Package 'LSAfun'

May 7, 2026

Type Package

Title Applied Latent Semantic Analysis (LSA) Functions

Description Provides functions that allow for convenient working with vector space models of semantics/distributional semantic models/word embeddings.
Originally built for LSA models (hence the name), but can be used for all such vector-based models.
For actually building a vector semantic space, use the package 'lsa' or other specialized software.
Downloadable semantic spaces can be found at <https://sites.google.com/site/fritzgntr/software-resources>.

Version 0.8.1

Date 2025-03-18

Depends R (>= 3.1.0), lsa, rgl

License GPL (>= 2)

LazyData true

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation no

Author Fritz Guenther [aut, cre]

Maintainer Fritz Guenther <fritz.guenther@uni-tuebingen.de>

Repository CRAN

Date/Publication 2025-04-02 17:50:10 UTC

Contents

`LSAfun-package` *Computations based on Latent Semantic Analysis*

Description

Offers methods and functions for working with Vector Space Models of semantics/distributional semantic models/word embeddings. The package was originally written for Latent Semantic Analysis (LSA), but can be used with all vector space models. Such models are created by algorithms working on a corpus of text documents. Those algorithms achieve a high-dimensional vector representation for word (and document) meanings. The exact LSA algorithm is described in Martin & Berry (2007).

Such a representation allows for the computation of word (and document) similarities, for example by computing cosine values of angles between two vectors.

The focus of this package

This package is not designed to create LSA semantic spaces. In R, this functionality is provided by the package `lsa`. The focus of the package *LSAfun* is to provide functions to be applied on existing LSA (or other) semantic spaces, such as

1. Similarity Computations
2. Neighborhood Computations
3. Applied Functions
4. Composition Methods

Video Tutorials

A video tutorial for this package can be found here: https://youtu.be/IlwIZvM2kg8

A video tutorial for using this package with vision-based representations from deep convolutional neural networks can be found here: https://youtu.be/0PNrXraWfzI

How to obtain a semantic space

LSAfun comes with one example LSA space, the wonderland space.

This package can also directly use LSA semantic spaces created with the `lsa`-package. Thus, it allows the user to use own LSA spaces. (Note that the function `lsa` gives a list of three matrices. Of those, the term matrix `U` should be used.)

The `lsa` package works with (very) small corpora, but gets difficulties in scaling up to larger corpora. In this case, it is recommended to use specialized software for creating semantic spaces, such as

- S-Space (Jurgens & Stevens, 2010), available here
- SemanticVectors (Widdows & Ferraro, 2008), available here
- gensim (Rehurek & Sojka, 2010), available here
- DISSECT (Dinu, Pham, & Baroni, 2013), available here

Downloading semantic spaces: Another possibility is to use one of the semantic spaces provided at https://sites.google.com/site/fritzgntr/software-resources. These are stored in the `.rda` format. To load one of these spaces into the `R` workspace, save them into a directory, set the working directory to that directory, and load the space using `load()`.

Author(s)

Fritz Guenther

`analogy` *Analogy*

Description

Implements the *king* - *man* + *woman* = *queen* analogy solving algorithm

Usage

```
analogy(x1,x2,y1=NA,n,tvectors=tvectors)
```

Arguments

x1	a character vector specifying the first word of the first pair (*man* in *man : king = woman : ?*)
x2	a character vector specifying the second word of the first pair (*king* in *man : king = woman : ?*)
y1	a character vector specifying the first word of the second pair (*woman* in *man : king = woman : ?*)
n	the number of neighbors to be computed
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

The analogy task is a popular benchmark for vector space models of meaning/word embeddings. It is based on the rationale that proportinal analogies *x1 is to x2 as y1 is to y2*, like *man : king = woman : ?* (correct answer: *queen*), can be solved via the following operation on the respective word vectors (all normalized to unit norm) `king - man + woman = queen` (that is, the nearest vector to `king - man + woman` should be `queen`) (Mikolov et al., 2013).

The `analogy()` function comes in two variants, taking as input either three words (`x1`, `x2`, and `y1`) or two words (`x1` and `x2`)

- The variant with three input words (`x1`, `x2`, and `y1`) implements the standard analogy solving algorithm for analogies of the type `x1 : x2 = y1 : ?`, searching the `n` nearest neighbors for `x2 - x1 + y1` (all normalized to unit norm) as the best-fitting candidates for `y2`
- The variant with two input words (`x1` and `x2`) only computes the difference between the two vectors (both normalized to unit norm) and the `n` nearest neighbors to the resulting difference vector

Value

Returns a list containing a numeric vector and the nearest neighbors to that vector:

- In the variant with three input words (`x1`, `x2`, and `y1`), returns:
  - `y2_vec` The result of `x2 - x1 + y1` (all normalized to unit norm) as a numeric vector
  - `y2_neighbors` A named numeric vector of the `n` nearest neighbors to `y2_vec`. The neighbors are given as names of the vector, and their respective cosines to `y2_vec` as vector entries.
- In the variant with two input words (`x1` and `x2`), returns:
  - `x_diff_vec` The result of `x2 - x1` (both normalized to unit norm) as a numeric vector
  - `x_diff_neighbors` A named numeric vector of the `n` nearest neighbors to `x_diff_vec`. The neighbors are given as names of the vector, and their respective cosines to `x_diff_vec` as vector entries.

Author(s)

Fritz Guenther

References

Mikolov, T., Yih, W. T., & Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics.

See Also

neighbors

Examples

```
data(wonderland)

analogy(x1="hatter",x2="mad",y1="cat",n=10,tvectors=wonderland)

analogy(x1="hatter",x2="mad",n=10,tvectors=wonderland)
```

---

asym *Asymmetric Similarity functions*

Description

Compute various asymmetric similarities between words

Usage

```
asym(x,y,method,t=0,tvectors)
```

Arguments

x	A single word, given as a character of `length(x) = 1`
y	A single word, given as a character of `length(y) = 1`
method	Specifying the formula to use for asymmetric similarity computation
t	A numeric threshold a dimension value of the vectors has to exceed so that the dimension is considered *active*; not needed for the `kintsch` method
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Asymmetric (or directional) similarities can be useful e.g. for examining *hypernymy* (category inclusion), for example the relation between *dog* and *animal* should be asymmetrical. The general idea is that, if one word is a hypernym of another (i.e. it is semantically narrower), then a significant number of dimensions that are salient in this word should also be salient in the semantically broader term (Lenci & Benotto, 2012).

In the formulas below, $w_x(f)$ denotes the value of vector x on dimension f. Furthermore, F_x is the set of *active* dimensions of vector x. A dimension f is considered active if $w_x(f) > t$, with t being a pre-defined, free parameter.

The options for `method` are defined as follows (see Kotlerman et al., 2010) (1)):

- `method = "weedsprec"`

$$weedsprec(u, v) = \frac{\sum_{f \in F_u \cap F_v} w_u(f)}{\sum_{f \in F_u} w_u(f)}$$

- `method = "cosweeds"`

$$cosweeds(u, v) = \sqrt{weedsprec(u, v) \times cosine(u, v)}$$

- `method = "clarkede"`

$$clarkede(u, v) = \frac{\sum_{f \in F_u \cap F_v} min(w_u(f), w_v(f))}{\sum_{f \in F_u} w_u(f)}$$

- `method = "invcl"`

$$invcl(u, v) = \sqrt{clarkede(u, v) \times (1 - clarkede(u, v))}$$

- `method = "kintsch"`
  Unlike the other methods, this one is not derived from the logic of hypernymy, but rather from asymmetrical similarities between words due to different amounts of knowledge about them. Here, asymmteric similarities between two words are computed by taking into account the vector length (i.e. the amount of information about those words). This is done by projecting one vector onto the other, and normalizing this resulting vector by dividing its length by the length of the longer of the two vectors (Details in Kintsch, 2014, see References).

Value

A numeric giving the asymmetric similarity between `x` and `y`

Author(s)

Fritz Guenther

References

Kintsch, W. (2015). Similarity as a Function of Semantic Distance and Amount of Knowledge. *Psychological Review, 121,* 559-561.

Kotlerman, L., Dagan, I., Szpektor, I., & Zhitomirsky-Geffet, M (2010). Directional distributional similarity for lexical inference. *Natural Language Engineering, 16,* 359-389.

Lenci, A., & Benotto, G. (2012). Identifying hypernyms in distributional semantic spaces. In *Proceedings of *SEM* (pp. 75-79), Montreal, Canada.

See Also

`Cosine` `conSIM`

Examples

```
data(wonderland)

asym("alice","girl",method="cosweeds",t=0,tvectors=wonderland)
asym("alice","rabbit",method="cosweeds",tvectors=wonderland)
```

`centroid_analysis` *Centroid Analysis*

Description

Performs a centroid analysis for a set of words

Usage

```
centroid_analysis(responses,targets = NULL,split=" ",unique.responses = FALSE,
reference.list = NULL,verbose = FALSE,rank.responses = FALSE,
tvectors=tvectors)
```

Arguments

`responses`	a character vector specifying multiple single words
`targets`	(optional:) a character vector specifying one or multiple single words
`split`	a character vector defining the character used to split the input strings into individual words (white space by default)
`unique.responses`	If `TRUE`, duplicated words in `responses` are discarded when computing the the centroid. `FALSE` by default, so multiple instances of the same word will be included.
`reference.list`	(optional:) A list of words in reference to which the neighborhood ranks are computed: Only entries in `reference.list` will be considered as possible neighbors. Only relevant when target words are provided in `target`. if `reference.list = NULL` (default), then `rownames(tvectors)` (all words in the semantic space) will be considered when computing ranks.
`verbose`	If `TRUE` (default: FALSE), a message will appear that specifies for which target the neighborhood ranks are currently being computed
`rank.responses`	If `FALSE` (default), responses themselves will not be considered for computing the neighborhood rank.
`tvectors`	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

The centroid analysis computes the average vector for a set of words. The intended use case is that these words are responses towards a given concept; the centroid then serves as the estimated vector representation for that concept.

Value

An object of class `centroid_analysis`. This object is a list consisting of:

`$centroid`	The centroid of the response vectors
`$cosines`	The cosine similarity between the response centroid and each target vector
`$ranks.target`	The rank of the response centroid *in the neighborhood of each target vector*, with reference to `reference.list`
`$ranks.centroid`	The rank of each target *in the neighborhood of the response centroid*, with reference to `reference.list`

Author(s)

Fritz Guenther, Aliona Petrenco

References

Pugacheva, V., & GÃ¼nther, F. (2024). Lexical choice and word formation in a taboo game paradigm. *Journal of Memory and Language, 135,* 104477.

See Also

`cosine`, `Cosine`, `neighbors`

Examples

```
data(wonderland)
centroid_analysis(responses=c("mouse","rabbit","cat","king","queen"),targets=c("alice","hare"),
          tvectors=wonderland)
```

---

choose.target *Random Target Selection*

Description

Randomly samples words within a given similarity range to the input

Usage

```
choose.target(x,lower,upper,n,tvectors=tvectors)
```

Arguments

`x`	a character vector of `length(x) = 1` specifying a word or a sentence/document
`lower`	the lower bound of the similarity range; a numeric
`upper`	the upper bound of the similarity range; a numeric
`n`	an integer giving the number of target words to be sampled
`tvectors`	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Computes cosine values between the input `x` and all the word vectors in `tvectors`. Then only selects words with a cosine similarity between `lower` and `upper` to the input, and randomly samples `n` of these words.

This function is designed for randomly selecting target words with a predefined similarity towards a given prime word (or sentence/document).

Value

A named numeric vector. The names of the vector give the target words, the entries their respective cosine similarity to the input.

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

See Also

`cosine`, `Cosine`, `neighbors`

Examples

```
data(wonderland)

choose.target("mad hatter",lower=.2,upper=.3,
                n=20, tvectors=wonderland)
```

`coherence` *Coherence of a text*

Description

Computes coherence of a given paragraph/document

Usage

```
coherence(x,split=c(".","!","?"),tvectors=tvectors, remove.punctuation=TRUE,
stopwords = NULL, method ="Add")
```

Arguments

x	a character vector of length(x) = 1 containing the document
split	a vector of expressions that determine where to split sentences
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
remove.punctuation	removes punctuation from x *after* splitting the sentences; TRUE by default
stopwords	a character vector defining a list of words that are *not* used to compute the sentence vectors for x
method	the compositional model to compute the document vector from its word vectors. The default option method = "Add" computes the document vector as the vector sum. With method = "Multiply", the document vector is computed via element-wise multiplication (see compose).

Details

This function applies the method described in Landauer & Dumais (1997): The *local coherence* is the cosine between two adjacent sentences. The *global coherence* is then computed as the mean value of these local coherences.

The format of x should be of the kind x <- "sentence1. sentence2. sentence3" Every sentence can also just consist of one single word.

To import a document Document.txt to from a directory for coherence computation, set your working directory to this directory using setwd(). Then use the following command lines:

```
fileName1 <- "Alice_in_Wonderland.txt"
x <- readChar(fileName1, file.info(fileName1)$size)
```

In the traditional LSA approach, the vector *D* for a document (or a sentence) consisting of the words *(t1, . , tn)* is computed as

$$D = \sum_{i=1}^{n} t_n$$

This is the default method (method="Add") for this function. Alternatively, this function provided the possibility of computing the document vector from its word vectors using element-wise multiplication (see Mitchell & Lapata, 2010 and compose).

A note will be displayed whenever not all words of one input string are found in the semantic space. ***Caution:*** In that case, the function will still produce a result, by omitting the words not found in the semantic space. Depending on the specific requirements of a task, this may compromise the results. Please check your input when you receive this message.

A warning message will be displayed whenever no word of one input string is found in the semantic space.

Value

A list of two elements; the first element ($local) contains the local coherences as a numeric vector, the second element ($global) contains the global coherence as a numeric.

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Mitchell, J., & Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science, 34,* 1388-1429.

See Also

cosine, Cosine, costring

Examples

```
data(wonderland)

coherence ("there was certainly too much of it in the air. even the duchess
sneezed occasionally; and as for the baby, it was sneezing and howling
alternately without a moment's pause. the only things in the kitchen
that did not sneeze, were the cook, and a large cat which was sitting on
the hearth and grinning from ear to ear.",
tvectors=wonderland)
```

---

compose *Two-Word Composition*

Description

Computes the vector of a complex expression p consisting of two single words u and v, following the methods examined in Mitchell & Lapata (2008) (see *Details*).

Usage

```
## Default
compose(x,y,method="Add", a=1,b=1,c=1,m,k,lambda=2,
      tvectors=tvectors, norm="none")
```

Arguments

x	a single word (character vector with `length(x) = 1`)
y	a single word (character vector with `length(y) = 1`)
a, b, c	weighting parameters, see *Details*
m	number of nearest words to the Predicate that are initially activated (see Predication)

k	size of the k-neighborhood; k $\leq$ m (see Predication)
lambda	dilation parameter for method = "Dilation"
method	the composition method to be used (see *Details*)
norm	whether to normalize the single word vectors before applying a composition function. Setting norm = "none" will not perform any normalizations, setting norm = "all" will normalize every involved word vector. Setting norm = "block" is only valid for the Predication method
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Let p be the vector with entries p_i for the two-word phrase consisiting of u with entries u_i and v with entries v_i. The different composition methods as described by Mitchell & Lapata (2008, 2010) are as follows:

- Additive Model (method = "Add")
$$p_i = u_i + v_i$$
- Weighted Additive Model (method = "WeightAdd")
$$p_i = a * u_i + b * v_i$$
- Multiplicative Model (method = "Multiply")
$$p_i = u_i * v_i$$
- Combined Model (method = "Combined")
$$p_i = a * u_i + b * v_i + c * u_i * v_i$$
- Predication (method = "Predication") (see Predication)
If method="Predication" is used, x will be taken as Predicate and y will be taken as Argument of the phrase (see *Examples*)
- Circular Convolution (method = "CConv")
$$p_i = \sum_j u_j * v_{i-j}$$
,
where the subscripts of v are interpreted modulo n with $n =$ length(x)(= length(y))
- Dilation (method = "Dilation")
$$p = (u * u) * v + (\lambda - 1) * (u * v) * u$$
,
with $(u * u)$ being the dot product of u and u (and $(u * v)$ being the dot product of u and v).

The Add, Multiply, and CConv methods are *symmetrical* composition methods,

i.e. compose(x="word1",y="word2") will give the same results as compose(x="word2",y="word1")

On the other hand, WeightAdd, Combined, Predication and Dilation are *asymmetrical*, i.e. compose(x="word1",y="word2") will give different results than compose(x="word2",y="word1")

Value

The phrase vector as a numeric vector

Author(s)

Fritz Guenther

References

Kintsch, W. (2001). Predication. *Cognitive science, 25,* 173-202.

Mitchell, J., & Lapata, M. (2008). Vector-based Models of Semantic Composition. In *Proceedings of ACL-08: HLT* (pp. 236-244). Columbus, Ohio.

Mitchell, J., & Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science, 34,* 1388-1429.

See Also

Predication

Examples

```
data(wonderland)

compose(x="mad",y="hatter",method="Add",tvectors=wonderland)

compose(x="mad",y="hatter",method="Combined",a=1,b=2,c=3,
tvectors=wonderland)

compose(x="mad",y="hatter",method="Predication",m=20,k=3,
tvectors=wonderland)

compose(x="mad",y="hatter",method="Dilation",lambda=3,
tvectors=wonderland)
```

---

conSIM *Similarity in Context*

Description

Compute Similarity of a word with a set of two other test words, given a third context word

Usage

```
conSIM(x,y,z,c,tvectors=tvectors)
```

Arguments

x	The relevant word, given as a character of length(x) = 1
y, z	The two test words, given each as a character of length(y) = 1
c	The context word in respect to which the similarity of x to y and z is to be computed (a character of length(y) = 1)
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Following the example from Kintsch (2014): If one has to judge the similarity between *France* one the one hand and the test words *Germany* and *Spain* on the other hand, this similarity judgement varies as a function of a fourth context word. If *Portugal* is given as a context word, *France* is considered to be more similar to *Germany* than to *Spain*, and vice versa for the context word *Poland*. Kintsch (2014) proposed a context sensitive, asymmetrical similarity measure for cases like this, which is implemented here

Value

A list of two similarity values

SIM_XY_zc: Similarity of x and y, given the alternative z and the context c

SIM_XZ_yc: Similarity of x and z, given the alternative y and the context c

Author(s)

Fritz Guenther

References

Kintsch, W. (2015). Similarity as a Function of Semantic Distance and Amount of Knowledge. *Psychological Review, 121,* 559-561.

Tversky, A. (1977). Features of similarity. *Psychological Review, 84,* 327-352.

See Also

Cosine asym

Examples

```
data(wonderland)

conSIM(x="rabbit",y="alice",z="hatter",c="dormouse",tvectors=wonderland)
```

`Cosine` *Compute cosine similarity*

Description

Computes the cosine similarity for two single words

Usage

```
Cosine(x,y,tvectors=tvectors)
```

Arguments

`x`	A single word, given as a character of `length(x) = 1`
`y`	A single word, given as a character of `length(y) = 1`
`tvectors`	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Instead of using numeric vectors, as the `cosine()` function from the lsa package does, this function allows for the direct computation of the cosine between two single words (i.e. Characters). which are automatically searched for in the LSA space given in as `tvectors`.

Value

The cosine similarity as a numeric

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Dennis, S. (2007). How to use the LSA Web Site. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of Latent Semantic Analysis* (pp. 35-56). Mahwah, NJ: Erlbaum.

http://wordvec.colorado.edu/

See Also

`distance` `asym`

Examples

```
data(wonderland)

Cosine("alice","rabbit",tvectors=wonderland)
```

`costring` *Sentence Comparison*

Description

Computes cosine values between sentences and/or documents

Usage

```
costring(x,y,tvectors=tvectors,split=" ",remove.punctuation=TRUE,
stopwords = NULL, method ="Add")
```

Arguments

`x`	a character vector
`y`	a character vector
`tvectors`	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
`split`	a character vector defining the character used to split the documents into words (white space by default)
`remove.punctuation`	removes punctuation from `x` and `y`; `TRUE` by default
`stopwords`	a character vector defining a list of words that are *not* used to compute the document/sentence vector for `x` and `y`
`method`	the compositional model to compute the document vector from its word vectors. The default option `method = "Add"` computes the document vector as the vector sum. With `method = "Multiply"`, the document vector is computed via element-wise multiplication (see `compose`).

Details

This function computes the cosine between two documents (or sentences) or the cosine between a single word and a document (or sentence).

In the traditional LSA approach, the vector *D* for a document (or a sentence) consisting of the words *(t1, . , tn)* is computed as

$$D = \sum_{i=1}^{n} t_n$$

This is the default method (`method="Add"`) for this function. Alternatively, this function provided the possibility of computing the document vector from its word vectors using element-wise multiplication (see Mitchell & Lapata, 2010 and `compose`).

The format of x (or y) can be of the kind x <- "word1 word2 word3" , but also of the kind x <- c("word1", "word2", "word3"). This allows for simple copy&paste-inserting of text, but also for using character vectors, e.g. the output of neighbors().

To import a document *Document.txt* to from a directory for comparisons, set your working directory to this directory using setwd(). Then use the following command lines:

```
fileName1 <- "Alice_in_Wonderland.txt"

x <- readChar(fileName1, file.info(fileName1)$size)
```

A note will be displayed whenever not all words of one input string are found in the semantic space. ***Caution:*** In that case, the function will still produce a result, by omitting the words not found in the semantic space. Depending on the specific requirements of a task, this may compromise the results. Please check your input when you receive this message.

A warning message will be displayed whenever no word of one input string is found in the semantic space.

Value

A numeric giving the cosine between the input documents/sentences

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Dennis, S. (2007). How to use the LSA Web Site. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of Latent Semantic Analysis* (pp. 35-56). Mahwah, NJ: Erlbaum.

Mitchell, J., & Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science, 34,* 1388-1429.

http://wordvec.colorado.edu/

See Also

cosine, Cosine, multicos, multidocs, multicostring

Examples

```
data(wonderland)
costring("alice was beginning to get very tired.",
      "a white rabbit with a clock ran close to her.",
      tvectors=wonderland)
```

distance *Compute distance*

Description

Computes distance metrics for two single words

Usage

```
distance(x,y,method="euclidean",tvectors=tvectors)
```

Arguments

x	A single word, given as a character of length(x) = 1
y	A single word, given as a character of length(y) = 1
method	Specifies whether to compute euclidean or cityblock metric
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Computes Minkowski metrics, i.e. geometric distances between the vectors for two given words. Possible options are euclidean for the Euclidean Distance, $d(x,y) = \sqrt{\sum (x-y)^2}$, and cityblock for the City Block metric, $d(x,y) = \sum |x-y|$

Value

The distance value as a numeric

Author(s)

Fritz Guenther

See Also

Cosine asym

Examples

```
data(wonderland)

distance("alice","rabbit",method="euclidean",tvectors=wonderland)
```

genericSummary *Summarize a text*

Description

Selects sentences from a text that best describe its topic

Usage

```
genericSummary(text,k,split=c(".","!","?"),min=5,...)
```

Arguments

text	A character vector of length(text) = 1 specifiying the text to be summarized
k	The number of sentences to be used in the summary
split	A character vector specifying which symbols determine the end of a sentence in the document
min	The minimum amount of words a sentence must have to be included in the computations
...	Further arguments to be passed on to textmatrix

Details

Applies the method of Gong & Liu (2001) for generic text summarization of text document *D* via Latent Semantic Analysis:

1. Decompose the document *D* into individual sentences, and use these sentences to form the candidate sentence set *S*, and set *k* = 1.
2. Construct the terms by sentences matrix *A* for the document *D*.
3. Perform the SVD on A to obtain the singular value matrix Σ, and the right singular vector matrix V^t. In the singular vector space, each sentence i is represented by the column vector $\psi_i = [v_i1, v_i2, ..., v_ir]^t$ of V^t.
4. Select the *k*'th right singular vector from matrix V^t.
5. Select the sentence which has the largest index value with the *k*'th right singular vector, and include it in the summary.
6. If *k* reaches the predefined number, terminate the op- eration; otherwise, increment *k* by one, and go to Step 4.

(Cited directly from Gong & Liu, 2001, *p.* 21)

Value

A character vector of the length *k*

Author(s)

Fritz Guenther

See Also

textmatrix, lsa, svd

Examples

```
D <- "This is just a test document. It is set up just to throw some random
sentences in this example. So do not expect it to make much sense. Probably, even
the summary won't be very meaningful. But this is mainly due to the document not being
meaningful at all. For test purposes, I will also include a sentence in this
example that is not at all related to the rest of the document. Lions are larger than cats."

genericSummary(D,k=1)
```

---

multicos *Vector x Vector Comparison*

Description

Computes a cosine matrix from given word vectors

Usage

```
multicos(x,y=x,tvectors=tvectors)
```

Arguments

x	a character vector or numeric of length=ncol(tvectors) (vector with same dimensionality as LSA space)
y	a character vector; y = x by default
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Submit a character vector consisting of *n* words to get a *n x n* cosine matrix of all their pairwise cosines.

Alternatively, submit two different character vectors to get their pairwise cosines. Single words are also possible arguments.

Also allows for computation of cosines between a given numeric vector with the same dimensionality as the LSA space and a vector consisting of n words.

Value

A matrix containing the pairwise cosines of x and y

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Dennis, S. (2007). How to use the LSA Web Site. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of Latent Semantic Analysis* (pp. 35-56). Mahwah, NJ: Erlbaum.

http://wordvec.colorado.edu/

See Also

cosine, Cosine, costring, multicostring

Examples

```
data(wonderland)
multicos("mouse rabbit cat","king queen",
          tvectors=wonderland)
```

---

multicostring *Sentence x Vector Comparison*

Description

Computes cosines between a sentence/ document and multiple words

Usage

```
multicostring(x,y,tvectors=tvectors,split=" ",remove.punctuation=TRUE,
stopwords = NULL, method ="Add")
```

Arguments

x	a character vector specifying a sentence/ document (or also a single word)
y	a character vector specifying multiple single words
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
split	a character vector defining the character used to split the documents into words (white space by default)

`remove.punctuation`
: removes punctuation from `x` and `y`; `TRUE` by default

`stopwords`
: a character vector defining a list of words that are *not* used to compute the document/sentence vector for `x`

`method`
: the compositional model to compute the document vector from its word vectors. The default option `method = "Add"` computes the document vector as the vector sum. With `method = "Multiply"`, the document vector is computed via element-wise multiplication (see `compose`).

Details

The format of `x` (or `y`) can be of the kind `x <- "word1 word2 word3"` , but also of the kind `x <- c("word1", "word2", "word3")`. This allows for simple copy&paste-inserting of text, but also for using character vectors, e.g. the output of `neighbors`.

Both x and y can also just consist of one single word. In the traditional LSA approach, the vector *D* for the document (or sentence) `x` consisting of the words *(t1, . , tn)* is computed as

$$D = \sum_{i=1}^{n} t_n$$

This is the default method (`method="Add"`) for this function. Alternatively, this function provided the possibility of computing the document vector from its word vectors using element-wise multiplication (see Mitchell & Lapata, 2010 and `compose`). See also `costring`).

A note will be displayed whenever not all words of one input string are found in the semantic space. ***Caution:*** In that case, the function will still produce a result, by omitting the words not found in the semantic space. Depending on the specific requirements of a task, this may compromise the results. Please check your input when you receive this message.

A warning message will be displayed whenever no word of one input string is found in the semantic space.

Value

A numeric giving the cosine between the input sentences/documents

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Dennis, S. (2007). How to use the LSA Web Site. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of Latent Semantic Analysis* (pp. 35-56). Mahwah, NJ: Erlbaum.

Mitchell, J., & Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science, 34,* 1388-1429.

http://wordvec.colorado.edu/

See Also

cosine, Cosine, multicos, costring

Examples

```
data(wonderland)

multicostring("alice was beginning to get very tired.",
        "a white rabbit with a clock ran close to her.",
        tvectors=wonderland)

multicostring("suddenly, a cat appeared in the woods",
names(neighbors("cheshire",n=20,tvectors=wonderland)),
tvectors=wonderland)
```

---

multidocs *Comparison of sentence sets*

Description

Computes cosine values between sets of sentences and/or documents

Usage

```
multidocs(x,y=x,chars=10,tvectors=tvectors,remove.punctuation=TRUE,
stopwords = NULL,method ="Add")
```

Arguments

x	a character vector containing different sentences/documents
y	a character vector containing different sentences/documents (y = x by default)
chars	an integer specifying how many letters (starting from the first) of each sentence/document are to be printed in the row.names and col.names of the output matrix
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
remove.punctuation	removes punctuation from x and y; TRUE by default
stopwords	a character vector defining a list of words that are *not* used to compute the document/sentence vector for x and y
method	the compositional model to compute the document vector from its word vectors. The default option method = "Add" computes the document vector as the vector sum. With method = "Multiply", the document vector is computed via element-wise multiplication (see compose).

Details

In the traditional LSA approach, the vector *D* for a document (or a sentence) consisting of the words *(t1, . , tn)* is computed as

$$D = \sum_{i=1}^{n} t_n$$

This is the default method (`method="Add"`) for this function. Alternatively, this function provided the possibility of computing the document vector from its word vectors using element-wise multiplication (see Mitchell & Lapata, 2010 and `compose`).

This function computes the cosines between two sets of documents (or sentences).

The format of `x` (or `y`) should be of the kind `x <- c("this is the first text","here is another text")` (or `y <- c("this is a third text","and here is yet another text")`)

A note will be displayed whenever not all words of one input string are found in the semantic space. ***Caution:*** In that case, the function will still produce a result, by omitting the words not found in the semantic space. Depending on the specific requirements of a task, this may compromise the results. Please check your input when you receive this message.

A warning message will be displayed whenever no word of one input string is found in the semantic space.

Value

A list of three elements:

`cosmat`	A numeric matrix giving the cosines between the input sentences/documents
`xdocs`	A legend for the row.names of `cosmat`
`ydocs`	A legend for the col.names of `cosmat`

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Dennis, S. (2007). How to use the LSA Web Site. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of Latent Semantic Analysis* (pp. 35-56). Mahwah, NJ: Erlbaum.

Mitchell, J., & Lapata, M. (2010). Composition in Distributional Models of Semantics. *Cognitive Science, 34,* 1388-1429.

http://wordvec.colorado.edu/

See Also

`cosine`, `Cosine`, `multicos`, `costring`

Examples

```
data(wonderland)
multidocs(x = c("alice was beginning to get very tired.",
                "the red queen greeted alice."),
          y = c("the mad hatter and the mare hare are having a party.",
                "the hatter sliced the cup of tea in half."),
      tvectors=wonderland)
```

`MultipleChoice` *Answers Multiple Choice Questions*

Description

Selects the nearest word to an input out of a set of options

Usage

```
MultipleChoice(x,y,tvectors=tvectors,remove.punctuation=TRUE, stopwords = NULL,
   method ="Add", all.results=FALSE)
```

Arguments

`x`	a character vector of `length(x) = 1` specifying a sentence/ document (or also a single word)
`y`	a character vector specifying multiple answer options (with each element of the vector being one answer option)
`tvectors`	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
`remove.punctuation`	removes punctuation from `x` and `y`; `TRUE` by default
`stopwords`	a character vector defining a list of words that are *not* used to compute the document/sentence vector for `x` and `y`
`method`	the compositional model to compute the document vector from its word vectors. The default option `method = "Add"` computes the document vector as the vector sum. With `method = "Multiply"`, the document vector is computed via element-wise multiplication (see `compose` and `costring`). With `method = "Analogy"`, the document vector is computed via vector subtraction; see *Description* for more information.
`all.results`	If `all.results=FALSE` (default), the function will only return the best answer as a character string. If `all.results=TRUE`, it will return a named numeric vector, where the names are the different answer options in `y` and the numeric values their respective cosine similarity to `x`, sorted by decreasing similarity.

Details

Computes all the cosines between a given sentence/document or word and multiple answer options. Then selects the nearest option to the input (the option with the highest cosine). This function relies entirely on the `costring` function.

A note will be displayed whenever not all words of one answer alternative are found in the semantic space. ***Caution:*** In that case, the function will still produce a result, by omitting the words not found in the semantic space. Depending on the specific requirements of a task, this may compromise the results. Please check your input when you receive this message.

A warning message will be displayed whenever no word of one answer alternative is found in the semantic space.

Using `method="Analogy"` requires the input in *both* `x` and `y` to only consist of word pairs (for example `x = c("helmet head")` and `y = c("kneecap knee", "atmosphere earth", "grass field")`). In that case, the function will try to identify the best-fitting answer in `y` by applying the `king - man + woman = queen` rationale to solve *man : king = woman : ?* (Mikolov et al., 2013): In that case, one should also have `king - man = queen - woman`. With `method="Analogy"`, the function will compute the difference between the normalized vectors `head - helmet`, and search the nearest of the vector differences `knee - kneecap`, `earth - atmosphere`, and `field - grass`.

Value

If `all.results=FALSE` (default), the function will only return the best answer as a character string. If `all.results=TRUE`, it will return a named numeric vector, where the names are the different answer options in `y` and the numeric values their respective cosine similarity to `x`, sorted by decreasing similarity.

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Mikolov, T., Yih, W. T., & Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT-2013).* Association for Computational Linguistics.

See Also

`cosine`, `Cosine`, `costring`, `multicostring`, `analogy`

Examples

```
data(wonderland)

LSAfun:::MultipleChoice("who does the march hare celebrate his unbirthday with?",
```

```
c("mad hatter","red queen","caterpillar","cheshire Cat"),
tvectors=wonderland)
```

`neighbors` *Find nearest neighbors*

Description

Returns the n nearest words to a given word or sentence/document

Usage

```
neighbors(x,n,tvectors=tvectors)
```

Arguments

`x`	a character vector of `length(x) = 1` or a numeric of `length=ncol(tvectors)` vector with same dimensionality as the semantic space
`n`	the number of neighbors to be computed
`tvectors`	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

The format of `x` should be of the kind `x <- "word1 word2 word3"` instead of

`x <- c("word1", "word2", "word3")` if sentences/documents are used as input. This allows for simple copy&paste-inserting of text.

To import a document *Document.txt* to from a directory for comparisons, set your working directory to this directory using `setwd()`. Then use the following command lines:

`fileName1 <- "Alice_in_Wonderland.txt"`

`x <- readChar(fileName1, file.info(fileName1)$size)`.

Since `x` can also be chosen to be any vector of the active LSA Space, this function can be combined with `compose()` to compute neighbors of complex expressions (see examples)

Value

A named numeric vector. The neighbors are given as names of the vector, and their respective cosines to the input as vector entries.

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Dennis, S. (2007). How to use the LSA Web Site. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of Latent Semantic Analysis* (pp. 35-56). Mahwah, NJ: Erlbaum.

http://wordvec.colorado.edu/

See Also

cosine, plot_neighbors, compose

Examples

```
data(wonderland)

neighbors("cheshire",n=20,tvectors=wonderland)

neighbors(compose("mad","hatter",method="Add",tvectors=wonderland),
n=20,tvectors=wonderland)
```

---

normalize *Normalize a vector*

Description

Normalizes a character vector to a unit vector

Usage

```
normalize(x)
```

Arguments

x	a numeric or integer vector

Details

The (euclidean) norm of a vector x is defined as

$$||x|| = \sqrt{\Sigma(x^2)}$$

To normalize a vector to a unit vector u with $||u|| = 1$, the following equation is applied:

$$x' = x/||x||$$

Value

The normalized vector as a numeric

Author(s)

Fritz Guenther

Examples

```
normalize(1:2)


## check vector norms:

x <- 1:2

sqrt(sum(x^2))              ## vector norm
sqrt(sum(normalize(x)^2))   ## norm = 1
```

---

oldbooks *A collection of five classic books*

Description

This object is a list containing five classical books:

- *Around the World in Eighty Days* by Jules Verne
- *The Three Musketeers* by Alexandre Dumas
- *Frankenstein* by Mary Shelley
- *Dracula* by Bram Stoker
- *The Strange Case of Dr Jekyll and Mr Hyde* by Robert Stevenson

as single-element character vectors. All five books were taken from the Project Gutenberg homepage and contain formatting symbols, such as \n for breaks.

Usage

```
data(oldbooks)
```

Format

A named list containing five character vectors as elements

Source

Project Gutenberg

References

Dumas, A. (1844). *The Three Musketeers*. Retrieved from http://www.gutenberg.org/ebooks/1257

Shelley, M. W. (1818). *Frankenstein; Or, The Modern Prometheus*. Retrieved from http://www.gutenberg.org/ebooks/84

Stevenson, R. L. (1886). *The Strange Case of Dr. Jekyll and Mr. Hyde*. Retrieved from http://www.gutenberg.org/ebooks/42

Stoker, B. (1897). *Dracula*. Retrieved from http://www.gutenberg.org/ebooks/345

Verne, J.(1873). *Around the World in Eighty Days*. Retrieved from http://www.gutenberg.org/ebooks/103

`pairwise` *Pairwise cosine computation*

Description

Computes pairwise cosine similarities

Usage

```
pairwise(x,y,tvectors=tvectors)
```

Arguments

`x`	a character vector
`y`	a character vector
`tvectors`	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

Computes pairwise cosine similarities for two vectors of words. These vectors need to have the same length.

Value

A vector of the same length as `x` and `y` containing the pairwise cosine similarities. Returns `NA` if at least one word in a pair is not found in the semantic space.

Author(s)

Fritz Guenther

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Dennis, S. (2007). How to use the LSA Web Site. In T. K. Landauer, D. S. McNamara, S. Dennis, & W. Kintsch (Eds.), *Handbook of Latent Semantic Analysis* (pp. 35-56). Mahwah, NJ: Erlbaum.

http://wordvec.colorado.edu/

See Also

cosine, Cosine, multicos,

Examples

```
data(wonderland)
pairwise("mouse rabbit cat","king queen hearts",
          tvectors=wonderland)
```

---

plausibility *Compute word (or compound) plausibility*

Description

Gives measures of semantic transparency (plausibility) for words or compounds

Usage

```
plausibility(x,method, n=10,stem,tvectors=tvectors)
```

Arguments

x	a character vector of length(x) = 1 or a numeric of length=ncol(tvectors) vector with same dimensionality as LSA space
method	the measure of semantic transparency, can be one of n_density,length, proximity, or entropy (see *Details*)
n	the number of neighbors for the n_density method
stem	the stem (or word) of comparison for the proximity method
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

The format of x should be of the kind x <- "word1 word2 word3" instead of x <- c("word1", "word2", "word3") if phrases of more than one word are used as input. Simple vector addition of the constituent vectors is then used to compute the phrase vector.

Since x can also be chosen to be any vector of the active LSA Space, this function can be combined with compose() to compute semantic transparency measures of complex expressions (see examples). Since semantic transparency methods were developed as measures for composed vectors, applying them makes most sense for those.

The methods are defined as follows:

- method = "n_density" The average cosine between a (word or phrase) vector and its *n* nearest neighbors, excluding the word itself when a single word is submitted (see also SND for a more detailed version)

- `method = "length"` The length of a vector (as computed by the standard Euclidean norm)
- `method = "proximity"` The cosine similarity between a compound vector and its stem word (for example between *mad hatter* and *hatter* or between *objectify* and *object*)
- `method = "entropy"` The entropy of the *K*-dimensional vector with the vector components $t_1, ..., t_K$, as computed by

$$entropy = \log K - \sum t_i * \log t_i$$

Value

The semantic transparency as a numeric

Author(s)

Fritz Guenther

References

Lazaridou, A., Vecchi, E., & Baroni, M. (2013). Fish transporters and miracle homes: How compositional distributional semantics can help NP parsing. In *Proceedings of EMNLP 2013* (pp. 1908 - 1913). Seattle, WA.

Marelli, M., & Baroni, M. (2015). Affixation in semantic space: Modeling morpheme meanings with compositional distributional semantics. *Psychological Review, 122,*. 485-515.

Vecchi, E. M., Baroni, M., & Zamparelli, R. (2011). (Linear) maps of the impossible: Capturing semantic anomalies in distributional space. In *Proceedings of the ACL Workshop on Distributional Semantics and Compositionality* (pp. 1-9). Portland, OR.

See Also

`Cosine`, `neighbors`, `compose`, `SND`

Examples

```
data(wonderland)

plausibility("cheshire cat",method="n_density",n=10,tvectors=wonderland)

plausibility(compose("mad","hatter",method="Multiply",tvectors=wonderland),
method="proximity",stem="hatter",tvectors=wonderland)
```

plot_doclist *2D- or 3D-Plot of a list of sentences/documents*

Description

2D or 3D-Plot of mutual word similarities to a given list of sentences/documents

Usage

```
plot_doclist(x,connect.lines="all",method="PCA",dims=3,
   axes=F,box=F,cex=1,chars=10,legend=T, size = c(800,800),
   alpha="graded",alpha.grade=1,col="rainbow",
   tvectors=tvectors,remove.punctuation=TRUE,...)
```

Arguments

x	a character vector of length(x) > 1 that contains multiple sentences/documents
dims	the dimensionality of the plot; set either dims = 2 or dims = 3
method	the method to be applied; either a Principal Component Analysis (method="PCA") or a Multidimensional Scaling (method="MDS")
connect.lines	(3d plot only) the number of closest associate words each word is connected with via line. Setting connect.lines="all" (default) will draw all connecting lines and will automatically apply alpha="graded"
axes	(3d plot only) whether axes shall be included in the plot
box	(3d plot only) whether a box shall be drawn around the plot
cex	(2d Plot only) A numerical value giving the amount by which plotting text should be magnified relative to the default.
chars	an integer specifying how many letters (starting from the first) of each sentence/document are to be printed in the plot
legend	(3d plot only) whether a legend shall be drawn illustrating the color scheme of the connect.lines. The legend is inserted as a background bitmap to the plot using bgplot3d. Therefore, they do not resize very gracefully (see the bgplot3d documentation for more information).
size	(3d plot only) A numeric vector with two elements, the first specifying the width and the second specifying the height of the plot device.
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
remove.punctuation	removes punctuation from x and y; TRUE by default
alpha	(3d plot only) A numeric vector specifying the luminance of the connect.lines. By setting alpha="graded", the luminance of every line will be adjusted to the cosine between the two words it connects.

alpha.grade	(3d plot only) Only relevant if alpha="graded". Specify a numeric value for alpha.grade to scale the luminance of all connect.lines up (alpha.grade > 1) or down (alpha.grade < 1) by that factor.
col	(3d plot only) A vector specifying the color of the connect.lines. With setting col ="rainbow" (default), the color of every line will be adjusted to the cosine between the two words it connects, according to the rainbow palette. Other available color palettes for this purpose are heat.colors, terrain.colors, topo.colors, and cm.colors (see rainbow). Additionally, you can customize any color scale of your choice by providing an input specifying more than one color (for example col = c("black","blue","red")).
...	additional arguments which will be passed to plot3d (in a three-dimensional plot only)

Details

Computes all pairwise similarities within a given list of sentences/documents. On this similarity matrix, a Principal Component Analysis (PCA) or a Multidimensional Sclaing (MDS) is applied to get a two- or three-dimensional solution that best captures the similarity structure. This solution is then plotted.

In the traditional LSA approach, the vector *D* for a document (or a sentence) consisting of the words *(t1, . , tn)* is computed as

$$D = \sum_{i=1}^{n} t_n$$

This function then computes the the cosines between two sets of documents (or sentences).

The format of x should be of the kind x <- c("this is the first text","here is another text")

For creating pretty plots showing the similarity structure within this list of words best, set connect.lines="all" and col="rainbow"

Value

see plot3d: this function is called for the side effect of drawing the plot; a vector of object IDs is returned.

plot_doclist further prints a list with two elements:

coordinates	the coordinate vectors of the sentences/documents in the plot as a data frame
xdocs	A legend for the sentence/document labels in the plot and in the coordinates

Author(s)

Fritz Guenther, Taylor Fedechko

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Mardia, K.V., Kent, J.T., & Bibby, J.M. (1979). *Multivariate Analysis*, London: Academic Press.

See Also

cosine, multidocs, plot_neighbors, plot_wordlist, plot3d, princomp, rainbow

Examples

```
data(wonderland)

## Standard Plot

docs <- c("alice was beginning to get very tired.",
          "the red queen greeted alice.",
          "the mad hatter and the mare hare are having a party.",
          "the hatter sliced the cup of tea in half.")

plot_doclist(docs,tvectors=wonderland,method="MDS",dims=2)
```

---

plot_neighbors *2D- or 3D-Plot of neighbors*

Description

2D- or 3D-Approximation of the neighborhood of a given word/sentence

Usage

```
plot_neighbors(x,n,connect.lines="all",start.lines=T,
   method="PCA",dims=3,axes=F,box=F,cex=1,legend=T, size = c(800,800),
   alpha="graded",alpha.grade = 1, col="rainbow",tvectors=tvectors,...)
```

Arguments

x	a character vector of length(x) = 1 or a numeric of length=ncol(tvectors) vector with same dimensionality as LSA space
n	the number of neighbors to be computed
dims	the dimensionality of the plot; set either dims = 2 or dims = 3
method	the method to be applied; either a Principal Component Analysis (method="PCA") or a Multidimensional Scaling (method="MDS")
connect.lines	(3d plot only) the number of closest associate words each word is connected with via line. Setting connect.lines="all" (default) will draw all connecting lines and will automatically apply alpha="graded"; it will furthermore override the start.lines argument
start.lines	(3d plot only) whether lines shall be drawn between x and all the neighbors
axes	(3d plot only) whether axes shall be included in the plot
box	(3d plot only) whether a box shall be drawn around the plot

cex	(2d Plot only) A numerical value giving the amount by which plotting text should be magnified relative to the default.
legend	(3d plot only) whether a legend shall be drawn illustrating the color scheme of the connect.lines. The legend is inserted as a background bitmap to the plot using bgplot3d. Therefore, they do not resize very gracefully (see the bgplot3d documentation for more information).
size	(3d plot only) A numeric vector with two elements, the first specifying the width and the second specifying the height of the plot device.
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
alpha	(3d plot only) a vector of one or two numerics between 0 and 1 specifying the luminance of start.lines (first entry) and connect.lines (second entry). Specifying only one numeric will pass this value to both kinds of lines. With setting alpha="graded", the luminance of every line will be adjusted to the cosine between the two words it connects.
alpha.grade	(3d plot only) Only relevant if alpha="graded". Specify a numeric value for alpha.grade to scale the luminance of all start.lines and connect.lines up (alpha.grade > 1) or down (alpha.grade < 1) by that factor.
col	(3d plot only) a vector of one or two characters specifying the color of start.lines (first entry) and connect.lines (second entry). Specifying only one colour will pass this colour to both kinds of lines. With setting col ="rainbow" (default), the colour of every line will be adjusted to the cosine between the two words it connects, according to the rainbow palette. Other available color palettes for this purpose are heat.colors, terrain.colors, topo.colors, and cm.colors (see rainbow). Additionally, you can customize any color scale of your choice by providing an input specifying more than two colors (for example col = c("black","blue","red")).
...	additional arguments which will be passed to plot3d (in a three-dimensional plot only)

Details

Attempts to create an image of the semantic neighborhood (based on cosine similarity) to a given word, sentence/ document, or vector. An attempt is made to depict this subpart of the LSA space in a two- or three-dimensional plot.

To achieve this, either a Principal Component Analysis (PCA) or a Multidimensional Scaling (MDS) is computed to preserve the interconnections between all the words in this neighborhod as good as possible. Therefore, it is important to note that the image created from this function is only the best two- or three-dimensional approximation to the true LSA space subpart.

For creating pretty plots showing the similarity structure within this neighborhood best, set connect.lines="all" and col="rainbow"

Value

For three-dimensional plots:see plot3d: this function is called for the side effect of drawing the plot; a vector of object IDs is returned

plot_neighbors also gives the coordinate vectors of the words in the plot as a data frame

Author(s)

Fritz Guenther, Taylor Fedechko

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Mardia, K.V., Kent, J.T., & Bibby, J.M. (1979). *Multivariate Analysis*, London: Academic Press.

See Also

cosine, neighbors, multicos, plot_wordlist, plot3d, princomp

Examples

```
data(wonderland)

## Standard Plot
plot_neighbors("cheshire",n=20,tvectors=wonderland)

## Pretty Plot
plot_neighbors("cheshire",n=20,tvectors=wonderland,
                connect.lines="all",col="rainbow")



plot_neighbors(compose("mad","hatter",tvectors=wonderland),
                n=20, connect.lines=2,tvectors=wonderland)
```

---

plot_wordlist *2D- or 3D-Plot of a list of words*

Description

2D or 3D-Plot of mutual word similarities to a given list of words

Usage

```
plot_wordlist(x,connect.lines="all",method="PCA",dims=3,
   axes=F,box=F,cex=1,legend=T, size = c(800,800),
   alpha="graded",alpha.grade=1,col="rainbow",
   tvectors=tvectors,...)
```

Arguments

x	a character vector of length(x) > 1 that contains multiple sentences/documents
dims	the dimensionality of the plot; set either dims = 2 or dims = 3
method	the method to be applied; either a Principal Component Analysis (method="PCA") or a Multidimensional Scaling (method="MDS")
connect.lines	(3d plot only) the number of closest associate words each word is connected with via line. Setting connect.lines="all" (default) will draw all connecting lines and will automatically apply alpha="graded".
axes	(3d plot only) whether axes shall be included in the plot
box	(3d plot only) whether a box shall be drawn around the plot
cex	(2d Plot only) A numerical value giving the amount by which plotting text should be magnified relative to the default.
legend	(3d plot only) whether a legend shall be drawn illustrating the color scheme of the connect.lines. The legend is inserted as a background bitmap to the plot using bgplot3d. Therefore, they do not resize very gracefully (see the bgplot3d documentation for more information).
size	(3d plot only) A numeric vector with two elements, the first specifying the width and the second specifying the height of the plot device.
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
alpha	(3d plot only) A numeric vector specifying the luminance of the connect.lines. By setting alpha="graded", the luminance of every line will be adjusted to the cosine between the two words it connects.
alpha.grade	(3d plot only) Only relevant if alpha="graded". Specify a numeric value for alpha.grade to scale the luminance of all connect.lines up (alpha.grade > 1) or down (alpha.grade < 1) by that factor.
col	(3d plot only) A vector specifying the color of the connect.lines. With setting col ="rainbow" (default), the color of every line will be adjusted to the cosine between the two words it connects, according to the rainbow palette. Other available color palettes for this purpose are heat.colors, terrain.colors, topo.colors, and cm.colors (see rainbow). Additionally, you can customize any color scale of your choice by providing an input specifying more than one color (for example col = c("black","blue","red")).
...	additional arguments which will be passed to plot3d (in a three-dimensional plot only)

Details

Computes all pairwise similarities within a given list of words. On this similarity matrix, a Principal Component Analysis (PCA) or a Multidimensional Sclaing (MDS) is applied to get a two- or three-dimensional solution that best captures the similarity structure. This solution is then plotted.

For creating pretty plots showing the similarity structure within this list of words best, set connect.lines="all" and col="rainbow"

Value

see plot3d: this function is called for the side effect of drawing the plot; a vector of object IDs is returned.

plot_wordlist also gives the coordinate vectors of the words in the plot as a data frame

Author(s)

Fritz Guenther, Taylor Fedechko

References

Landauer, T.K., & Dumais, S.T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. *Psychological Review, 104,* 211-240.

Mardia, K.V., Kent, J.T., & Bibby, J.M. (1979). *Multivariate Analysis*, London: Academic Press.

See Also

cosine, neighbors, multicos, plot_neighbors, plot3d, princomp, rainbow

Examples

```
data(wonderland)

## Standard Plot

words <- c("alice","hatter","queen","knight","hare","cheshire")

plot_wordlist(words,tvectors=wonderland,method="MDS",dims=2)
```

Predication *Compute Vector for Predicate-Argument-Expressions*

Description

Computes vectors for complex expressions of type PREDICATE[ARGUMENT] by applying the method of Kintsch (2001) (see *Details*).

Usage

```
Predication(P,A,m,k,tvectors=tvectors,norm="none")
```

Arguments

P	Predicate of the expression, a single word (character vector)
A	Argument of the expression, a single word (character vector)
m	number of nearest words to the Predicate that are initially activated
k	size of the k-neighborhood; $k \leq m$
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
norm	whether to normalize the single word vectors before applying a composition function. Setting norm = "none" will not perform any normalizations, setting norm = "all" will normalize every involved word vector (Predicate, Argument, and every single activated neighbor). Setting norm = "block" will normalize the Argument vector and will normalize the [Predicate + neighbors] vector, to weight the Argument and the "Predicate in context" equally.

Details

The vector for the expression is computed following the Predication Process by Kintsch (2001):

The m nearest neighbors to the Predicate are computed. Of those, the k nearest neighbors to the Argument are selected. The vector for the expression is then computed as the sum of Predicate vector, Argument vector, and the vectors of those k neighbors (the k-neighborhood).

Value

An object of class Pred: This object is a list consisting of:

$PA	The vector for the complex expression as described above
$P.Pred	The vector for Predicate plus the *k*-neighborhoodvectors without the Argument vector
$neighbors	The words in the *k*-neighborhood.
$P	The Predicate given as input
$A	The Argument given as input

Author(s)

Fritz Guenther

References

Kintsch, W. (2001). Predication. *Cognitive Science, 25,* 173-202.

See Also

cosine, neighbors, multicos, compose

Examples

```
data(wonderland)

Predication(P="mad",A="hatter",m=20,k=3,tvectors=wonderland)
```

`priming` *Simulated data for a Semantic Priming Experiment*

Description

A data frame containing simulated data for a Semantic Priming Experiment. This data contains 514 prime-target pairs, which are taken from the Hutchison, Balota, Cortese and Watson (2008) study. These pairs are generated by pairing each of 257 target words with one semantically related and one semantically unrelated prime.

The data frame contains four columns:

- First column: Prime Words
- Second column: Target Words
- Third column: **Simulated** Reaction Times
- Fourth column: Specifies whether a prime-target pair is considered semantically related or unrelated

Usage

```
data(priming)
```

Format

A data frame with 514 rows and 4 columns

References

Hutchison, K. A., Balota, D. A., Cortese, M. & Watson, J. M. (2008). Predicting semantic priming at the item level. *Quarterly Journal of Experimental Psychology, 61,* 1036-1066.

SND *Semantic neighborhood density*

Description

Returns semantic neighborhood with semantic neighborhood size and density

Usage

```
SND(x,n=NA,threshold=3.5,tvectors=tvectors)
```

Arguments

x	a character vector of `length(x) = 1` or a numeric of `length=ncol(tvectors)` vector with same dimensionality as the semantic space
n	if specified as a numeric, determines the size of the neighborhood as the `n` nearest words to `x`. If `n=NA` (default), the semantic neighborhood will be determined according to a similarity threshold (see `threshold`)
threshold	specifies the similarity threshold that determines if a word is counted as a neighbor for `x`, following the method by Buchanan et al. (2011) (see `Description` below)
tvectors	the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)

Details

There are two principle approaches to determine the semantic neighborhood of a target word:

- Set an a priori size of the semantic neighborhood to a fixed value `n` (e.g., Marelli & Baroni, 2015). The `n` closest words to the target word are counted as its semantic neighbors. The semantic neighborhood size is then necessarily `n`; the semantic neighborhood density is the mean similarity between these neighbors and the target word (see also `plausibility`)
- Determine the semantic neighborhood based on a similarity threshold; all words whose similarity to the target word exceeds this threshold are counted as its semantic neighbors (e.g., Buchanan, Westbury, & Burgess, 2001). First, the similarity between the target word and all words in the semantic space is computed. These similarities are then transformed into *z*-scores. Traditionally, the threshold is set to $z = 3.5$ (e.g., Buchanan, Westbury, & Burgess, 2001).

If a single target word is used as `x`, this target word itself (which always has a similarity of 1 to itself) is excluded from these computations so that it cannot be counted as its own neighbor

Value

A list of three elements:

- neighbors: A names numeric vector of all identified neighbors, with the names being these neighbors and the values their similarity to x
- n_size: The number of neighbors as a numeric
- SND: The semantic neighborhood density (SND) as a numeric

Author(s)

Fritz Guenther

References

Buchanan, L., Westbury, C., & Burgess, C. (2001). Characterizing semantic space: Neighborhood effects in word recognition. *Psychonomic Bulletin & Review, 8,* 531-544.

Marelli, M., & Baroni, M. (2015). Affixation in semantic space: Modeling morpheme meanings with compositional distributional semantics. *Psychological Review, 122,* 485-515.

See Also

cosine, plot_neighbors, compose

Examples

```
data(wonderland)

SND("cheshire",n=20,tvectors=wonderland)

SND("alice",threshold=2,tvectors=wonderland)
```

---

syntest *A multiple choice test for synonyms and antonyms*

Description

This object multiple choice test for synonyms and antonyms, consisting of seven columns.

1. The first column defines the question, i.e. the word a synonym or an antonym has to be found for.
2. The second up to the fifth column show the possible answer alternatives.
3. The sixth column defines the correct answer.
4. The seventh column indicates whether a synonym or an antonym has to be found for the word in question.

The test consists of twenty questions, which are given in the twenty rows of the data frame.

Usage

```
data(syntest)
```

Format

A data frame with 20 rows and 7 columns

`wonderland` *LSA Space: Alice's Adventures in Wonderland*

Description

This data set is a 50-dimensional LSA space derived from Lewis Carrol's book "Alice's Adventures in Wonderland". The book was split into 791 paragraphs which served as documents for the LSA algorithm (Landauer, Foltz & Laham, 1998). Only words that appeared in at least two documents were used for building the LSA space.

This LSA space contains 1123 different terms, all in lower case letters, and was created using the `lsa`-package. It can be used as `tvectors` for all the functions in the `LSAfun`-package.

Usage

```
data(wonderland)
```

Format

A 1123x50 matrix with terms as rownames.

Source

Alice in Wonderland from Project Gutenberg

References

Landauer, T., Foltz, P., and Laham, D. (1998) *Introduction to Latent Semantic Analysis*. In: Discourse Processes 25, pp. 259-284.

Carroll, L. (1865). *Alice's Adventures in Wonderland*. New York: MacMillan.

Index