative (only present if `M$ok`, and may be `NA` unless `nonneg.check=TRUE` was specified)
`S$ok`	whether score matrix S is available
`S$sparse`	whether S is sparse or dense (only present if `S$ok`)
`S$canonical`	whether S is in canonical DSM format (only present if `S$ok`)
`S$nonneg`	whether S is non-negative (only present if `S$ok`, and may be `NA` unless `nonneg.check=TRUE` was specified)
`locked`	`TRUE` if matrix combines data with inconsistent row or column marginals (in this case, association scores cannot be computed any more)

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`dsm`, `print.dsm`

Examples

```
check.dsm(DSM_TermTerm)
```

---

context.vectors *Compute Bag-of-Words Context Vectors (wordspace)*

Description

Compute bag-of-words context vectors as proposed by Schütze (1998) for automatic word sense disambiguation and induction. Each context vector is the centroid of the DSM vectors of all terms occurring in the context.

Usage

```
context.vectors(M, contexts, split = "\\s+",
                drop.missing = TRUE, row.names=NULL)
```

Arguments

M	numeric matrix of row vectors for the terms specified by rownames(M), or an object of class dsm
contexts	the contexts for which bag-of-words representations are to be computed. Must be a character vector, a list of character vectors, or a list of labelled numeric vectors (see Details below).
split	Perl regular expression determining how contexts given as a character vector are split into terms. The default behaviour is to split on whitespace.
drop.missing	if TRUE (default), contexts that do not contain any known terms are silently dropped; otherwise the corresponding context vectors will be all zeroes.
row.names	a character vector of the same length as contexts, specifying row names for the resulting matrix of centroid vectors

Details

The contexts argument can be specified in several different ways:

- A character vector: each element represents a context given as a string, which will be split on the Perl regular expression split and then looked up in M. Repetitions are allowed and will be weighted accordingly in the centroid.
- A list of character vectors: each item represents a pre-tokenized context given as a sequence of terms to be looked up in M. Repetitions are allowed and will be weighted accordingly in the centroid.
- A list of labelled numeric vectors: each item represents a bag-of-words representation of a context, where labels are terms to be looked up in M and the corresponding values their frequency counts or (possibly non-integer) weights.
- *(deprecated)* A logical vector corresponding to the rows of M, which will be used directly as an index into M.
- *(deprecated)* An unlabelled integer vector, which will be used as an index into the rows of M.

For each context, terms not found in the matrix M are silently computed. Then a context vector is computed as the centroid of the remaining term vectors. If the context contains multiple occurrences of the same term, its vector will be weighted accordingly. If the context is specified as a bag-of-words representations, the terms are weighted according to the corresponding numerical values.

Neither word order nor any other structural properties of the contexts are taken into account.

Value

A numeric matrix with the same number of columns as M and one row for each context (excluding contexts without known terms if drop.missing=TRUE). If the vector contexts has names or

`row.names` is specified, the matrix rows will be labelled accordingly. Otherwise the row labels correspond to the indices of the respective entries in `contexts`, so matrix rows can always be identified unambiguously if `drop.missing=TRUE`.

If `drop.missing=FALSE`, a context without any known terms (including an empty context) is represented by an all-zero vector.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Schütze, Hinrich (1998). Automatic word sense discrimination. *Computational Linguistics*, **24**(1), 97–123.

See Also

SemCorWSD

Examples

```
# different ways of specifying contexts
M <- DSM_TermTermMatrix
context.vectors(M, c("dog cat cat", "cause effect")) # contexts as strings
context.vectors(M, list(c("dog", "cat", "cat"), c("cause", "effect"))) # pre-tokenized
context.vectors(M, list(c(dog=1, cat=2), c(cause=1, effect=1))) # bag of words

# illustration of WSD algorithm: 6 sentences each for two senses of "vessel"
VesselWSD <- subset(SemCorWSD, target == "vessel")
with(VesselWSD, cat(paste0(sense, ": ", sentence, "\n")))

# provide sense labels in case some contexts are dropped b/c of too many missing words
Centroids <- with(VesselWSD, context.vectors(DSM_Vectors, lemma, row.names=sense))
Centroids[, 1:5]

(res <- kmeans(Centroids, 2)$cluster) # flat clustering with k-means
table(rownames(Centroids), res)       # ... works perfectly

## Not run:
plot(hclust(dist.matrix(Centroids, as.dist=TRUE)))

## End(Not run)
```

`convert.lemma` *Transform CWB/Penn-Style Lemmas into Other Notation Formats (wordspace)*

Description

Transform POS-disambiguated lemma strings in CWB/Penn format (see Details) into several other notation formats.

Usage

```
convert.lemma(lemma, format=c("CWB", "BNC", "DM", "HW", "HWLC"), hw.tolower=FALSE)
```

Arguments

lemma	a character vector specifying one or more POS-disambiguated lemmas in CWB/Penn notation
format	the notation format to be generated (see Details)
hw.tolower	convert headword part to lowercase, regardless of output format

Details

Input strings must be POS-disambiguated lemmas in CWB/Penn notation, i.e. in the form

```
<headword>_<P>
```

where <headword> is a dictionary headword (which may be case-sensitive) and <P> is a one-letter code specifying the simple part of speech. Standard POS codes are

```
N ... nouns
Z ... proper nouns
V ... lexical and auxiliary verbs
J ... adjectives
R ... adverbs
I ... prepositions (including all uses of "to")
D ... determiners
. ... punctuation
```

For other parts of speech, the first character of the corresponding Penn tag may be used. Note that these codes are not standardised and are only useful for distinguishing between content words and function words.

The following output formats are supported:

CWB returns input strings without modifications, but validates that they are in CWB/Penn format

BNC BNC-style POS-disambiguated lemmas based on the simplified CLAWS tagset. The headword part of the lemma is unconditionally converted to lowercase. The standard POS codes listed above are translated into SUBST (nouns and proper nouns), VERB (verbs), ADJ (adjectives), ADV (adverbs), ART (determiners), PREP (prepositions), and STOP (punctuation). Other POS codes have no direct CLAWS equivalents and are mapped to UNC (unclassified), so the transformation should only be used for the categories listed above.

`DM` POS-disambiguated lemmas in the format used by Distributional Memory (Baroni & Lenci 2010), viz. `<headword>-<p>` with POS code in lowercase and headword in its original capitalisation. For example, `light_N` will be mapped to `light-n`.

`HW` just the undisambiguated headword

`HWLC` undisambiguated headword mapped to lowercase (same as `HW` with `hw.tolower=TRUE`)

Value

A character vector of the same length as `lemma`, containing the transformed lemmas. See Details above for the different output formats.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Baroni, Marco and Lenci, Alessandro (2010). Distributional Memory: A general framework for corpus-based semantics. *Computational Linguistics*, **36**(4), 673–712.

Examples

```
convert.lemma(RG65$word1, "CWB") # original format
convert.lemma(RG65$word1, "BNC") # BNC-style (simple CLAWS tags)
convert.lemma(RG65$word1, "DM")  # as in Distributional Memory
convert.lemma(RG65$word1, "HW")  # just the headword
```

---

dim.dsm *Dimensions of a DSM Object (wordspace)*

Description

Retrieve the dimensions of the co-occurrence and/or score matrix represented by a DSM object.

Usage

```
## S3 method for class 'dsm'
dim(x)
```

Arguments

`x`	an object of class `dsm`

Details

Note that an assignment form (`dim<-`) for modifying dimensions is not provided.

Value

An integer vector of length 2, specifying the number of rows and the number of columns of the DSM matrix.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

dsm, check.dsm, print.dsm, dimnames.dsm

Examples

```
dim(DSM_TermTerm)
```

---

dimnames.dsm *Dimnames of a DSM Object (wordspace)*

Description

Retrieve or set the dimnames of the co-occurrence and/or score matrix represented by a DSM object.

Usage

```
## S3 method for class 'dsm'
dimnames(x)
## S3 replacement method for class 'dsm'
dimnames(x) <- value
```

Arguments

x	an object of class dsm
value	a list of two character vectors with new row and column names for x. Both vectors must have appropriate length and may not be NULL.

Details

This method automatically checks that the row and column names of the co-occurrence and/or score matrix are consistent with the target terms and features listed in the row/column information tables.

Value

The dimnames() of a DSM object are always a list of length 2, consisting of two character vectors with row and column labels, respectively.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

dsm, check.dsm, dim.dsm

Examples

```
rownames(DSM_TermContext)
colnames(DSM_TermContext)

tmp <- DSM_TermContext
rownames(tmp)[3] <- "pet"
head(tmp, 4, 6)
```

---

dist.matrix *Distances/Similarities between Row or Column Vectors (wordspace)*

Description

Compute a symmetric matrix of distances (or similarities) between the rows or columns of a matrix; or compute cross-distances between the rows or columns of two different matrices. This implementation is faster than dist and can operate on sparse matrices (in canonical DSM format).

Usage

```
dist.matrix(M, M2 = NULL, method = "cosine", p = 2,
            normalized = FALSE, byrow = TRUE, convert = TRUE, as.dist = FALSE,
            terms = NULL, terms2 = terms, skip.missing = FALSE)
```

Arguments

M	a dense or sparse matrix representing a scored DSM, or an object of class dsm
M2	an optional dense or sparse matrix representing a second scored DSM, or an object of class dsm. If present, cross-distances between the rows (or columns) of M and those of M2 will be computed.
method	distance or similarity measure to be used (see "Distance Measures" below for details)
p	exponent of the minkowski L_p-metric, a numeric value in the range $0 \leq p < \infty$. The range $0 \leq p < 1$ represents a generalization of the standard Minkowski distance, which cannot be derived from a proper mathematical norm (see details below).

`normalized`	if `TRUE`, assume that the row (or column) vectors of `M` and `M2` have been appropriately normalised (depending on the selected distance measure) in order to speed up calculations. This option is often used with the `cosine` metric, for which vectors must be normalized wrt. the Euclidean norm. It is currently ignored for other distance measures.
`byrow`	whether to calculate distances between row vectors (default) or between column vectors (`byrow=FALSE`)
`convert`	if `TRUE`, similarity measures are automatically converted to distances in an appropriate way (see "Distance Measures" below for details). Note that this is the default setting and `convert=FALSE` has to be specified explicitly in order to obtain a similarity matrix.
`as.dist`	convert the full symmetric distance matrix to a compact object of class `dist`. This option cannot be used if cross-distances are calculated (with argument `M2`) or if a similarity measure has been selected (with option `convert=FALSE`).
`terms`	a character vector specifying rows of `M` for which distance matrix is to be computed (or columns if `byrow=FALSE`)
`terms2`	a character vector specifying rows of `M2` for which the cross-distance matrix is to be computed (or columns if `byrow=FALSE`). If only the argument `terms` is specified, the same set of rows (or columns) will be selected from both `M` and `M2`; you can explicitly specify `terms2=NULL` in order to compute cross-distances for all rows (or columns) of `M2`.
`skip.missing`	if `TRUE`, silently ignores terms not found in `M` (or in `M2`). By default (`skip.missing=FALSE`) an error is raised in this case.

Value

By default, a numeric matrix of class `dist.matrix`, specifying distances or similarities between term vectors. A similarity matrix is marked by an additional attribute `similarity` with value `TRUE`. If the distance or similarity matrix is symmetric (i.e. neither a cross-distance matrix nor based on an asymmetric distance measure), it is marked by an attribute `symmetric` with value `TRUE`.

If `as.dist=TRUE`, the matrix is compacted to an object of class `dist`.

Distance Measures

Given two DSM vectors x and y, the following distance metrics can be computed:

`euclidean` The Euclidean distance given by

$$d_2(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

`manhattan` The Manhattan (or "city block") distance given by

$$d_1(x, y) = \sum_i |x_i - y_i|$$

`maximum` The maximum distance given by

$$d_\infty(x, y) = \max_i |x_i - y_i|$$

`minkowski` The Minkowski distance is a family of metrics determined by a parameter $0 \leq p < \infty$, which encompasses the Euclidean, Manhattan and maximum distance as special cases. Also known as L_p-metric, it is defined by

$$d_p(x, y) = \left(\sum_i |x_i - y_i|^p \right)^{1/p}$$

for $p \geq 1$ and by

$$d_p(x, y) = \sum_i |x_i - y_i|^p$$

for $0 \leq p < 1$. In the latter case, it is not homogeneous and cannot be derived from a corresponding mathematical norm (cf. `rowNorms`).

Special cases include the Euclidean metric $d_2(x, y)$ for $p = 2$ and the Manhattan metric $d_1(x, y)$ for $p = 1$, but the dedicated methods above provide more efficient implementations. For $p \to \infty$, $d_p(x, y)$ converges to the maximum distance $d_\infty(x, y)$, which is also selected by setting `p=Inf`. For $p = 0$, $d_p(x, y)$ corresponds to the Hamming distance, i.e. the number of differences

$$d_0(x, y) = \#\{i | x_i \neq y_i\}$$

`canberra` The Canberra metric has been implemented for compatibility with the `dist` function, even though it is probably not very useful for DSM vectors. It is given by

$$\sum_i \frac{|x_i - y_i|}{|x_i| + |y_i|}$$

(see https://en.wikipedia.org/wiki/Canberra_distance). Terms with $x_i = y_i = 0$ are silently dropped from the summation.

Note that `dist` uses a different formula

$$\sum_i \frac{|x_i - y_i|}{|x_i + y_i|}$$

which is highly problematic unless x and y are guaranteed to be non-negative. Terms with $x_i = y_i = 0$ are imputed, i.e. set to the average value of all nonzero terms.

In addition, the following similarity measures can be computed and optionally converted to a distance metric (or dissimilarity):

`cosine` **(default)** The cosine similarity given by

$$\cos \phi = \frac{x^T y}{||x||_2 \cdot ||y||_2}$$

If `normalized=TRUE`, the denominator is omitted. If `convert=TRUE` (the default), the cosine similarity is converted to angular distance ϕ, given in degrees ranging from 0 to 180.

`jaccard` The generalized Jaccard coefficient given by

$$J(x, y) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)}$$

which is only defined for non-negative vectors x and y. If `convert=TRUE` (the default), the Jaccard metric $1 - J(x, y)$ is returned (see Kosub 2016 for details). Note that $J(0, 0) = 1$.

`overlap` An asymmetric measure of overlap given by

$$o(x, y) = \frac{\sum_i \min(x_i, y_i)}{\sum_i x_i}$$

for non-negative vectors x and y. If `convert=TRUE` (the default), the result is converted into a dissimilarity measure $1 - o(x, y)$, which is not a metric, of course. Note that $o(0, y) = 1$ and in particular $o(0, 0) = 1$.

Overlap computes the proportion of the "mass" of x that is shared with y; as a consequence, $o(x, y) = 1$ whenever $x \leq y$. If both vectors are normalized as probability distributions ($||x||_1 = ||y||_1 = 1$) then overlap is symmetric ($o(x, y) = o(y, x)$) and can be thought of as the shared probability mass of the two distributions. In this case, `normalized=TRUE` can be passed in order to simplify the computation to $o(x, y) = \sum_i \min(x_i, y_i)$.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`plot` and `head` methods for distance matrices; `nearest.neighbours` and `pair.distances` also accept a precomputed `dist.matrix` object instead of a DSM matrix `M`

`rowNorms` for length normalization of DSM vectors, which is highly recommended for most distance metrics (and implicit in `cosine`)

Examples

```
M <- DSM_TermTermMatrix
dist.matrix(M, as.dist=TRUE)                     # angular distance
dist.matrix(M, method="euclidean", as.dist=TRUE) # Euclidean distance
dist.matrix(M, method="manhattan", as.dist=TRUE) # Manhattan distance
dist.matrix(M, method="minkowski", p=1, as.dist=TRUE)  # L_1 distance
dist.matrix(M, method="minkowski", p=99, as.dist=TRUE) # almost L_Inf
dist.matrix(M, method="maximum", as.dist=TRUE)         # L_Inf (maximum)
dist.matrix(M, method="minkowski", p=.5, as.dist=TRUE) # L_0.5 distance
dist.matrix(M, method="minkowski", p=0, as.dist=TRUE)  # Hamming distance

round(dist.matrix(M, method="cosine", convert=FALSE), 3) # cosine similarity
```

`dsm` *Create DSM Object Representing a Distributional Semantic Model (wordspace)*

Description

This is the constructor function for `dsm` objects representing distributional semantic models, i.e. a co-occurrence matrix together with additional information on target terms (rows) and features (columns). A new DSM can be initialised with a dense or sparse co-occurrence matrix, or with a triplet representation of a sparse matrix.

Usage

```
dsm(M = NULL, target = NULL, feature = NULL, score = NULL,
    rowinfo = NULL, colinfo = NULL, N = NA,
    globals = list(), raw.freq = FALSE, sort = FALSE, verbose = FALSE)
```

Arguments

M	a dense or sparse co-occurrence matrix. A sparse matrix must be a subclass of sparseMatrix from the Matrix package. See "Details" below.
target	a character vector of target terms (see "Details" below)
feature	a character vector of feature terms (see "Details" below)
score	a numeric vector of co-occurrence frequencies or weighted/transformed scores (see "Details" below)
rowinfo	a data frame containing information about the rows of the co-occurrence matrix, corresponding to target terms. The data frame must include a column term with the target term labels. If unspecified, a minimal rowinfo table is compiled automatically (see "Details" below).
colinfo	a data frame containing information about the columns of the co-occurrence matrix, corresponding to feature terms. The data frame must include a column term with the feature term labels. If unspecified, a minimal colinfo table is compiled automatically (see "Details" below).
N	a single numeric value specifying the effective sample size of the co-occurrence matrix. This value may be determined automatically if raw.freq=TRUE.
globals	a list of global variables, which are included in the globals field of the DSM object. May contain an entry for the sample size N, which can be overridden by an explicitly specified value in the argument N.
raw.freq	if TRUE, entries of the co-occurrence matrix are interpreted as raw frequency counts. By default, it is assumed that some weighting/transformation has already been applied.
sort	if TRUE, sort rows and columns of a co-occurrence matrix specified in triplet form alphabetically. If the matrix is given directly (in argument M), rows and columns are never reordered.
verbose	if TRUE, a few progress and information messages are shown

Details

The co-occurrence matrix forming the core of the distributional semantic model (DSM) can be specified in two different ways:

1. As a dense or sparse matrix in argument M. A sparse matrix must be a subclass of dMatrix (from the Matrix package) and is automatically converted to the canonical storage mode used by the wordspace package. Row and column labels may be specified with arguments target and feature, which must be character vectors of suitable length; otherwise dimnames(M) are used.

2. As a triplet representation in arguments `target` (row label), `feature` (column label) and `score` (co-occurrence frequency or pre-computed score). The three arguments must be vectors of the same length; each set of corresponding elements specifies a non-zero cell of the co-occurrence matrix. If multiple entries for the same cell are given, their frequency or score values are added up.

The optional arguments `rowinfo` and `colinfo` are data frames with additional information about target and feature terms. If they are specified, they must contain a column `$term` matching the row or column labels of the co-occurrence matrix. Marginal frequencies and nonzero or document counts can be given in columns `$f` and `$nnzero`; any further columns are interpreted as meta-information on the target or feature terms. The rows of each data frame are automatically reordered to match the rows or columns of the co-occurrence matrix. Target or feature terms that do not appear in the co-occurrence matrix are silently discarded.

Counts of nonzero cells for each row and column are computed automatically, unless they are already present in the `rowinfo` and `colinfo` data frames. If the co-occurrence matrix contains raw frequency values, marginal frequencies for the target and feature terms are also computed automatically unless given in `rowinfo` and `colinfo`; the same holds for the effective sample size `N`.

If `raw.freq=TRUE`, all matrix entries must be non-negative; fractional frequency counts are allowed, however.

Value

An object of class `dsm`, a list with the following components:

`M` — A co-occurrence matrix of raw frequency counts in canonical format (see `dsm.canonical.matrix`).

`S` — A weighted and transformed co-occurrence matrix ("score" matrix) in canonical format (see `dsm.canonical.matrix`). Either `M` or `S` or both may be present. The object returned by `dsm()` will include `M` if `raw.freq=TRUE` and `S` otherwise.

`rows` — A data frame with information about the target terms, corresponding to the rows of the co-occurrence matrix. The data frame usually has at least three columns:

- `rows$term` the target term = row label
- `rows$f` marginal frequency of the target term; must be present if the DSM object contains a raw co-occurrence matrix `M`
- `rows$nnzero` number of nonzero entries in the corresponding row of the co-occurrence matrix

Further columns may provide additional information.

`cols` — A data frame with information about the feature terms, corresponding to the columns of the co-occurrence matrix, in the same format as `rows`.

`globals` — A list of global variables. The following variables have a special meaning:

- `globals$N` effective sample size of the underlying corpus; may be `NA` if raw co-occurrence counts are not available
- `globals$locked` if `TRUE`, the marginal frequencies are no longer valid due to a `merge`, `rbind` or `cbind` operation; in this case, association scores cannot be computed from the co-occurrence frequencies `M`

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

See dsm.canonical.matrix for a description of the canonical matrix formats. DSM objects are usually loaded directly from a disk file in UCS (read.dsm.ucs) or triplet (read.dsm.triplet) format.

Examples

```
MyDSM <- dsm(
  target =  c("boat", "boat", "cat",  "dog", "dog"),
  feature = c("buy",  "use",  "feed", "buy", "feed"),
  score =   c(1,      3,      2,      1,     1),
  raw.freq = TRUE
)

print(MyDSM)   # 3 x 3 matrix with 5 out of 9 nonzero cells
print(MyDSM$M) # the actual co-occurrence matrix

print(MyDSM$rows) # row information
print(MyDSM$cols) # column information
```

---

dsm.canonical.matrix *Canonical Formats for a DSM Co-occurrence Matrix (wordspace)*

Description

Test whether a co-occurrence matrix is represented in a DSM canonical format, or convert matrix to canonical format.

Usage

```
dsm.is.canonical(x, nonneg.check = FALSE)

dsm.canonical.matrix(x, triplet = FALSE, annotate = FALSE, nonneg.check = FALSE)
```

Arguments

x	a dense or sparse DSM co-occurrence matrix
nonneg.check	if TRUE, check whether all elements of the matrix are non-negative
triplet	if TRUE and if x is sparse, return a matrix in triplet format (class dgTMatrix) rather than in column-compressed format (class dgCMatrix). Note that this is *not* a canonical DSM format.

annotate — if TRUE, annotate x with attributes sparse and nonneg, indicating whether the matrix is in sparse representation and non-negative, respectively. Non-negativity is only checked if nonneg.check=TRUE; otherwise an existing attribute will be passed through without validation.

Details

Note that conversion into canonical format may result in unnecessary copying of x, especially if annotate=TRUE. For optimal performance, set annotate=FALSE whenever possible and do not call dsm.canonical.matrix() as a no-op.

Instead of

```
M <- dsm.canonical.matrix(M, annotate=TRUE, nonneg=TRUE)
```

use

```
M.flags <- dsm.is.canonical(M, nonneg=FALSE)
if (!M.flags$canonical) M <- dsm.canonical.matrix(M)
M.flags <- dsm.is.canonical(M, nonneg=TRUE)
```

If nonneg.check=FALSE and x has an attribute nonneg, its value is accepted without validation.

Checking non-negativity can be expensive and create substantial memory overhead. It is guaranteed to be efficient for a matrix in canonical format.

Value

dsm.is.canonical() returns a data frame containing a single row with the following items:

sparse — whether x is a sparse (TRUE) or dense (TRUE) matrix

canonical — whether x is in canonical format

nonneg — whether all cells of x are non-negative; may be NA if nonneg.check=FALSE

dsm.canonical.matrix() returns a matrix in canonical DSM format, i.e.

- of class matrix for a dense matrix (even if x is a denseMatrix object);
- of class dgCMatrix for a sparse matrix.

If triplet=TRUE and x is sparse, it returns a matrix of class dgTMatrix, which is *not* a canonical format.

If annotate=TRUE, the returned matrix has attributes sparse and nonneg (possibly NA).

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

`dsm.projection` *Reduce Dimensionality of DSM by Subspace Projection (wordspace)*

Description

Reduce dimensionality of DSM by linear projection of row vectors into a lower-dimensional subspace. Various projections methods with different properties are available.

Usage

```
dsm.projection(model, n,
               method = c("svd", "rsvd", "asvd", "ri", "ri+svd"),
               oversampling = NA, q = 2, rate = .01, power=1,
               with.basis = FALSE, verbose = FALSE)
```

Arguments

`model`	either an object of class `dsm`, or a dense or sparse numeric matrix
`method`	projection method to use for dimensionality reduction (see "DETAILS" below)
`n`	an integer specifying the number of target dimensions. Use `n=NA` to generate as many latent dimensions as possible (i.e. the minimum of the number of rows and columns of the DSM matrix).
`oversampling`	oversampling factor for stochastic dimensionality reduction algorithms (`rsvd`, `asvd`, `ri+svd`). If unspecified, the default value is 2 for `rsvd`, 10 for `asvd` and 10 for `ri+svd` (subject to change).
`q`	number of power iterations in the randomized SVD algorithm (Halko *et al.* 2009 recommend `q=1` or `q=2`)
`rate`	fill rate of random projection vectors. Each random dimension has on average `rate * ncol(model)` nonzero components in the original space
`power`	apply power scaling after SVD-based projection, i.e. multiply each latent dimension with a suitable power of the corresponding singular value. The default `power=1` corresponds to a regular orthogonal projection. For power > 1, the first SVD dimensions – i.e. those capturing the main patterns of M – are given more weight; for power < 1, they are given less weight. The setting `power=0` results in a full equalization of the dimensions and is also known as "whitening" in the PCA case.
`with.basis`	if `TRUE`, also returns orthogonal basis of the subspace as attribute of the reduced matrix (not available for random indexing methods)
`verbose`	if `TRUE`, some methods display progress messages during execution

Details

The following dimensionality reduction algorithms can be selected with the `method` argument:

svd singular value decomposition (SVD), using the efficient SVDLIBC algorithm (Berry 1992) from package **sparsesvd** if the input is a sparse matrix. If the DSM has been scored with `scale="center"`, this method is equivalent to principal component analysis (PCA).

rsvd randomized SVD (Halko *et al.* 2009, p. 9) based on a factorization of rank `oversampling * n` with `q` power iterations.

asvd approximate SVD, which determines latent dimensions from a random sample of matrix rows including `oversampling * n` data points. This heuristic algorithm is highly inaccurate and has been **deprecated**.

ri random indexing (RI), i.e. a projection onto random basis vectors that are approximately orthogonal. Basis vectors are generated by setting a proportion of `rate` elements randomly to $+1$ or -1. Note that this does not correspond to a proper orthogonal projection, so the resulting coordinates in the reduced space should be used with caution.

ri+svd RI to `oversampling * n` dimensions, followed by SVD of the pre-reduced matrix to the final `n` dimensions. This is not a proper orthogonal projection because the RI basis vectors in the first step are only approximately orthogonal.

Value

A numeric matrix with `n` columns (latent dimensions) and the same number of rows as the original DSM. Some SVD-based algorithms may discard poorly conditioned singular values, returning fewer than `n` columns.

If `with.basis=TRUE` and an orthogonal projection is used, the corresponding orthogonal basis B of the latent subspace is returned as an attribute `"basis"`. B is column-orthogonal, hence B^T projects into latent coordinates and BB^T is an orthogonal subspace projection in the original coordinate system.

For orthogonal projections, the attribute `"R2"` contains a numeric vector specifying the proportion of the squared Frobenius norm of the original matrix captured by each of the latent dimensions. If the original matrix has been centered (so that a SVD projection is equivalent to PCA), this corresponds to the proportion of variance "explained" by each dimension.

For SVD-based projections, the attribute `"sigma"` contains the singular values corresponding to latent dimensions. It can be used to adjust the power scaling exponent at a later time.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Berry, Michael~W. (1992). Large scale sparse singular value computations. *International Journal of Supercomputer Applications*, **6**, 13–49.

Halko, N., Martinsson, P. G., and Tropp, J. A. (2009). Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Technical Report 2009-05, ACM, California Institute of Technology.

See Also

rsvd for the implementation of randomized SVD, and sparsesvd for the SVDLIBC wrapper

Examples

```
# 240 English nouns in space with correlated dimensions "own", "buy" and "sell"
M <- DSM_GoodsMatrix[, 1:3]

# SVD projection into 2 latent dimensions
S <- dsm.projection(M, 2, with.basis=TRUE)

100 * attr(S, "R2") # dim 1 captures 86.4% of distances
round(attr(S, "basis"), 3) # dim 1 = commodity, dim 2 = owning vs. buying/selling

S[c("time", "goods", "house"), ] # some latent coordinates

## Not run:
idx <- DSM_GoodsMatrix[, 4] > .85 # only show nouns on "fringe"
plot(S[idx, ], pch=20, col="red", xlab="commodity", ylab="own vs. buy/sell")
text(S[idx, ], rownames(S)[idx], pos=3)

## End(Not run)
```

---

dsm.score *Weighting, Scaling and Normalisation of Co-occurrence Matrix (wordspace)*

Description

Compute feature scores for a term-document or term-term co-occurrence matrix, using one of several standard association measures. Scores can optionally be rescaled with an isotonic transformation function and centered or standardized. In addition, row vectors can be normalized to unit length wrt. a given norm.

This function has been optimized for efficiency and low memory overhead.

Usage

```
dsm.score(model, score = "frequency",
          sparse = TRUE, negative.ok = NA,
          transform = c("none", "log", "root", "sigmoid"),
          scale = c("none", "standardize", "center", "scale"),
          normalize = FALSE, method = "euclidean", p = 2, tol = 1e-6,
          matrix.only = FALSE, update.nnzero = FALSE,
          batchsize = 1e6, gc.iter = Inf)
```

Arguments

`model`	a DSM model, i.e. an object of class `dsm`
`score`	the association measure to be used for feature weighting; either a character string naming one of the built-in measures or a user-defined function (see "Details" below)
`sparse`	if `TRUE` (the default), compute sparse non-negative association scores (see "Details" below). Non-sparse association scores are only allowed if `negative.ok=TRUE`.
`negative.ok`	whether operations that introduce negative values into the score matrix (non-sparse association scores, standardization of columns, etc.) are allowed. The default (`negative.ok=NA`) is `TRUE` if the co-occurrence matrix M is dense, and `FALSE` if it is sparse. See "Details" below for the special value `negative.ok="nonzero"`.
`transform`	scale transformation to be applied to association scores (see "Details" below)
`scale`	if not `"none"`, standardize columns of the scored matrix by z-transformation (`"standardize"`), center them without rescaling (`"center"`), or scale to unit RMS without centering (`"scale"`)
`normalize`	if `TRUE` normalize row vectors of scored matrix to unit length, according to the norm indicated by `method` and `p`
`method`, `p`	norm to be used with `normalize=TRUE`. See `rowNorms` for admissible values and details on the corresponding norms
`tol`	if `normalize=TRUE`, row vectors with norm below `tol` are explicitly set to all zeroes instead of attempting to normalize them (see `normalize.rows` for more information)
`matrix.only`	whether to return updated DSM model (default) or only the matrix of scores (`matrix.only=TRUE`)
`update.nnzero`	if `TRUE` and a full DSM model is returned, update the counts of nonzero entries in rows and columns according to the matrix of scores (there may be fewer nonzero entries with sparse association scores, or more from dense association scores and/or column scaling)
`batchsize`	if `score` is a user-defined function, the co-occurrence matrix is divided into blocks of approx. `batchsize` elements each in order to reduce memory overhead
`gc.iter`	how often to run the garbage collector when computing user-defined association scores; `gc()` is called after every `gc.iter` batches in order to reclaim temporary data and keep memory overhead as low as possible. This option should only be specified if memory is very tight, since garbage collector runs can be expensive (e.g. when there are many distinct strings in the workspace). With the default value `gc.iter=Inf`, no calls to `gc()` will be made; `gc.iter=4` seems to give a good trade-off between memory overhead and degraded performance.

Details

Association measures: Association measures (AM) for feature scoring are defined in the notation of Evert (2008). The most important symbols are $O_{11} = O$ for the observed co-occurrence frequency, $E_{11} = E$ for the co-occurrence frequency expected under a null hypothesis of independence, R_1 for the marginal frequency of the target term, C_1 for the marginal frequency of the

feature term or context, and N for the sample size of the underlying corpus. Evert (2008) explains in detail how these values are computed for different types of co-occurrence; practical examples can be found in the distributional semantics tutorial at http://wordspace.collocations.de/.

Several commonly used AMs are implemented in optimized C++ code for efficiency and minimal memory overhead. They are selected by name, which is passed as a character string in the `score` argument. See below for a list of built-in measures and their full equations.

Other AMs can be applied by passing a user-defined function in the `score` argument. See "User-defined association measures" at the end of this section for details.

Built-in association measures: The names of the following measures can be abbreviated to a unique prefix. Equations are given in the notation of Evert (2008).

`frequency` **(default)** Co-occurrence **frequency**:

$$O_{11}$$

Use this association measure to operate on raw, unweighted co-occurrence frequency data.

`MI` **(Pointwise) Mutual Information**, a log-transformed version of the ratio between observed and expected co-occurrence frequency:

$$\log_2 \frac{O_{11}}{E_{11}}$$

Pointwise MI has a very strong bias towards pairs with low expected co-occurrence frequency (because of E_{11} in the denominator). It should only be applied if low-frequency targets and features have been removed from the DSM.

The sparse version of MI (with negative scores cut off at 0) is sometimes referred to as "positive pointwise Mutual Information" (**PPMI**) in the literature.

`log-likelihood` The G^2 statistic of a likelihood ratio test for independence of rows and columns in a contingency table, which is very popular in computational linguistics under the name **log-likelihood**:

$$\pm 2 \left(\sum_{ij} O_{ij} \cdot \log \frac{O_{ij}}{E_{ij}} \right)$$

This implementation computes *signed* association scores, which are negative iff $O_{11} < E_{11}$. Log-likelihood has a strong bias towards high co-occurrence frequency and often produces a highly skewed distribution of scores. It may therefore be advisable to combine it with an additional `log` transformation.

`simple-ll` Simple **log-likelihood** (Evert 2008, p. 1225):

$$\pm 2 \left(O_{11} \cdot \log \frac{O_{11}}{E_{11}} - (O_{11} - E_{11}) \right)$$

This measure provides a good approximation to the full log-likelihood measure (Evert 2008, p. 1235), but can be computed much more efficiently. It is also very similar to the **local-MI** measure used by several popular DSMs.

Like `log-likelihood`, this measure computes *signed* association scores and has a strong bias towards high co-occurrence frequency.

`t-score` The **t-score** association measure, which is popular for collocation identification in computational lexicography:

$$\frac{O_{11} - E_{11}}{\sqrt{O_{11}}}$$

T-score is known to filter out low-frequency data effectively. If used as a non-sparse measure, a "discounted" version with $\sqrt{(O+1)}$ in the denominator is computed.

`chi-squared` The X^2 statistic of Pearson's **chi-squared** test for independence of rows and columns in a contingency table, with Yates's correction applied:

$$\pm \frac{N\big(|O_{11}O_{22} - O_{12}O_{21}| - N/2\big)^2}{R_1 R_2 C_1 C_2}$$

This implementation computes *signed* association scores, which are negative iff $O_{11} < E_{11}$. The formula above gives a more compact form of Yates's correction than the familiar sum over the four cells of the contingency table.

`z-score` The **z-score** association measure, based on a normal approximation to the binomial distribution of co-occurrence by chance:

$$\frac{O_{11} - E_{11}}{\sqrt{E_{11}}}$$

Z-score has a strong bias towards pairs with low expected co-occurrence frequency (because of E_{11} in the denominator). It should only be applied if low-frequency targets and features have been removed from the DSM.

`Dice` The **Dice coefficient** of association, which corresponds to the harmonic mean of the conditional probabilities $P(\text{feature}|\text{target})$ and $P(\text{target}|\text{feature})$:

$$\frac{2O_{11}}{R_1 + C_1}$$

Note that Dice is inherently sparse: it preserves zeroes and does not produce negative scores.

The following additional scoring functions can be selected:

`tf.idf` The **tf-idf** weighting scheme popular in Information Retrieval:

$$O_{11} \cdot \log \frac{1}{df}$$

where df is the relative document frequency of the corresponding feature term and should be provided as a variable `df` in the model's column information. Otherwise, it is approximated by the feature's nonzero count n_p (variable `nnzero`) divided by the number K of rows in the co-occurrence matrix:

$$df = \frac{n_p + 1}{K + 1}$$

The discounting avoids division-by-zero errors when $n_p = 0$.

`reweight` Apply scale transformation, column scaling and/or row normalization to previously computed feature scores (from `model$S`). This is the only `score` that can be used with a DSM that does not contain raw co-occurrence frequency data.

Sparse association scores: If `sparse=TRUE`, negative association scores are cut off at 0 in order to (i) ensure that the scored matrix is non-negative and (ii) preserve sparseness. The implementation assumes that association scores are always ≤ 0 for $O_{11} = 0$ in this case and only computes scores for nonzero entries in a sparse matrix. All built-in association measures satisfy this criterion.

Other researchers sometimes refer to such sparse scores as "positive" measures, most notably positive point-wise Mutual Information (PPMI). Since `sparse=TRUE` is the default setting, `score="MI"` actually computes the PPMI measure.

Non-sparse association scores can only be computed if `negative.ok=TRUE` and will force a dense matrix representation. For this reason, the default is `FALSE` for a sparse co-occurrence matrix and `TRUE` for a dense one. A special setting `negative.ok="nonzero"` is provided for those who wish to abuse `dsm.score` for collocation analysis. In combination with `sparse=FALSE`, it will allow negative score values, but compute them only for the nonzero entries of a sparse co-occurrence matrix. For a dense co-occurrence matrix, this setting is fully equivalent to `negative.ok=TRUE`.

Scale transformations: Association scores can be re-scaled with an isotonic transformation function that preserves sign and ranking of the scores. This is often done in order to de-skew the distribution of scores or as an approximate binarization (presence vs. absence of features). The following built-in transformations are available:

`none` **(default)** A **linear** transformation leaves association scores unchanged.

$$f(x) = x$$

`log` The **logarithmic** transformation has a strong de-skewing effect. In order to preserve sparseness and sign of association scores, a signed and discounted version has been implemented.

$$f(x) = \operatorname{sgn}(x) \cdot \log(|x| + 1)$$

`root` The **signed square root** transformation has a mild de-skewing effect.

$$f(x) = \operatorname{sgn}(x) \cdot \sqrt{|x|}$$

`sigmoid` The **sigmoid** transformation produces a smooth binarization where negative values saturate at -1, positive values saturate at $+1$ and zeroes remain unchanged.

$$f(x) = \tanh x$$

User-defined association measures: Instead of the name of a built-in AM, a function implementing a user-defined measure can be passed in the `score` argument. This function will be applied to the co-occurrence matrix in batches of approximately `batchsize` elements in order to limit the memory overhead incurred. A user-defined AM can be combined with any of the transformations above, and `sparse=TRUE` will cut off all negative scores.

The user function can use any of following arguments to access the contingency tables of observed and expected frequencies, following the notation of Evert (2008):

`O`, `E` observed and expected co-occurrence frequency

`R1`, `R2`, `C1`, `C2` the row and column marginals of the contingency table

`N` sample size

`f`, `f1`, `f2` the frequency signature of a target-feature pair, a different notation for $f = O$, $f_1 = R_1$ and $f_2 = C_1$

`O11`, `O12`, `O21`, `O22` the contingency table of observed frequencies

`E11`, `E12`, `E21`, `E22` the contingency table of expected frequencies

`rows` a data frame containing information about the target items (from the `rows` element of `model`)

`cols` a data frame containing information about the feature items (from the `cols` element of `model`)

`...` must be specified to ignore unused arguments

Except for `rows` and `cols`, all these arguments will be numeric vectors of the same lengths or scalar values (`N`), and the function must return a numeric vector of the same length.

For example, the built-in Mutual Information measure could also be implemented with the user function

```
my.MI <- function (O, E, ...) log2(O / E)
```

and tf.idf scoring could be implemented as follows, provided that the feature information table `model$cols` contains a column `df` with relative document frequencies:

```
my.tfidf <- function (O11, cols, ...) O11 * log(1 / cols$df)
dsm.score(model, score=my.tfidf)
```

Warning: User-defined AMs are much less efficient than the built-in measures and should only be used on large data sets if there is a good reason to do so. Increasing `batchsize` may speed up the computation to some degree at the expense of bigger memory overhead.

Value

Either an updated DSM model of class `dsm` (default) or the matrix of (scaled and normalised) association scores (`matrix.only=TRUE`).

Note that updating DSM models may require a substantial amount of temporary memory (because of the way memory management is implemented in R). This can be problematic when running a 32-bit build of R or when dealing with very large DSM models, so it may be better to return only the scored matrix in such cases.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

More information about assocation measures and the notation for contingency tables can be found at http://www.collocations.de/ and in

Evert, Stefan (2008). Corpora and collocations. In A. Lüdeling and M. Kytö (eds.), *Corpus Linguistics. An International Handbook*, chapter 58, pages 1212–1248. Mouton de Gruyter, Berlin, New York.

See Also

`dsm`

Examples

```
model <- DSM_TermTerm
model$M # raw co-occurrence matrix

model <- dsm.score(model, score="MI")
round(model$S, 3) # PPMI scores

model <- dsm.score(model, score="reweight", transform="sigmoid")
round(model$S, 3) # additional sigmoid transformation

## user-defined scoring functions can implement additional measures,
## e.g. the conditional probability Pr(feature | target) as a percentage
my.CP <- function (O11, R1, ...) 100 * O11 / R1  # "..." is mandatory
model <- dsm.score(model, score=my.CP)
round(model$S, 3)

## shifted PPMI (with k = 2) creates all-zero rows and columns
model <- dsm.score(model, score=function (O, E, ...) log2(O / E) - 2,
                   normalize=TRUE, update.nnzero=TRUE)
round(model$S, 3) # normalization preserves all-zero rows
## use subset to remove such rows and columns
m2 <- subset(model, nnzero > 0, nnzero > 0) # must have updated nnzero counts
round(m2$S, 3)

## Not run:
# visualization of the scale transformations implemented by dsm.score
x <- seq(-2, 4, .025)
plot(x, x, type="l", lwd=2, xaxs="i", yaxs="i", xlab="x", ylab="f(x)")
abline(h=0, lwd=0.5); abline(v=0, lwd=0.5)
lines(x, sign(x) * log(abs(x) + 1), lwd=2, col=2)
lines(x, sign(x) * sqrt(abs(x)), lwd=2, col=3)
lines(x, tanh(x), lwd=2, col=4)
legend("topleft", inset=.05, bg="white", lwd=3, col=1:4,
       legend=c("none", "log", "root", "sigmoid"))

## End(Not run)
```

DSM_GoodsMatrix *A Scored Co-occurrence Matrix of Nouns Denoting Goods (wordspace)*

Description

A pre-scored verb-object co-occurrence matrix for 240 target nouns denoting goods and the 3 feature verbs *own*, *buy* and *sell*. This matrix is useful for illustrating the application and purpose of dimensionality reduction techniques.

Usage

```
DSM_GoodsMatrix
```

Format

A numeric matrix with 240 rows corresponding to target nouns denoting goods and 4 columns, corresponding to

`own`, `buy`, `sell`: association scores for co-occurrences of the nouns with the verbs *own*, *buy* and *sell*

`fringe`: an indicator of how close each point is to the "fringe" of the data set (ranging from 0 to 1)

Details

Co-occurrence data are based on verb-object dependency relations in the British National Corpus, obtained from `DSM_VerbNounTriples_BNC`. Only nouns that co-occur with all three verbs are included in the data set.

The co-occurrence matrix is weighted with *non-sparse* log-likelihood (`simple-ll`) and an additional logarithmic transformation (`log`). Row vectors are *not* normalized.

The *fringeness score* in column `fringe` indicates how close a data point is to the fringe of the data set. Values are distance quantiles based on PCA-whitened Manhattan distance from the centroid. For example, `fringe >= .8` characterizes 20% of points that are closest to the fringe. Fringeness is mainly used to select points to be labelled in plots or to take stratified samples from the data set.

Examples

```
DSM_GoodsMatrix[c("time", "goods", "service"), ]
```

DSM_HieroglyphsMatrix *A Small Co-occurrence Matrix (wordspace)*

Description

A small co-occurrence matrix of verb-object combinations from the British National Corpus (BNC) Verbs correspond to columns of the matrix and their object nouns to rows. This matrix is shown as the "hieroglyphs" example in the DSM turorial.

Usage

```
DSM_HieroglyphsMatrix
```

Format

A numeric matrix with 7 rows and 6 columns.

Rows represent the target nouns *knife*, *cat*, *dog*, *boat*, *cup*, *pig* and *banana*. Columns represent the feature verbs *get*, *see*, *use*, *hear*, *eat* and *kill*.

Examples

```
print(DSM_HieroglyphsMatrix)

## cosine similarities between rows of the matrix
round(dist.matrix(DSM_HieroglyphsMatrix, convert=FALSE), 3)
```

DSM_SingularValues *Typical Singular Values of a Term-Context Matrix (wordspace)*

Description

Typical singular values of a term-document matrix based on encyclopedia articles.

Usage

```
DSM_SingularValues
```

Format

A numeric vector of length 2623.

Details

The data were obtained by singular value decomposition of a term-document matrix representing 100,000 Wikipedia articles, with 2623 target terms from a Basic English vocabulary. Articles were truncated to the first ca. 500 words. Occurrence frequencies of the target terms were log-scaled and rows of the matrix were L2-normalized before applying the SVD.

Examples

```
## Not run:
plot(DSM_SingularValues, type="h", xaxs="i", yaxs="i")

## End(Not run)
```

DSM_TermContextMatrix *Example of a Term-Context Co-occurrence Matrix (wordspace)*

Description

This matrix is a typical example of a term-context DSM co-occurrence matrix, derived from the English Wikipedia. It is available as a plain matrix in sparse representation, and as DSM object including marginal frequency data.

Usage

```
DSM_TermContextMatrix

DSM_TermContext
```

Format

`DSM_TermContextMatrix` is a sparse numeric matrix of class `dgCMatrix` with 7 rows and 7 columns.

Rows represent the target nouns *cat*, *dog*, *animal*, *time*, *reason*, *cause*, *effect*.

Columns specify the occurrence frequencies of these nouns in Wikipedia articles on *Felidae*, *Pet*, *Feral*, *Boat*, *Philosohpy*, *Kant* and *Back Pain*.

`DSM_TermContext` is an object of class `dsm` based on the same co-occurrence matrix, but with additional information on marginal frequencies of the target terms and feature contexts.

See Also

This matrix/DSM describes the same target nouns as the term-term matrix `DSM_TermTermMatrix` and corresponding DSM object `DSM_TermTerm`.

Examples

```
DSM_TermContextMatrix["time", ] # row vector for target noun "time"

all.equal(DSM_TermContextMatrix, head(DSM_TermContext, Inf))

# M M' = symmetric matrix of co-occurrence frequencies of nouns within articles
tcrossprod(DSM_TermContextMatrix)
```

DSM_TermTermMatrix *Example of a Term-Term Co-occurrence Matrix (wordspace)*

Description

This matrix is a typical example of a term-term DSM co-occurrence matrix, derived from the English Wikipedia. It is available as a plain matrix in dense representation, and as a DSM object including marginal frequency data.

Usage

```
DSM_TermTermMatrix

DSM_TermTerm
```

Format

`DSM_TermTermMatrix` is a numeric matrix with 7 rows and 7 columns.

Rows represent the target nouns *cat*, *dog*, *animal*, *time*, *reason*, *cause*, *effect*.

Columns specify co-occurrence frequencies of these nouns with the words *breed*, *tail*, *feed*, *kill*, *important*, *explain* and *likely* in articles of the English Wikipedia. Co-occurring words must appear within a distance of at most two word tokens of each other.

`DSM_TermTerm` is an object of class `dsm` based on the same co-occurrence matrix, but with additional information on marginal frequencies of the target and feature terms.

See Also

This matrix/DSM describes the same target terms as the term-context matrix `DSM_TermContextMatrix` and corresponding DSM object `DSM_TermContext`.

Examples

```
DSM_TermTermMatrix["time", ] # row vector for target noun "time"

all.equal(DSM_TermTermMatrix, head(DSM_TermTerm, Inf))

## Not run:
plot(hclust(dist.matrix(DSM_TermTermMatrix, as.dist=TRUE)))

## End(Not run)
```

DSM_Vectors *Pre-Compiled DSM Vectors for Selected Words (wordspace)*

Description

A matrix of 50-dimensional pre-compiled DSM vectors for selected English content words, covering most of the words needed for several basic evaluation tasks included in the package. Targets are given as disambiguated lemmas in the form <headword>_<pos>, e.g. walk_V and walk_N.

Usage

```
DSM_Vectors
```

Format

A numeric matrix with 1667 rows and 50 columns.

Row labels are disambiguated lemmas of the form <headword>_<pos>, where the part-of-speech code is one of N (noun), V (verb), J (adjective) or R (adverb).

Attribute "sigma" contains singular values that can be used for post-hoc power scaling of the latent dimensions (see dsm.projection).

Details

The vocabulary of this DSM covers several basic evaluation tasks, including RG65, WordSim353 and ESSLLI08_Nouns, as well as the target nouns *bank* and *vessel* from SemCorWSD. In addition, 40 nearest neighbours each of the words white_J, apple_N, kindness_N and walk_V are included.

Co-occurrence frequency data were extracted from a collection of Web corpora with a total size of ca. 9 billion words, using a L4/R4 surface window and 30,000 lexical words as feature terms. They were scored with sparse simple log-likelihood with an additional log transformation, normalized to Euclidean unit length, and projected into 1000 latent dimensions using randomized SVD (see rsvd. For size reasons, the vectors have been compressed into 50 latent dimensions and renormalized.

Examples

```
nearest.neighbours(DSM_Vectors, "walk_V", 25)

eval.similarity.correlation(RG65, DSM_Vectors) # fairly good

# post-hoc power scaling: whitening (correspond to power=0 in dsm.projection)
sigma <- attr(DSM_Vectors, "sigma")
M <- scaleMargins(DSM_Vectors, cols=1 / sigma)
eval.similarity.correlation(RG65, M) # very good
```

DSM_VerbNounTriples_BNC

Verb-Noun Co-occurrence Frequencies from British National Corpus (wordspace)

Description

A table of co-occurrence frequency counts for verb-subject and verb-object pairs in the British National Corpus (BNC). Subject and object are represented by the respective head noun. Both verb and noun entries are lemmatized. Separate frequency counts are provided for the written and the spoken part of the BNC.

Usage

```
DSM_VerbNounTriples_BNC
```

Format

A data frame with 250117 rows and the following columns:

`noun`: noun lemma

`rel`: syntactic relation (`subj` or `obj`)

`verb`: verb lemma

`f`: co-occurrence frequency of noun-rel-verb triple in subcorpus

`mode`: subcorpus (`written` for the writte part of the BNC, `spoken` for the spoken part of the BNC)

Details

In order to save disk space, triples that occur less than 5 times in the respective subcorpus have been omitted from the table. The data set should therefore not be used for practical applications.

Source

Syntactic dependencies were extracted from the British National Corpus (Aston & Burnard 1998) using the C&C robust syntactic parser (Curran *et al.* 2007). Lemmatization and POS tagging are also based on the C&C output.

References

Aston, Guy and Burnard, Lou (1998). *The BNC Handbook*. Edinburgh University Press, Edinburgh. See also the BNC homepage at http://www.natcorp.ox.ac.uk/.

Curran, James; Clark, Stephen; Bos, Johan (2007). Linguistically motivated large-scale NLP with C&C and Boxer. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Posters and Demonstrations Sessions*, pages 33–36, Prague, Czech Republic.

Examples

```
# compile some typical DSMs for spoken part of BNC
bncS <- subset(DSM_VerbNounTriples_BNC, mode == "spoken")
dim(bncS) # ca. 14k verb-rel-noun triples

# dependency-filtered DSM for nouns, using verbs as features
# (note that multiple entries for same relation are collapsed automatically)
bncS_depfilt <- dsm(
  target=bncS$noun, feature=bncS$verb, score=bncS$f,
  raw.freq=TRUE, verbose=TRUE)

# dependency-structured DSM
bncS_depstruc <- dsm(
  target=bncS$noun, feature=paste(bncS$rel, bncS$verb, sep=":"), score=bncS$f,
  raw.freq=TRUE, verbose=TRUE)
```

ESSLLI08_Nouns *Noun Clustering Task from ESSLLI 2008 (wordspace)*

Description

A set of 44 nouns denoting basic-level concepts from 6 semantic classes, used as a gold standard in the ESSLLI 2008 shared task on noun clustering.

Usage

```
ESSLLI08_Nouns
```

Format

A data frame with 44 rows and the following 5 columns:

`word` a character vector specifying the 44 nouns in CWB/Penn format (see `convert.lemma`)

`class` a factor vector specifying the semantic class of each noun (bird, fruitTree, green, groundAnimal, tool, vehicle)

`class2` a factor vector specifying a coarser 3-class categorization (animal, vegetable, artifact)

`class3` a factor vector specifying a coarser 2-class categorization (natural, artifact)

`freq.bnc` a numeric vector specifying the frequency of each noun in the British National Corpus

Source

http://wordspace.collocations.de/doku.php/data:esslli2008:concrete_nouns_categorization

Examples

```
print(ESSLLI08_Nouns)
```

`eval.clustering` *Evaluate DSM on Clustering Task (wordspace)*

Description

Performs evaluation on a word clustering task by comparing a flat clustering solution based on semantic distances with a gold classification.

Usage

```
eval.clustering(task, M, dist.fnc = pair.distances, ...,
                details = FALSE, format = NA, taskname = NA,
                scale.entropy = FALSE, n.clusters = NA,
                word.name = "word", class.name = "class")
```

Arguments

`task`	a data frame listing words and their classes, usually in columns named `word` and `class`
`M`	a scored DSM matrix, passed to `dist.fnc`
`dist.fnc`	a callback function used to compute distances between word pairs. It will be invoked with character vectors containing the components of the word pairs as first and second argument, the DSM matrix `M` as third argument, plus any additional arguments (`...`) passed to `eval.multiple.choice`. The return value must be a numeric vector of appropriate length. If one of the words in a pair is not represented in the DSM, the corresponding distance value should be set to `Inf`.
`...`	any further arguments are passed to `dist.fnc` and can be used e.g. to select a distance measure
`details`	if `TRUE`, a detailed report with information on each task item is returned (see "Value" below for details)
`format`	if the task definition specifies POS-disambiguated lemmas in CWB/Penn format, they can automatically be transformed into some other notation conventions; see `convert.lemma` for details
`taskname`	optional row label for the short report (`details=FALSE`)
`scale.entropy`	whether to scale cluster entropy values to the range $[0, 1]$
`n.clusters`	number of clusters. The (very sensible) default is to generate as many clusters as their are classes in the gold standard.
`word.name`	the name of the column of `task` containing words
`class.name`	the name of the column of `task` containing gold standard classes

Details

The test words are clustered using the "partitioning around medoids" (PAM) algorithm (Kaufman & Rousseeuw 1990, Ch. 2) based on their semantic distances. The PAM algorithm is used because it works with arbitrary distance measures (including neihbour rank), produces a stable solution (unlike most iterative algorithms) and has shown to be on par with state-of-the-art spherical k-means clustering (CLUTO) in evaluation studies.

Each cluster is automatically assigned a majority label, i.e. the gold standard class occurring most frequently in the cluster. This represents the best possible classification that can be derived from the clustering.

As evaluation metrics, clustering **purity** (accuracy of the majority classification) and **entropy** are computed. The latter is defined as a weighted average over the entropy of the class distribution within each cluster, expressed in bits. If `scale.entropy=TRUE`, the value is divided by the overall entropy of the class distribution in the gold standard, scaling it to the range $[0, 1]$.

NB: The semantic distance measure selected with the extra arguments (`...`) should be *symmetric*. In particular, it is not very sensible to specify `rank="fwd"` or `rank="bwd"`.

NB: Similarity measures are not supported by the current clustering algorithm. Make sure not to call `dist.matrix` (from `dist.fnc`) with `convert=FALSE`!

Value

The default short report (`details=FALSE`) is a data frame with a single row and the columns `purity` (clustering purity as a percentage), `entropy` (scaled or unscaled clustering entropy) and `missing` (number of words not found in the DSM).

The detailed report (`details=TRUE`) is a data frame with one row for each test word and the following columns:

`word`	the test word (character)
`cluster`	cluster to which the word has been assigned; all unknown words are collected in an additional cluster `"n/a"`
`label`	majority label of this cluster (factor with same levels as `gold`)
`gold`	gold standard class of the test word (factor)
`correct`	whether majority class assignment is correct (logical)
`missing`	whether word was not found in the DSM (logical)

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

Suitable gold standard data sets in this package: `ESSLLI08_Nouns`

Support functions: `pair.distances`, `convert.lemma`

Examples

```
eval.clustering(ESSLLI08_Nouns, DSM_Vectors, class.name="class2")
```

`eval.multiple.choice` *Evaluate DSM on Multiple Choice Task (wordspace)*

Description

Evaluates DSM on a multiple choice task by selecting the answer option closest to the target term in distributional space. A typical example is the TOEFL Synonym Task (Landauer & Dumais 1997).

Usage

```
eval.multiple.choice(task, M, dist.fnc = pair.distances, ...,
                     details = FALSE, format = NA, taskname = NA,
                     target.name = "target", correct.name = "correct",
                     distractor.name = "^distract")
```

Arguments

`task`	a data frame listing the target word, the correct answer, and one or more additional choices (distractors) for each test item
`M`	a scored DSM matrix, passed to `dist.fnc`
`dist.fnc`	a callback function used to compute distances between term pairs (or similarity scores, which must be marked with an attribute `similarity=TRUE`). See "Details" below for further information.
`...`	any further arguments are passed to `dist.fnc` and can be used e.g. to select a distance measure
`details`	if `TRUE`, a detailed report with information on each task item is returned (see "Value" below for details)
`format`	if the task definition specifies POS-disambiguated lemmas in CWB/Penn format, they can automatically be transformed into some other notation conventions; see `convert.lemma` for details
`taskname`	optional row label for the short report (`details=FALSE`)
`target.name`	the name of the column of `task` containing the target word
`correct.name`	the name of the column of `task` containing the correct choice
`distractor.name`	a regular expression matching columns of `task` containing the distractors. The regular expression is matched with `perl=TRUE`.

Details

The callback function `dist.fnc` will be invoked with character vectors containing the components of the term pairs as first and second argument, the DSM matrix `M` as third argument, plus any additional arguments (`...`) passed to `eval.multiple.choice`. The return value must be a numeric vector of appropriate length. If one of the terms in a pair is not represented in the DSM, the

corresponding distance value should be set to `Inf` (or `-Inf` in the case of similarity scores). In most cases, the default callback `pair.distances` is sufficient if used with suitable parameter settings.

For each task item, distances between the target word and the possible choices are computed. Then all choices are ranked according to their distances; in the case of a tie, the *higher* rank is assigned to both words. A task item counts as a TP (*true positive*, i.e. a successful answer by the DSM) if the correct choice is ranked in first place. Note that if it is tied with another choice, both will be assigned rank 2, so the item does not count as a TP.

If either the target word is missing from the DSM or none of the choices is found in the DSM, the result for this item is set to `NA`, which counts as a FP (*false positive*) in the accuracy computation.

With the default `dist.fnc` callback, additional arguments `method` and `p` can be used to select a distance measure (see `dist.matrix` for details). It is pointless to specify `rank="fwd"`, as the neighbour ranks produce exactly the same candidate ranking as the distance values.

Value

The default short report (`details=FALSE`) is a data frame with a single row and the columns `accuracy` (percentage correct), `TP` (number of correct answers), `FP` (number of wrong answers) and `missing` (number of test items for which the distance between target and correct choice was not found in the DSM).

The detailed report (`details=TRUE`) is a data frame with one row for each task item and the following columns:

`target`	the target word (character)
`correct`	whether model's choice is correct (logical or `NA`)
`best.choice`	best choice according to the DSM (character)
`best.dist`	distance of best choice from target (numeric)
`correct.choice`	correct answer (numeric)
`correct.rank`	rank of correct answer among choices (integer)
`correct.dist`	distance of correct answer from target (numeric)

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Landauer, Thomas K. and Dumais, Susan T. (1997). A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological Review*, **104**(2), 211–240.

See Also

Suitable gold standard data sets in this package: **TODO**

Support functions: `pair.distances`, `convert.lemma`

Examples

```
## TODO
```

`eval.similarity.correlation` *Evaluate DSM on Correlation with Similarity Ratings (wordspace)*

Description

Performs evaluation by comparing the distances (or similarities) computed by a DSM with (typically human) word similarity ratings. Well-know examples are the noun pair ratings collected by Rubenstein & Goodenough (1965; RG65) and Finkelstein et al. (2002; WordSim353).

The quality of the DSM predictions is measured by Spearman rank correlation rho.

Usage

```
eval.similarity.correlation(task, M, dist.fnc=pair.distances,
                            details=FALSE, format=NA, taskname=NA,
                      word1.name="word1", word2.name="word2", score.name="score",
                            ...)
```

Arguments

`task`	a data frame containing word pairs (usually in columns `word1` and `word2`) with similarity ratings (usually in column `score`); any other columns will be ignored
`M`	a scored DSM matrix, passed to `dist.fnc`
`dist.fnc`	a callback function used to compute distances or similarities between word pairs. It will be invoked with character vectors containing the components of the word pairs as first and second argument, the DSM matrix `M` as third argument, plus any additional arguments (`...`) passed to `eval.similarity.correlation`. The return value must be a numeric vector of appropriate length. If one of the words in a pair is not represented in the DSM, the corresponding distance value should be set to `Inf` (or `-Inf` in the case of similarities).
`details`	if `TRUE`, a detailed report with information on each task item is returned (see Value below for details)
`format`	if the task definition specifies POS-disambiguated lemmas in CWB/Penn format, they can automatically be transformed into some other notation conventions; see `convert.lemma` for details
`taskname`	optional row label for the short report (`details=FALSE`)
`...`	any further arguments are passed to `dist.fnc` and can be used e.g. to select a distance measure
`word1.name`	the name of the column of `task` containing the first word of each pair
`word2.name`	the name of the column of `task` containing the second word of each pair
`score.name`	the name of the column of `task` containing the corresponding similarity ratings

Details

DSM distances are computed for all word pairs and compared with similarity ratings from the gold standard. As an evaluation criterion, Spearman rank correlation between the DSM and gold standard scores is computed. The function also reports a confidence interval for Pearson correlation, which might require suitable transformation to ensure a near-linear relationship in order to be meaningful.

NB: Since the correlation between similarity ratings and DSM distances will usually be negative, the evaluation report omits minus signs on the correlation coefficients.

With the default `dist.fnc`, the distance values can optionally be transformed through an arbitrary function specified in the `transform` argument (see `pair.distances` for details). Examples include `transform=log` (esp. for neighbour rank as a distance measure) and `transform=function(x) 1/(1+x)` (in order to transform distances into similarities). Note that Spearman rank correlation is not affected by any monotonic transformation, so the main evaluation results will remain unchanged.

If one or both words of a pair are not found in the DSM, the distance is set to a fixed value 10% above the maximum of all other DSM distances, or 10% below the minimum in the case of similarity values. This is done in order to avoid numerical and visualization problems with `Inf` values; the particular value used does not affect the rank correlation coefficient.

With the default `dist.fnc` callback, additional arguments `method` and `p` can be used to select a distance measure (see `dist.matrix` for details); `rank=TRUE` can be specified in order to use neighbour rank as a measure of semantic distance.

Value

The default short report (`details=FALSE`) is a data frame with a single row and the following columns:

`rho`	(absolute value of) Spearman rank correlation coefficient rho
`p.value`	p-value indicating evidence for a significant correlation
`missing`	number of pairs not included in the DSM
`r`	(absolute value of) Pearson correlation coefficient r
`r.lower`	lower bound of confidence interval for Pearson correlation
`r.upper`	upper bound of confidence interval for Pearson correlation

The detailed report (`details=TRUE`) is a copy of the original task data with two additional columns:

`distance`	distance calculated by the DSM for each word pair, possibly transformed (numeric)
`missing`	whether word pair is missing from the DSM (logical)

In addition, the short report is appended to the data frame as an attribute `"eval.result"`, and the optional `taskname` value as attribute `"taskname"`. The data frame is marked as an object of class `eval.similarity.correlation`, for which suitable `print` and `plot` methods are defined.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Finkelstein, Lev, Gabrilovich, Evgeniy, Matias, Yossi, Rivlin, Ehud, Solan, Zach, Wolfman, Gadi, and Ruppin, Eytan (2002). Placing search in context: The concept revisited. *ACM Transactions on Information Systems*, **20**(1), 116–131.

Rubenstein, Herbert and Goodenough, John B. (1965). Contextual correlates of synonymy. *Communications of the ACM*, **8**(10), 627–633.

See Also

Suitable gold standard data sets in this package: `RG65`, `WordSim353`

Support functions: `pair.distances`, `convert.lemma`

Plotting and printing evaluation results: `plot.eval.similarity.correlation`, `print.eval.similarity.correlation`

Examples

```
eval.similarity.correlation(RG65, DSM_Vectors)

## Not run:
plot(eval.similarity.correlation(RG65, DSM_Vectors, details=TRUE))

## End(Not run)
```

`head.dist.matrix` *Return the Top Left Corner of a Distance Matrix (wordspace)*

Description

Returns the first `n` rows and first `k` columns of a distance matrix returned by the `dist.matrix` function.

Usage

```
## S3 method for class 'dist.matrix'
head(x, n = 6L, k = n, ...)
```

Arguments

`x`	an distance matrix of class `dist.matrix`
`n`	a single integer specifying the number of rows to extract
`k`	a single integer specifying the number of columns to extract (default: same as number of rows)
`...`	all other arguments are silently ignored

Details

Note that in contrast to other head methods, negative values of n (and k) are not supported. There is also currently no corresponding tail method.

Value

A numeric matrix with n rows and k columns.

Note: this matrix is no longer marked as an object of class dist.matrix and thus prints nicely without attributes.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

head for the generic method.

Examples

```
dm <- dist.matrix(DSM_Vectors[1:100, ])
print(head(dm, 8, 5), digits=3)
```

---

head.dsm *Return the Top Left Corner of a DSM Matrix (wordspace)*

Description

Returns the first n rows and first k columns of the co-occurrence matrix stored in a dsm object. If a scored matrix is available, it is automatically used; otherwise the raw frequencies are shown.

Usage

```
## S3 method for class 'dsm'
head(x, n = 6L, k = n, ...)
```

Arguments

x	an object of class dsm
n	a single integer specifying the number of rows to extract
k	a single integer specifying the number of columns to extract (default: same as number of rows)
...	all other arguments are silently ignored

Details

Note that in contrast to other head methods, negative values of n (and k) are not supported. There is also currently no corresponding tail method.

Value

A dense or sparse co-occurrence matrix with n rows and k columns.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

head for the generic method.

Examples

```
head(DSM_TermTerm, Inf, Inf) # show full co-occurrence matrix

head(DSM_TermTerm, 3, 4)
```

---

match.split *Find Parallel Matches for Values in Groups (wordspace)*

Description

Given a set of values and a grouped vector x, find parallel matches of each value in the different groups and return their positions in the original vector x. If there are multiple matches of the same value in a group, only the position of the first match is returned.

Usage

```
match.split(x, f, values=NULL, groups=NULL, nomatch=NA_integer_)
```

Arguments

x	vector to be divided into groups and matched against
f	a factor that defines the grouping (or a vector that can be converted to a factor)
values	values to be matched in x. Defaults to values that occur in all groups of x as determined by f and groups
groups	a character vector listing the set of groups to be formed. Defaults to the levels of f and should be a subset of these levels if given explicitly
nomatch	the value to be returned in cases where no match is found (coerced to an integer)

Value

An integer matrix with one row for each value (in `values`) and one column for each group (in `groups`), specifying the index in `x` of the first match of a value within the respective group. If not match is found for a given combination of value and group, `nomatch` is inserted (defaults to `NA`).

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

`merge.dsm` *Merge Rows or Columns from Different DSM Objects (wordspace)*

Description

Warning: this function is deprecated and will be removed in a future release of `wordspace`. It may be re-introduced later with different semantics.

Usage

```
  ## S3 method for class 'dsm'
merge(x, y, ..., rows=TRUE, all=FALSE, term.suffix=NULL)
```

Arguments

`x, y, ...`	two or more objects of class `dsm` to be merged
`rows`	whether to merge rows (TRUE, default) or columns (FALSE) of the DSM matrices
`all`	if FALSE (default), only features shared by all DSMs are included in the merged DSM (or target terms with `rows=FALSE`). If TRUE, all features are included with missing frequency / score values replaced by zero (analogously for target terms with `rows=FALSE`). **This option is not implemented yet.**
`term.suffix`	optional character vector specifying one suffix string for each DSM, which will be appended to row (`rows=TRUE`) or column (`rows=FALSE`) labels in order to make them unique

Value

if `term.suffix` is specified, row information of returned DSM object will be extended with variables `orig.term` specifying the original terms and `orig.part` specifying the original component model (identified by the corresponding entry from `term.suffix`)

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

`nearest.neighbours` *Find Nearest Neighbours in DSM Space (wordspace)*

Description

Find the nearest neighbours of a term vector in a DSM, given either as a scored cooccurrence matrix or a pre-computed distance matrix. The target term can be selected by name (in which case the cooccurrence or distance matrix must be labelled appropriately) or specified as a vector (if the DSM is given as a matrix).

Usage

```
nearest.neighbours(M, term, n = 10, M2 = NULL, byrow = TRUE,
                   drop = TRUE, skip.missing = FALSE, dist.matrix = FALSE,
                   ..., batchsize=50e6, verbose=FALSE)
```

Arguments

`M`	either a dense or sparse matrix representing a scored DSM (or an object of class `dsm`), or a pre-computed distance matrix returned by `dist.matrix` (as an object of class `dist.matrix`). Note that the compact representation produced by the `dist` function (class `dist`) is not accepted.
`term`	either a character vector specifying one or more target terms for which nearest neighbours will be found, or a matrix specifying the target vectors directly. A plain vector is interpreted as a single-row matrix.
`n`	an integer giving the number of nearest neighbours to be returned for each target term
`M2`	an optional dense or sparse matrix (or object of class `dsm`). If specified, nearest neighbours are found among the rows (default) or columns (`byrow=FALSE`) of `M2`, allowing for NN search in a cross-distance setting.
`byrow`	whether target terms are looked up in rows (default) or columns (`byrow=FALSE`) of `M`. NB: Target vectors in the `term` argument are always given as row vectors, even if `byrow=FALSE`.
`drop`	if `TRUE`, the return value is simplified to a vector (or distance matrix) if it contains nearest neighbours for exactly one target term (default). Set `drop=FALSE` to ensure that `nearest.neighbours` always returns a list.
`skip.missing`	if `TRUE`, silently ignores target terms not found in the DSM or distance matrix. By default (`skip.missing=FALSE`) an error is raised in this case.
`dist.matrix`	if `TRUE`, return a full distance matrix between the target term and its nearest neighbours (instead of a vector of neighbours). Note that a pre-computed distance matrix `M` must be symmetric in this case.
`...`	additional arguments are passed to `dist.matrix` if `M` is a scored DSM matrix. See the manpage of `dist.matrix` for details on available parameters and settings.

batchsize	if term is a long list of lookup terms, it will automatically be processed in batches. The number of terms per batch is chosen in such a way that approximately batchsize intermediate similarity values have to be computed and stored at a time (not used if M is a pre-computed distance matrix).
verbose	if TRUE, display some progress messages indicating how data are split into batches

Details

In most cases, the target term itself is automatically excluded from the list of neighbours. There are two exceptions:

1. The target term is given as a vector rather than by name.
2. Nearest neighbours are determined in a cross-distance setting. This is the case if (i) M2 is specified or (ii) M is a pre-computed distance matrix and not marked to be symmetric.

With dist.matrix=TRUE, the returned distance matrix always includes the target term.

M can also be a pre-computed distance or similarity matrix from an external source, which must be marked with as.distmat. If M is a sparse similarity matrix, only non-zero cells will be considered when looking for the nearest neighbours. Keep in mind that dist.matrix=TRUE is only valid if M is a symmetric matrix and marked as such.

Value

A list with one entry for each target term found in M, giving

- dist.matrix=FALSE (default): the nearest neighbours as a numeric vector of distances or similarities labelled with the corresponding terms and ordered by distance
- dist.matrix=TRUE: a full distance or similarity matrix for the target term and its nearest neighbours (as an object of class dist.matrix). An additional attribute selected contains a logical vector indicating the position of the target term in the matrix.

If drop=TRUE, a list containing only a single target term will be simplified to a plain vector or distance matrix.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

dist.matrix for more information on available distance metrics and similarity measures

Examples

```
nearest.neighbours(DSM_Vectors, c("apple_N", "walk_V"), n=10)

nearest.neighbours(DSM_Vectors, "apple_N", n=10, method="maximum")

as.dist(nearest.neighbours(DSM_Vectors, "apple_N", n=10, dist.matrix=TRUE))
```

`normalize.rows` *Normalize Rows or Columns of Matrix to Unit Length (wordspace)*

Description

Efficiently normalize the row or column vectors of a dense or sparse matrix to unit length.

Usage

```
normalize.rows(M, method = "euclidean", p = 2, ...,
               tol = 1e-6, inplace = FALSE)

normalize.cols(M, method = "euclidean", p = 2, ...,
               tol = 1e-6, inplace = FALSE)
```

Arguments

`M`	a dense or sparse numeric matrix
`method`	norm to be computed, see `rowNorms`
`p`	exponent of Minkowski p-norm in the range $0 < p \leq \infty$. Note that normalization is not possible for very small values of p.
`...`	any further arguments are passed to `rowNorms` (or `colNorms`)
`tol`	row/column vectors with norm below `tol` are assumed to be all zeroes and cannot be normalized (see "Details" below)
`inplace`	if `TRUE`, modify the matrix `M` in place. Don't ever set this argument to `TRUE`.

Details

These functions return a matrix with row (or column) vectors rescaled to a length of 1 according to the selected norm.

All-zero vectors (with $\|0\| = 0$) cannot be normalized. In order to avoid scaling up rounding errors, rows (or columns) with $\|x\| < tol$ are explicitly set to 0 (and thus not normalized). Since a suitable threshold for rounding errors depends on the scaling behaviour of the selected norm and the provenance of M, it is advisable to set `tol` explicitly to an appropriate value. Pass `tol = 0` to normalize all nonzero vectors.

The generalized Minkowski norm with $p < 1$ is not homogeneous but can still be normalized. This is numerically unstable for very small values of p, which will be rejected with an error message. The Hamming length ($p = 0$) cannot be normalized at all. See `rowNorms` for more information.

Value

A row-normalized (or column-normalized) matrix with the same dimensions as M.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

See rowNorms for details on available norms and their parameters.

`pair.distances` *Semantic Distances Between Word Pairs (wordspace)*

Description

Compute semantic distances (or similarities) between pairs of target terms based on a scored DSM matrix `M`, according to any of the distance measures supported by `dist.matrix`. If one of the terms in a pair is not represented in the DSM, the distance is set to `Inf` (or to `-Inf` in the case of a similarity measure).

Usage

```
pair.distances(w1, w2, M, ..., transform = NULL,
               rank = c("none", "fwd", "bwd", "avg"),
               avg.method = c("arithmetic", "geometric", "harmonic"),
               batchsize = 10e6, verbose = FALSE)
```

Arguments

`w1`	a character vector specifying the first term of each pair
`w2`	a character vector of the same length as `w1`, specifying the second term of each pair
`M`	a sparse or dense DSM matrix, suitable for passing to `dist.matrix`, or an object of class `dsm`. Alternatively, `M` can be a pre-computed distance or similarity matrix returned by `dist.matrix` or marked as such with `as.distmat`.
`...`	further arguments are passed to `dist.matrix` and determine the distance or similarity measure to be used (see `dist.matrix` for details)
`rank`	whether to return the distance between the two terms (`"none"`) or the neighbour rank (see "Details" below)
`transform`	an optional transformation function applied to the distance, similarity or rank values (e.g. `transform=log10` for logarithmic ranks). This option is provided as a convenience for evaluation code that calls `pair.distances` with user-specified arguments.
`avg.method`	with `rank="avg"`, whether to compute the arithmetic, geometric or harmonic mean of forward and backward rank
`batchsize`	maximum number of similarity values to compute per batch. This parameter has an essential influence on efficiency and memory use of the algorithm and has to be tuned carefully for optimal performance.
`verbose`	if `TRUE`, display some progress messages indicating how data are split into batches

Details

The rank argument controls whether semantic distance is measured directly by geometric distance (none), by forward neighbour rank (fwd), by backward neighbour rank (bwd), or by the average of forward and backward rank (avg). Forward neighbour rank is the rank of w2 among the nearest neighbours of w1. Backward neighbour rank is the rank of w1 among the nearest neighbours of w2. The average can be computed as an arithmetic, geometric or harmonic mean, depending on avg.method.

Note that a transformation function is applied *after* averaging. In order to compute the arithmetic mean of log ranks, set transform=log10, rank="avg" and avg.method="geometric".

Neighbour ranks assume that each target term is its own nearest neighbour and adjust ranks to account for this (i.e. w1 == w2 should return a rank of 0). If M is a pre-computed distance matrix, the adjustment is only applied if it is also marked as symmetric (because otherwise w1 might not appear in the list of neighbours at all). This might lead to unexpected results once asymmetric measures are implemented in dist.matrix.

For a sparse pre-computed similarity matrix M, only non-zero cells are considered as neighbours and all other ranks are set to Inf. This is consistent with the behaviour of nearest.neighbours.

pair.distances is used as a default callback in several evaluation functions, which rely on the attribute similarity to distinguish between distance measures and similarity scores. For this reason, transformation functions should always be **isotonic** (order-preserving) so as not to mislead the evaluation procedure.

Value

If rank="none" (the default), a numeric vector of the same length as w1 and w2 specifying the distances or similarities between the term pairs, according to the metric selected with the extra arguments (...).

Otherwise, an integer or numeric vector of the same length as w1 and w2 specifying forward, backward or average neighbour rank for the two terms.

In either case, a distance or rank of Inf (or a similarity of -Inf) is returned for any term pair not represented in the DSM. Attribute similarity is set to TRUE if the returned values are similarity scores rather than distances.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

dist.matrix, eval.similarity.correlation, eval.multiple.choice, nearest.neighbours

Examples

```
transform(RG65, angle=pair.distances(word1, word2, DSM_Vectors))
```

`plot.dist.matrix` *Plotting Distance Matrices (wordspace)*

Description

Visualization of a DSM distance matrix as a neighbourhood graph based on multidimensional scaling (MDS).

Usage

```
  ## S3 method for class 'dist.matrix'
plot(x, y, labels=rownames(x), show.labels=TRUE, label.pos=3,
     selected=attr(x, "selected"), show.selected=TRUE,
    col="black", cex=1, pch=20, pt.cex=1.2, selected.cex=1.2, selected.col="red",
    show.edges=TRUE, edges.lwd=6, edges.col="#AABBFF", edges.threshold=quantile(x, 2/3),
     method=c("isomds", "sammon"), aspect=1, expand=.05, ...)
```

Arguments

`x`	a symmetric distance matrix of class `dist.matrix`. NB: similarity values and asymmetric distance measures are not supported.
`y`	unused, must not be specified
`labels`	a character vector of labels for the DSM vectors (defaults to rownames of `x`)
`show.labels`	if `TRUE` (default), labels are displayed if available
`label.pos`	position of labels (default: above points). Possible values are `1` (below), `2` (left), `3` (above) and `4` (right).
`selected`	logical vector of selected points that will be highlighted (defaults to optional `selected` attribute of distance matrix)
`show.selected`	if `TRUE` (default), points marked by `selected` are highlighted in the plot
`col`	colour of points and labels
`cex`	numeric character expansion factor for points and labels
`pch`	plot symbol for points
`pt.cex`	character expansion factor for points relative to labels
`selected.cex`	additional character expansion factor for selected points and labels
`selected.col`	colour of selected points and labels (if `show.selected=TRUE`)
`show.edges`	if `TRUE` (default), edges are drawn between points. The line width of each edge is proportional to the distance between the corresponding points.
`edges.lwd`	maximal line width of edges (for $d = 0$)
`edges.col`	colour of edges, usually a light or translucent shade
`edges.threshold`	maximal distance up to which edges are drawn. The default is to display two thirds of all edges.

`method`	whether to perform non-metric (`isomds`) or metric (`sammon`) multidimensional scaling
`aspect`	aspect ratio of plot window (e.g. `aspect=16/10` for a window that is 8 inches wide and 5 inches high). Setting a correct aspect ratio ensures that the distances between points in the MDS map are correctly represented in the plot.
`expand`	fraction by which plotting region is extended on each side. Adjust this parameter so that points and labels are completely visible.
`...`	all other arguments are passed to the initial plot function, which sets up the display but does not draw any graphical elements

Details

For multidimensional scaling (MDS), the functions `isoMDS` and `sammon` from the **MASS** package are used.

Value

Invisibly returns a two-column matrix with MDS coordinates of all displayed points and `labels` as rownames (if available).

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`nearest.neighbours`, which produces distance matrices suitable for plotting if the option `dist.matrix=TRUE` is specified

Examples

```
## Not run:
plot(nearest.neighbours(DSM_Vectors, "walk_V", n=20, dist.matrix=TRUE))

## End(Not run)
```

---

plot.eval.similarity.correlation

Printing and Plotting Similarity Correlation Evaluation Results (wordspace)

Description

Suitable printing and visualization of evaluation results from `eval.similarity.correlation`. The `print` method displays an evaluation summary (stored in attribute `"eval.result"`) after the full data frame. The `plot` method displays a scatterplot of gold standard ratings against DSM distances with optional regression line (`lowess`), a summary of evaluation results at the top, and various other formatting options.

Usage

```
  ## S3 method for class 'eval.similarity.correlation'
print(x, ...)

  ## S3 method for class 'eval.similarity.correlation'
plot(x, y, line = TRUE,
     categories = NULL, cat.col = NA, cat.legend = "bottomleft",
     pch = 20, cex = 1, xlim = NULL, ylim = NULL,
     xlab = "human rating", ylab = "distributional model",
     main = attr(x, "taskname"), ...)
```

Arguments

x	detailed evaluation report from eval.similarity.correlation (with details=TRUE)
y	unused, must not be specified
line	if TRUE, a non-linear regression line is added to the plot in order to indicate the precise relationship between gold standard ratings and DSM distances (based on the lowess smoother)
categories	a factor with one entry for each word pair in the evaluation task. If specified, points in the scatterplot are colour-coded according to the categories of the corresponding word pairs. Note that categories is evaluated within the data frame x, so any column of x can directly be used as a variable.
cat.col	a vector of colours to be used for the different categories (defaults to the standard palette built into R)
cat.legend	corner of the plot in which to display the legend box for category colours
pch, cex	symbol used for plotting and its size
xlim, ylim	range of values shown on the x-axis and y-axis, respectively
xlab, ylab, main	axis labels and main title of the plot
...	all other arguments are passed to the scatterplot function (plot.default) and can be used to set additional graphics parameters

Details

Word pairs not found in the DSM are always shown as empty boxes in the scatterplot, regardless of the pch parameter.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

eval.similarity.correlation

Examples

```
## Not run:
plot(eval.similarity.correlation(WordSim353, DSM_Vectors, details=TRUE))

## End(Not run)
```

`print.dsm` *Print Information About DSM Object (wordspace)*

Description

Prints a short summary describing a `dsm` object, including the number of rows and columns.

Usage

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

Arguments

`x`	an object of class `dsm`
`...`	all other arguments are silently ignored

Details

The main purpose of this method is to keep users from accidentally trying to print out the internal data structures of a large DSM object.

For compatibility with the generic method (and the documentation of `print`), the DSM object is returned invisibly.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`print` for the generic method.

Examples

```
print(dsm(DSM_HieroglyphsMatrix))
```

rbind.dsm *Combine DSM Objects by Rows and Columns (wordspace)*

Description

Combine conformable DSM matrices by rows or columns. Additional information in the DSM objects (such as marginal frequencies) is checked for consistency and updated automatically.

Warning: these functions are experimental and may be removed or modified in a future release of `wordspace`

Usage

```
  ## S3 method for class 'dsm'
rbind(..., term.suffix=NULL, deparse.level = 1)

  ## S3 method for class 'dsm'
cbind(..., term.suffix=NULL, deparse.level = 1)
```

Arguments

`...`	one or more objects of class `dsm`, which must have the same feature dimensions (`rbind`) or target terms (`cbind`)
`term.suffix`	optional character vector specifying one suffix string for each DSM, which will be appended to row (`rbind`) or column (`cbind`) labels in order to make them unique
`deparse.level`	ignored

Value

if `term.suffix` is specified, row information of returned DSM object will be extended with variables `orig.term` specifying the original terms and `orig.part` specifying the original component model (identified by the corresponding entry from `term.suffix`)

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

`read.dsm.matrix` *Load DSM Matrix from File (wordspace)*

Description

This function loads a DSM matrix from a disk file in the specified format (see section sQuote(Formats) for details).

Usage

```
read.dsm.matrix(file, format = c("word2vec"),
                encoding = "UTF-8", batchsize = 1e6, verbose=FALSE)
```

Arguments

`file`	either a character string naming a file or a `connection` open for writing (in text mode)
`format`	input file format (see section sQuote(Formats)). The input file format cannot be guessed automatically.
`encoding`	character encoding of the input file (ignored if `file` is a connection)
`batchsize`	for certain input formats, the matrix is read in batches of `batchsize` cells each in order to limit memory overhead
`verbose`	if `TRUE`, show progress bar when reading in batches

Details

In order to read text formats from a compressed file, pass a `gzfile`, `bzfile` or `xzfile` connection with appropriate `encoding` in the argument `file`. Make sure not to open the connection before passing it to `read.dsm.matrix`.

Formats

Currently, the only supported file format is `word2vec`.

`word2vec` This widely used text format for word embeddings is only suitable for a dense matrix. Row labels must be unique and may not contain whitespace. Values are usually rounded to a few decimal digits in order to keep file size manageable.

The first line of the file lists the matrix dimensions (rows, columns) separated by a single blank. It is followed by one text line for each matrix row, starting with the row label. The label and are cells are separated by single blanks, so row labels cannot contain whitespace.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

write.dsm.matrix, read.dsm.triplet, read.dsm.ucs

Examples

```
fn <- system.file("extdata", "word2vec_hiero.txt", package="wordspace")
read.dsm.matrix(fn, format="word2vec")
```

---

read.dsm.triplet *Load DSM Data from Triplet Representation (wordspace)*

Description

This function loads a sparse distributional semantic model in triplet representation – (target label, feature label, score) – from a disk file or a pipe. Such a triplet file usually represents a pre-scored DSM, but it can also be used to read raw co-occurrence frequencies. In this case, marginals and sample size can either be derived from the co-occurrence matrix (for syntactic and term-context models) or provided in separate TAB-delimited tables (for surface and textual co-occurrence, or if frequency thresholds have been applied).

Usage

```
read.dsm.triplet(filename, freq = FALSE, value.first = FALSE, tokens = FALSE,
                 rowinfo = NULL, rowinfo.header = NULL,
                 colinfo = NULL, colinfo.header = NULL,
                 N = NA, span.size = 1,
                 sep = "\t", quote = "", nmax = -1, sort = FALSE,
                 encoding = getOption("encoding"), verbose = FALSE)
```

Arguments

filename	the name of a file containing the triplet data (see 'File Format' below for details), which may be compressed ('.gz', '.bz2', '.xz'). If filename ends in |, it is opened as a Unix pipe for reading.
freq	whether values are raw co-occurrence frequencies (TRUE) or pre-computed scores (FALSE)
value.first	if TRUE, triplets are given as (score, row label, column label) instead of the default (row label, column label, score)
tokens	if TRUE, the input file contains pair *tokens*, i.e. row and column labels without score/frequency values. Co-occurrence frequencies will automatically be calculated, but this input format should only be used for small samples up to a few millon tokens.
rowinfo	the name of an optional TAB-delimited table file with additional information about the target terms (see 'File Format' below for details), which may be compressed ('.gz', '.bz2', '.xz').

`rowinfo.header`	if the `rowinfo` file does not start with a header row, specify its column names as a character vector here
`colinfo`	the name of an optional TAB-delimited table file with additional information about the feature terms or contexts (see 'File Format' below for details), which may be compressed ('`.gz`', '`.bz2`', '`.xz`').
`colinfo.header`	if the `colinfo` file does not start with a header row, specify its column names as a character vector here
`N`	sample size to assume for the distributional model (see 'Details' below)
`span.size`	if marginal frequencies are provided externally for surface co-occurrence, they need to be adjusted for span size. If this hasn't been taken into account in data extraction, it can be approximated by specifying the total number of tokens in a span here (see 'Details' below).
`sep`, `quote`	specify field separator and the types of quotes used by the disk file (see the `scan` documentation for details). By default, a TAB-delimited file without quotes is assumed.
`nmax`	if the number of entries (= text lines) in the triplet file is known, it can be specified here in order to make loading faster and more memory-efficient. Caution: If `nmax` is smaller than the number of lines in the disk file, the extra lines will silently be discarded.
`sort`	if `TRUE`, the rows and columns of the co-occurrence matrix will be sorted alphabetically according to their labels (i.e. the target and feature terms); otherwise they are listed as encountered in the triplet representation
`encoding`	character encoding of the input files, which will automatically be converted to R's internal representation if possible. See 'Encoding' in `file` for details.
`verbose`	if `TRUE`, a few progress and information messages are shown

Details

The function `read.dsm.triplet` can be used to read triplet representations of three different types of DSM.

1. A pre-scored DSM matrix: If `freq=FALSE` and `tokens=FALSE`, the triplet file is assumed to contain pre-scored entries of the DSM matrix. Marginal frequencies are not required for such a model, but additional information about targets and features can be provided in separate `rowinfo=` and `colinfo=` files.

2. Raw co-occurrence frequencies (syntactic or term-context): If the triplet file contains syntactic co-occurrence frequencies or term-document frequency counts, specify `freq=TRUE`. For small data sets, frequencies can also be aggregated directly in R from co-occurrence tokens; specify `tokens=TRUE`.

Unless high frequency thresholds or other selective filters have been applied to the input data, the marginal frequencies of targets and features as well as the sample size can automatically be derived from the co-occurrence matrix. *Do not specify* `rowinfo=` *or* `colinfo=` *in this case!*

Evert (2008) explains the differences between syntactic, textual and surface co-occurrence.

3. Raw co-occurrence frequencies with explicit marginals: For surface and textual co-occurrence data, the correct marginal frequencies cannot be derived automatically and have to be provided in auxiliary table files specified with `rowinfo=` and `colinfo`. These files must contain a column `f` with the marginal frequency data. In addition, the total sample size (which cannot be derived from the marginals) has to be passed in the argument `N=`. Of course, it is still necessary to specify `freq=TRUE` (or `token=TRUE`) in order to indicate that the input data aren't pre-computed scores.

The computation of consistent marginal frequencies is particulary tricky for surface co-occurrence (Evert 2008, p. 1233f) and specialized software should be used for this purpose. As an approximation, simple corpus frequencies of target and feature terms can be corrected by a factor corresponding to the total size of the collocational span (e.g. `span.size=8` for a symmetric L4/R4 span, cf. Evert 2008, p. 1225). The `read.dsm.triplet` function applies this correction to the row marginals.

Explicit marginals should also be provided if syntactic co-occurrence data or text-context frequencies have been filtered, either individually with a frequency threshold or by selecting a subset of the targets and features. See the examples below for an illustration.

Value

An object of class `dsm` containing a sparse DSM.

For a model of type 1 (pre-scored) it will include the score matrix `$S` but no co-occurrence frequency data. Such a DSM object cannot be passed to `dsm.score`, except with `score="reweight"`. For models of type 2 and 3 it will include the matrix of raw co-occurrence frequencies `$M`, but no score matrix.

File Format

Triplet files: The triplet file must be a plain-text table with two or three TAB-delimited columns and no header. It may be compressed in `.gz`, `.bz2` or `.xz` format.

For `tokens=TRUE`, each line represents a single pair token with columns

1. target term
2. feature term / context

For `tokens=FALSE`, each line represents a pair type (i.e. a unique cell of the co-occurrence matrix) with columns:

1. target term
2. feature term / context
3. score (`freq=FALSE`) *or* co-occurrence frequency (`freq=TRUE`)

If `value.first=TRUE`, the score entry is expected in the first column:

1. score *or* co-occurrence frequency
2. target term
3. feature term / context

Note that the triplet file may contain multiple entries for the same cell, whose values will automatically be added up. This might not be very sensible for pre-computed scores.

Row and column information: Additional information about target terms (matrix rows) and feature terms / contexts (matrix columns) can be provided in additional TAB-delimited text tables, optionally compressed in `.gz`, `.bz2` or `.xz` format.

Such tables can have an arbitrary number of columns whose data types are inferred from the first few rows of the table. Tables should start with a header row specifying the column labels; otherwise they must be passed in the `rowinfo.header` and `colinfo.header` arguments.

Every table must contain a column `term` listing the target terms or feature terms / contexts. Their ordering need not be the same as in the main co-occurrence matrix, and redundant entries will silently be dropped.

If `freq=TRUE` or `tokens=TRUE`, the tables must also contain marginal frequencies in a column `f`. Nonzero counts for rows and columns of the matrix are automatically added unless a column `nnzero` is already present.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Evert, Stefan (2008). Corpora and collocations. In A. Lüdeling and M. Kytö (eds.), *Corpus Linguistics. An International Handbook*, chapter 58, pages 1212–1248. Mouton de Gruyter, Berlin, New York.

See Also

dsm, read.dsm.ucs

Examples

```
## this helper function displays the cooccurrence matrix together with marginals
with.marginals <- function (x) {
  y <- x$M
  rownames(y) <- with(x$rows, sprintf("%-8s | %6d", term, f))
  colnames(y) <- with(x$cols, sprintf("  %s | %d", term, f))
  y
}

## we will read this term-context DSM from a triplet file included in the package
with.marginals(DSM_TermContext)

## the triplet file with term-document frequencies
triplet.file <- system.file("extdata", "term_context_triplets.gz", package="wordspace")
cat(readLines(triplet.file), sep="\n") # file format

## marginals incorrect because matrix covers only subset of targets & features
TC1 <- read.dsm.triplet(triplet.file, freq=TRUE)
with.marginals(TC1) # marginal frequencies far too small

## TAB-delimited file with marginal frequencies and other information
marg.file <- system.file("extdata", "term_context_marginals.txt.gz", package="wordspace")
cat(readLines(marg.file), sep="\n") # notice the header row with "term" and "f"
```

```
## single table with marginals for rows and columns, but has to be specified twice
TC2 <- read.dsm.triplet(triplet.file, freq=TRUE,
                        rowinfo=marg.file, colinfo=marg.file, N=108771103)
with.marginals(TC2) # correct marginal frequencies

## marginals table without header: specify column lables separately
no.hdr <- system.file("extdata", "term_context_marginals_noheader.txt",
                      package="wordspace")
hdr.names <- c("term", "f", "df", "type")
TC3 <- read.dsm.triplet(triplet.file, freq=TRUE,
                        rowinfo=no.hdr, rowinfo.header=hdr.names,
                        colinfo=no.hdr, colinfo.header=hdr.names, N=108771103)
all.equal(TC2, TC3, check.attributes=FALSE) # same result
```

`read.dsm.ucs` *Load Raw DSM Data from Disk Files in UCS Export Format (wordspace)*

Description

This function loads raw DSM data – a cooccurrence frequency matrix and tables of marginal frequencies – in **UCS** export format. The data are read from a directory containing several text files with predefined names, which can optionally be compressed (see 'File Format' below for details).

Usage

```
read.dsm.ucs(filename, encoding = getOption("encoding"), verbose = FALSE)
```

Arguments

`filename`	the name of a directory containing files with the raw DSM data.
`encoding`	character encoding of the input files, which will automatically be converted to R's internal representation if possible. See 'Encoding' in `file` for details.
`verbose`	if `TRUE`, a few progress and information messages are shown

Value

An object of class `dsm` containing a dense or sparse DSM.

Note that the information tables for target terms (field `rows`) and feature terms (field `cols`) include the correct marginal frequencies from the UCS export files. Nonzero counts for rows are and columns are added automatically unless they are already present in the disk files. Additional fields from the information tables as well as all global variables are preserved with their original names.

File Format

The UCS export format is a directory containing the following files with the specified names:

- 'M' **or** 'M.mtx'
  cooccurrence matrix (dense, plain text) or sparse matrix (MatrixMarket format)
- 'rows.tbl'
  row information (labels term, marginal frequencies f)
- 'cols.tbl'
  column information (labels term, marginal frequencies f)
- 'globals.tbl'
  table with single row containing global variables; must include variable N specifying sample size

Each individual file may be compressed with an additional filename extension .gz, .bz2 or .xz; read.dsm.ucs automatically decompresses such files when loading them.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

The UCS toolkit is a software package for collecting and manipulating co-occurrence data available from http://www.collocations.de/software.html.

UCS relies on compressed text files as its main storage format. They can be exported as a DSM with ucs-tool export-dsm-matrix.

See Also

dsm, read.dsm.triplet

RG65 *Similarity Ratings for 65 Noun Pairs (wordspace)*

Description

A database of human similarity ratings for 65 English noun pairs, collected by Rubenstein & Goodenough (1965).

Usage

```
RG65
```

Format

A data frame with 65 rows and the following 3 columns:

`word1` first noun (character)

`word2` second noun (character)

`score` average similarity rating by human judges on scale from 0 to 4 (numeric)

The nouns are given as disambiguated lemmas in the form `<headword>_N`.

Details

The word pairs are sorted by increasing similarity score, as in the original paper.

Source

Rubenstein, Herbert and Goodenough, John B. (1965). Contextual correlates of synonymy. *Communications of the ACM*, **8**(10), 627–633.

Examples

```
head(RG65, 10) # least similar pairs
tail(RG65, 10) # most similar pairs
```

---

rowNorms *Compute Norms of Row and Column Vectors of a Matrix (wordspace)*

Description

Efficiently compute the norms of all row or column vectors of a dense or sparse matrix.

Usage

```
rowNorms(M, method = "euclidean", p = 2)

colNorms(M, method = "euclidean", p = 2)
```

Arguments

`M`	a dense or sparse numeric matrix
`method`	norm to be computed (see "Norms" below for details)
`p`	exponent of the `minkowski` p-norm, a numeric value in the range $1 \leq p \leq \infty$. Values $0 \leq p < 1$ are also permitted as an extension but do not correspond to a proper mathematical norm (see details below).

Value

A numeric vector containing one norm value for each row or column of `M`.

Norms

Given a row or column vector x, the following length measures can be computed:

`euclidean` The **Euclidean** norm given by

$$\|x\|_2 = \sqrt{\sum_i x_i^2}$$

`maximum` The **maximum** norm given by

$$\|x\|_\infty = \max_i |x_i|$$

`manhattan` The **Manhattan** norm given by

$$\|x\|_1 = \sum_i |x_i|$$

`minkowski` The **Minkowski** (or L_p) norm given by

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

for $p \geq 1$. The Euclidean ($p = 2$) and Manhattan ($p = 1$) norms are special cases, and the maximum norm corresponds to the limit for $p \to \infty$.

As an extension, values in the range $0 \leq p < 1$ are also allowed and compute the length measure

$$\|x\|_p = \sum_i |x_i|^p$$

For $0 < p < 1$ this formula defines a p-norm, which has the property $\|r \cdot x\| = |r|^p \cdot \|x\|$ for any scalar factor r instead of being homogeneous. For $p = 0$, it computes the Hamming length, i.e. the number of nonzero elements in the vector x.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`dist.matrix`, `normalize.rows`

Examples

```
rowNorms(DSM_TermContextMatrix, "manhattan")

# fast and memory-friendly nonzero counts with "Hamming length"
rowNorms(DSM_TermContextMatrix, "minkowski", p=0)
colNorms(DSM_TermContextMatrix, "minkowski", p=0)
sum(colNorms(DSM_TermContextMatrix, "minkowski", p=0)) # = nnzero(DSM_TermContextMatrix)
```

`rsvd` *Randomized Singular Value Decomposition (wordspace)*

Description

An implementation of the randomized truncated SVD algorithm of Halko, Martinsson & Tropp (2009).

Usage

```
rsvd(M, n, q = 2, oversampling = 2, transpose = FALSE, verbose = FALSE)
```

Arguments

`M`	a dense or sparse numeric matrix
`n`	an integer specifying the desired number of singular components. This argument must be specified and must satisfy `n <= min(nrow(M), ncol(M))`.
`q`	number of power iterations (Halko *et al.* recommend `q=1` or `q=2`)
`oversampling`	oversampling factor. The rSVD algorithm computes an approximate SVD factorization of rank `n * oversampling`, which is then truncated to the first `n` components.
`transpose`	if `TRUE`, apply the rSVD algorithm to the transpose `t(M)`, which may be more efficient depending on the dimensions of `M`
`verbose`	whether to display progress messages during execution

Details

This implementation of randomized truncated SVD is based on the randomized PCA algorithm (Halko *et al.* 2009, p. 9). The discussion in Sec. 4 and 5 of the paper shows that the same algorithm applies to the case where the columns of A are not centered (Algorithm 4.3 + Algorithm 5.1).

Value

A list with components

`u`	a matrix whose columns contain the first `n` left singular vectors of `M`
`v`	a matrix whose columns contain the first `n` right singular vectors of `M`
`d`	a vector containing the first `n` singular values of `M`

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

References

Halko, N., Martinsson, P. G., and Tropp, J. A. (2009). Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions. Technical Report 2009-05, ACM, California Institute of Technology.

See Also

svd, dsm.projection, sparsesvd

`scaleMargins` *Scale Rows and/or Columns of a Matrix (wordspace)*

Description

This function provides a fast and memory-efficient way to scale the rows and/or columns of a dense or sparse matrix. Each row is multiplied with the corresponding element of the vector `rows`, each column with the corresponding element of the vector `cols`.

Usage

```
scaleMargins(M, rows=NULL, cols=NULL, duplicate=TRUE)
```

Arguments

`M`	a dense or sparse matrix in canonical format
`rows`	a numeric vector with length equal to the number of rows of M, or a single number. If missing or `NULL`, the rows of M are not rescaled.
`cols`	a numeric vector with length equal to the number of columns of M, or a single number. If missing or `NULL`, the columns of M are not rescaled.
`duplicate`	if `FALSE`, modify the matrix `M` in place. Don't ever set this argument to `FALSE`.

Details

If `M` is not in canonical format (dense numeric matrix or sparse matrix of class `dgCMatrix`), it will automatically be converted. In this case, the precise behaviour of `duplicate=FALSE` is undefined.

`duplicate=FALSE` is intended for internal use only.

Value

The rescaled dense or sparse matrix.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

dsm.is.canonical

Examples

```
M <- matrix(1, 5, 3)
scaleMargins(M, rows=1:5, cols=c(1, 10, 100))
```

SemCorWSD *SemCor Word Sense Disambiguation Task (wordspace)*

Description

A collection of sentences containing ambiguous words manually labelled with WordNet senses. The data were obtained from the SemCor corpus version 3.0.

Usage

```
SemCorWSD
```

Format

A data frame with 647 rows and the following 8 columns (all of type character):

`id` Unique item ID

`target` The target word (lemmatized)

`pos` Word class of the target word (`n`, `v` or `a`)

`sense` Sense of the target word in this sentence (given as a WordNet lemma)

`gloss` WordNet definition of this sense

`sentence` The sentence containing the ambiguous word

`hw` Lemmatized form of the sentence ("headwords"); punctuation marks are excluded and all remaining words are case-folded

`lemma` Lemmatized and POS-disambiguated form in CWB/Penn format, e.g. `move_N` for the headword *move* used as a noun

Details

Target words and senses had to meet the following criteria in order to be included in the data set:

- sense occurs $f \geq 5$ times in SemCor 3.0
- sense accounts for at least 10% of all occurrences of the target
- at least two senses of target remain after previous two filters

SemCorWSD contains sentence contexts for the following target words:

- ambiguous nouns from Schütze (1998): *interest*, *plant*, *space*, *vessel*
- misc. ambiguous nouns: *bank*
- misc. ambiguous verbs: *find*, *grasp*, *open*, *run*

Source

TODO (SemCor reference, NLTK extraction)

References

Schütze, Hinrich (1998). Automatic word sense discrimination. *Computational Linguistics*, **24**(1), 97–123.

See Also

context.vectors

Examples

```
with(SemCorWSD, table(sense, target))

# all word senses with brief definitions ("glosses")
with(SemCorWSD, sort(unique(paste0(target, " ", sense, ": ", gloss))))
```

---

signcount *Efficiently Count Positive, Negative and Zero Values (wordspace)*

Description

This function counts the number of positive, negative and zero elements in a numeric vector or matrix, including some types of dense and sparse representations in the Matrix package.

It can be used to test for non-negativity and compute nonzero counts efficiently, without any memory overhead.

Usage

```
signcount(x, what = c("counts", "nonneg", "nnzero"))
```

Arguments

x	a numeric vector or array, or a numeric Matrix (supported formats are dgeMatrix, dgCMatrix and dgRMatrix)
what	whether to return the counts of positive, negative and zero elements (counts), the number of nonzero elements (nnzero), or to test for non-negativity (nonneg); see 'Value' below

Details

x must not contain any undefined values; signcount does not check whether this is actually the case.

Value

what="counts" A labelled numeric vector of length 3 with the counts of positive (pos), zero (zero) and negative (neg) values.

what="nonneg" A single logical value: TRUE if x is non-negative, FALSE otherwise.

what="nonzero" A single numeric value, the total number of nonzero elements in x.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

Examples

```
signcount(DSM_TermTermMatrix)     # dense matrix
signcount(DSM_TermContextMatrix)  # sparse dgCMatrix
signcount(DSM_TermContextMatrix, "nonneg") # confirm that it is non-negative
```

---

subset.dsm *Subsetting Distributional Semantic Models (wordspace)*

Description

Filter the rows and/or columns of a DSM object according to user-specified conditions.

Usage

```
## S3 method for class 'dsm'
subset(x, subset, select, recursive = FALSE, drop.zeroes = FALSE,
       matrix.only = FALSE, envir = parent.frame(), run.gc = FALSE, ...)
```

Arguments

x	an object of class dsm
subset	Boolean expression or index vector selecting a subset of the rows; the expression can use variables term and f to access target terms and their marginal frequencies, nnzero for the number of nonzero elements in each row, further optional variables from the row information table, as well as global variables such as the sample size N

`select`	Boolean expression or index vector selecting a subset of the columns; the expression can use variables `term` and `f` to access feature terms and their marginal frequencies, `nnzero` for the number of nonzero elements in each column, further optional variables from the column information table, as well as global variables such as the sample size `N`
`recursive`	if `TRUE` and both `subset` and `select` conditions are specified, the `subset` is applied repeatedly until the DSM no longer changes. This is typically needed if conditions on nonzero counts or row/column norms are specified, which may be affected by the subsetting procedure.
`drop.zeroes`	if `TRUE`, all rows and columns without any nonzero entries after subsetting are removed from the model (nonzero counts are based on the score matrix S if available, raw cooccurrence frequencies M otherwise)
`matrix.only`	if `TRUE`, return only the selected subset of the score matrix S (if available) or frequency matrix M, not a full DSM object. This may conserve a substantial amount of memory when processing very large DSMs.
`envir`	environment in which the `subset` and `select` conditions are evaluated. Defaults to the context of the function call, so all variables visible there can be used in the expressions.
`run.gc`	whether to run the garbage collector after each iteration of a recursive subset (`recursive=TRUE`) in order to keep memory overhead as low as possible. This option should only be specified if memory is very tight, since garbage collector runs can be expensive (e.g. when there are many distinct strings in the workspace).
`...`	any further arguments are silently ignored

Value

An object of class `dsm` containing the specified subset of the model `x`.

If necessary, counts of nonzero elements for each row and/or column are updated automatically.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

dsm

Examples

```
print(DSM_TermContext$M)
model <- DSM_TermContext

subset(model, nchar(term) <= 4)$M     # short target terms
subset(model, select=(nnzero <= 3))$M # columns with <= 3 nonzero cells

subset(model, nchar(term) <= 4, nnzero <= 3)$M # combine both conditions
```

```
subset(model, nchar(term) <= 4, nnzero >= 2)$M # still three columns with nnzero < 2
subset(model, nchar(term) <= 4, nnzero >= 2, recursive=TRUE)$M
```

`t.dsm` *Swap the Rows and Columns of a DSM Object (wordspace)*

Description

Given a distributional model `x`, `t(x)` returns a new DSM object representing the transposed co-occurrence and/or score matrix. Marginal frequencies and other row/column information are updated accordingly.

Usage

```
## S3 method for class 'dsm'
t(x)
```

Arguments

`x`	an object of class `dsm`

Value

A `dsm` object with rows and columns swapped.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`dsm`

Examples

```
tdm <- DSM_TermContext # term-document model
tdm$M

dtm <- t(tdm) # document-term model
dtm$M
```

`WordSim353` *Similarity Ratings for 351 Noun Pairs (wordspace)*

Description

A database of human similarity ratings for 351 English noun pairs, collected by Finkelstein et al. (2002) and annotated with semantic relations (similarity vs. relatedness) by Agirre et al. (2009).

Usage

```
WordSim353
```

Format

A data frame with 351 rows and the following 6 columns:

`word1` first noun (character)

`word2` second noun (character)

`score` average similarity rating by human judges on scale from 0 to 10 (numeric)

`relation` semantic relation between first and second word (factor, see Details below)

`similarity` whether word pair belongs to the *similarity* subset (logical)

`relatedness` whether word pair belongs to the *relatedness* subset (logical)

The nouns are given as disambiguated lemmas in the form `<headword>_N`.

Details

The data set is known as `WordSim353` because it originally consisted of 353 noun pairs. One duplicate entry (*money–cash*) as well as the trivial combination *tiger–tiger* (which may have been included as a control item) have been omitted in the present version, however.

The following semantic relations are distinguished in the `relation` variable: `synonym`, `antonym`, `hypernym`, `hyponym`, `co-hyponym`, `holonym`, `meronym` and `other` (topically related or completely unrelated).

Note that the *similarity* and *relatedness* subsets are not disjoint, because they share 103 unrelated noun pairs (semantic relation `other` and score below 5.0).

Source

Similarity ratings (Finkelstein *et al.* 2002): https://gabrilovich.com/resources/data/wordsim353/wordsim353.html

Semantic relations (Agirre *et al.* 2009): *download no longer available*

References

Agirre, Eneko, Alfonseca, Enrique, Hall, Keith, Kravalova, Jana, Pasca, Marius, and Soroa, Aitor (2009). A study on similarity and relatedness using distributional and WordNet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-HLT 2009)*, pages 19–27, Boulder, Colorado.

Finkelstein, Lev, Gabrilovich, Evgeniy, Matias, Yossi, Rivlin, Ehud, Solan, Zach, Wolfman, Gadi, and Ruppin, Eytan (2002). Placing search in context: The concept revisited. *ACM Transactions on Information Systems*, **20**(1), 116–131.

Examples

```
head(WordSim353, 20)

table(WordSim353$relation) # semantic relations

# split into "similarity" and "relatedness" subsets
xtabs(~ similarity + relatedness, data=WordSim353)
```

`wordspace.openmp` *Control multi-core processing in wordspace functions (wordspace)*

Description

Control whether multi-core processing is used by wordspace functions (if available) and how many threads are run in parallel. See "Details" below for more information.

Usage

```
wordspace.openmp(threads = NULL)
```

Arguments

`threads`	if specified, number of parallel threads to be used for multi-core processing

Details

The `wordspace` package has experimental support for multi-core processing using OpenMP on some platforms. So far, only the `dist.matrix` function uses multi-core processing (for all distance measures except `cosine`).

Even where supported, OpenMP is not enabled by default and has to be activated explicitly with `wordspace.openmp(threads=N)`, where `N` is the number of parallel threads to be used.

Call `wordspace.openmp()` without arguments to find out whether OpenMP is supported on your platform and obtain information about the maximum number of threads available as well as the current setting.

Note that multi-threading of other R packages and functions (such as optimised matrix algebra in the BLAS library) is never affected by this function.

Value

If `threads` is unspecified or `NULL`, a data frame with a single row and the following information is returned:

`available`	`TRUE` if OpenMP multi-core support is available
`max`	maximum number of threads that can be used (0 if not available)
`threads`	currently selected number of threads (defaults to 1 if not available)

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

`dist.matrix`

Examples

```
wordspace.openmp()
```

---

write.dsm.matrix *Export DSM Matrix to File (wordspace)*

Description

This function exports a DSM matrix to a disk file in the specified format (see section 'Formats' for details).

Usage

```
write.dsm.matrix(x, file, format = c("word2vec"), round=FALSE,
                 encoding = "UTF-8", batchsize = 1e6, verbose=FALSE)
```

Arguments

`x`	a dense or sparse matrix representing a DSM, or an object of class `dsm`
`file`	either a character string naming a file or a `connection` open for writing (in text mode)
`format`	desired output file format. See section 'Formats' for a list of available formats and their limitations.
`round`	for some output formats, numbers can be rounded to the specified number of decimal digits in order to reduce file size

encoding	character encoding of the output file (ignored if file is a connection)
batchsize	for certain output formats, the matrix is written in batches of batchsize cells each in order to limit memory overhead
verbose	if TRUE, show progress bar when writing in batches

Details

In order to save text formats to a compressed file, pass a gzfile, bzfile or xzfile connection with appropriate encoding in the argument file. Make sure not to open the connection before passing it to write.dsm.matrix. See section 'Examples' below.

Formats

Currently, the only supported file format is word2vec.

word2vec This widely used text format for word embeddings is only suitable for a dense matrix. Row labels must be unique and may not contain whitespace. Values are usually rounded to a few decimal digits in order to keep file size manageable.

The first line of the file lists the matrix dimensions (rows, columns) separated by a single blank. It is followed by one text line for each matrix row, starting with the row label. The label and are cells are separated by single blanks, so row labels cannot contain whitespace.

Author(s)

Stephanie Evert (https://purl.org/stephanie.evert)

See Also

read.dsm.matrix

Examples

```
model <- dsm.score(DSM_TermTerm, score="MI", normalize=TRUE) # a typical DSM

# save in word2vec text format (rounded to 3 digits)
fn <- tempfile(fileext=".txt")
write.dsm.matrix(model, fn, format="word2vec", round=3)
cat(readLines(fn), sep="\n")

# save as compressed file in word2vec format
fn <- tempfile(fileext=".txt.gz")
fh <- gzfile(fn, encoding="UTF-8") # need to set file encoding here
write.dsm.matrix(model, fh, format="word2vec", round=3)
# write.dsm.matrix() automatically opens and closes the connection
cat(readLines(gzfile(fn)), sep="\n")
```

Index

∗ **classes**
dsm, 18
∗ **datasets**
DSM_GoodsMatrix, 31
DSM_HieroglyphsMatrix, 32
DSM_SingularValues, 33
DSM_TermContextMatrix, 34
DSM_TermTermMatrix, 35
DSM_Vectors, 36
DSM_VerbNounTriples_BNC, 37
ESSLLI08_Nouns, 38
RG65, 65
SemCorWSD, 70
WordSim353, 75
∗ **file**
read.dsm.matrix, 59
read.dsm.triplet, 60
read.dsm.ucs, 64
write.dsm.matrix, 77
∗ **hplot**
plot.eval.similarity.correlation, 55
∗ **methods**
as.distmat, 4
as.dsm, 5
∗ **package**
wordspace-package, 3
∗ **utilities**
signcount, 71

as.dist, *4*
as.distmat, 4, *50*, *52*
as.dsm, 5, *6*
as.dsm.DocumentTermMatrix, *6*
as.dsm.DocumentTermMatrix *(*as.dsm.tm*)*, 6
as.dsm.TermDocumentMatrix, *6*
as.dsm.TermDocumentMatrix *(*as.dsm.tm*)*, 6
as.dsm.tm, 6
as.matrix.dsm, 7

bzfile, *59*, *78*

cbind.dsm *(*rbind.dsm*)*, 58
check.dsm, 8, *14*, *15*
colNorms, *51*
colNorms *(*rowNorms*)*, 66
connection, *59*, *77*
context.vectors, 9, *71*
convert.lemma, 11, *38–43*, *45*

denseMatrix, *22*
dgCMatrix, *21*, *22*, *34*, *71*
dgeMatrix, *71*
dgRMatrix, *71*
dgTMatrix, *21*, *22*
dim.dsm, *8*, 13, *15*
dimnames.dsm, *8*, 14, *14*
dimnames<-.dsm *(*dimnames.dsm*)*, 14
dist, *15*, *17*
dist.matrix, *3–5*, 15, *42*, *44*, *45*, *49*, *50*, *52*, *53*, *67*, *77*
dMatrix, *19*
DocumentTermMatrix, *6*
dsm, *3*, *5*, *6*, *8*, *9*, *14*, *15*, 18, *30*, *34*, *35*, *62–65*, *73*, *74*
dsm-object *(*dsm*)*, 18
dsm.canonical.matrix, *20*, 21, *21*
dsm.is.canonical, *8*, *70*
dsm.is.canonical *(*dsm.canonical.matrix*)*, 21
dsm.projection, 23, *36*, *69*
dsm.score, 25, *62*
DSM_GoodsMatrix, 31
DSM_HieroglyphsMatrix, 32
DSM_SingularValues, 33
DSM_TermContext, *35*
DSM_TermContext *(*DSM_TermContextMatrix*)*, 34
DSM_TermContextMatrix, 34, *35*
DSM_TermTerm, *34*

DSM_TermTerm *(DSM_TermTermMatrix)*, 35
DSM_TermTermMatrix, *34*, 35
DSM_Vectors, 36
DSM_VerbNounTriples_BNC, *32*, 37

ESSLLI08_Nouns, *36*, 38, *40*
eval.clustering, 39
eval.multiple.choice, 41, *53*
eval.similarity.correlation, 43, *53*, *55*, *56*

file, *61*, *64*

gzfile, *59*, *78*

head, *5*, *18*, *46*, *47*
head.dist.matrix, 45
head.dsm, 46

isoMDS, *55*

lowess, *55*, *56*

match.split, 47
Matrix, *71*
matrix, *4*, *22*
merge.dsm, 48

nearest.neighbors *(*nearest.neighbours*)*, 49
nearest.neighbours, *3–5*, *18*, 49, *53*, *55*
normalize.cols *(*normalize.rows*)*, 51
normalize.rows, *26*, 51, *67*

pair.distances, *4*, *5*, *18*, *40*, *42*, *44*, *45*, 52
plot, *5*, *18*, *44*
plot.default, *56*
plot.dist.matrix, 54
plot.eval.similarity.correlation, *45*, 55
print, *44*, *57*
print.dsm, *9*, *14*, 57
print.eval.similarity.correlation, *45*
print.eval.similarity.correlation *(*plot.eval.similarity.correlation*)*, 55

rbind.dsm, 58
read.dsm.matrix, 59, *78*
read.dsm.triplet, *3*, *21*, 60, *60*, *65*
read.dsm.ucs, *21*, *60*, *63*, 64
RG65, *36*, *43*, *45*, 65
rowNorms, *17*, *18*, *26*, *51*, *52*, 66
rsvd, *25*, *36*, 68

sammon, *55*
scaleMargins, 69
scan, *61*
SemCorWSD, *11*, *36*, 70
signcount, 71
sparseMatrix, *4*, *19*
sparsesvd, *25*, *69*
subset.dsm, 72
svd, *69*

t.dsm, 74
tail, *46*, *47*
TermDocumentMatrix, *6*

WordSim353, *36*, *43*, *45*, 75
wordspace *(*wordspace-package*)*, 3
wordspace-package, 3
wordspace.openmp, 76
write.dsm.matrix, *60*, 77

xzfile, *59*, *78*