Package ‘LSAfun’

December 4, 2019

Type Package
Title Applied Latent Semantic Analysis (LSA) Functions
Description Provides functions that allow for convenient working with latent semantic analysis (LSA) and other vector space models of semantics. For actually building a vector semantic space, use the package ‘lsa’ or other specialized software. Downloadable semantic spaces can be found here: <https://sites.google.com/site/fritzgntr/software-resources>. A description of the LSA algorithm can be found in Landauer and Dumais (1997) <doi:10.1037/0033-295X.104.2.211>.

Version 0.6.1
Date 2019-12-04
Depends R (>= 3.1.0), lsa, rgl
License GPL (>= 2)
LazyData true
NeedsCompilation no
Author Fritz Guenther [aut, cre]
Maintainer Fritz Guenther <fritz.guenther@uni-tuebingen.de>
Repository CRAN
Date/Publication 2019-12-04 20:20:02 UTC

R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSAfun-package</td>
<td>2</td>
</tr>
<tr>
<td>asym</td>
<td>3</td>
</tr>
<tr>
<td>breakdown</td>
<td>5</td>
</tr>
<tr>
<td>choose.target</td>
<td>6</td>
</tr>
<tr>
<td>coherence</td>
<td>7</td>
</tr>
<tr>
<td>compose</td>
<td>9</td>
</tr>
<tr>
<td>conSIM</td>
<td>11</td>
</tr>
<tr>
<td>Cosine</td>
<td>12</td>
</tr>
<tr>
<td>costring</td>
<td>13</td>
</tr>
<tr>
<td>distance</td>
<td>15</td>
</tr>
</tbody>
</table>
Description

Offers methods and functions for working with Vector Space Models of semantics, such as Latent Semantic Analysis (LSA). 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 \texttt{lsa}. The focus of the package \texttt{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
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](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 Günther

---

**asym**

*Asymmetric Similarity functions*

**Description**

Compute various asymmetric similarities between words

**Usage**

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

**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)
- `breakdown` if TRUE, the function `breakdown` is applied to the input
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"**
  \[
  cosweeds(u, v) = \sqrt{weedsprec(u, v) \times \text{cosine}(u, v)}
  \]

- **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, asymmetric 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 Günther
breakdown

References


See Also

Cosine conSIM

Examples

data(wonderland)

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

breakdown

Clean up special characters

Description

Replaces special characters in character vectors

Usage

breakdown(x)

Arguments

x a character vector

Details

Applies the following functions to a character vector

• sets all letters to lower case
• replaces umlauts (for example ä replaced by ae)
• removes accents from letters (for example é replaced by e)
• replaces ß by ss
  Also removes other special characters, like punctuation signs, numbers and breaks

Value

A character vector
Author(s)

Fritz Günther

See Also

gsub

Examples

breakdown("Märchen")

breakdown("I was visiting Orléans last week. It was nice, though!")

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, breakdown = FALSE)

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)
breakdown  if TRUE, the function breakdown is applied to the input

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.
coherence

Author(s)
Fritz Günther

References

See Also
cosine, Cosine, neighbors

Examples
data(wonderland)

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

```
coherence

Coherence of a text
```

Description
Computes coherence of a given paragraph/document

Usage
```
coherence(x,split=c(" ","!","?"),t_vectors=t_vectors,breakdown=FALSE)
```

Arguments
```
x a character vector of length(x) = 1 containing the document
split a vector of expressions that determine where to split sentences
t_vectors the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
breakdown if TRUE, the function breakdown is applied to the input
```
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 (or y) 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:

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

### 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 Günther

### References


### See Also

`cosine`, `Cosine`, `costring`

### Examples

```r
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**

```r
## Default
ccompose(x,y,method="Add", a=1,b=1,c=1,m,k,lambda=2,
tvectors=tvectors,breakdown=FALSE, 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 \( \text{method} = "\text{Dilation}" \)
- `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)
- `breakdown` if TRUE, the function `breakdown` is applied to the input

**Details**

Let \( p \) be the vector with entries \( p_i \) for the two-word phrase consisting 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** (\( \text{method} = "\text{Add}" \))
  \[
  p_i = u_i + v_i
  \]

- **Weighted Additive Model** (\( \text{method} = "\text{WeightAdd}" \))
  \[
  p_i = a \cdot u_i + b \cdot v_i
  \]

- **Multiplicative Model** (\( \text{method} = "\text{Multiply}" \))
  \[
  p_i = u_i \cdot v_i
  \]
- Combined Model (method = "Combined")

\[ p_i = a \ast u_i + b \ast v_i + c \ast u_i \ast v_i \]

- 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")

\[ p_i = \sum_{j} u_j \ast v_{i-j} \]

, where the subscripts of \( v \) are interpreted modulo \( n \) with \( n = \text{length}(x) = \text{length}(y) \)

- Dilation (method = "Dilation")

\[ p = (u \ast u) \ast v + (\lambda - 1) \ast (u \ast v) \ast u \]

, with \((u \ast u)\) being the dot product of \( u \) and \( u \) (and \((u \ast v)\) 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="word1") will give different results than compose(x="word2",y="word1")

**Value**

The phrase vector as a numeric vector

**Author(s)**

Fritz Günther

**References**


**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, breakdown = FALSE)

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)
breakdown if TRUE, the function breakdown is applied to the input

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
Cosine

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 Günther

References


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,breakdown=FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A single word, given as a character of length(x) = 1</td>
</tr>
<tr>
<td>y</td>
<td>A single word, given as a character of length(y) = 1</td>
</tr>
<tr>
<td>tvectors</td>
<td>the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)</td>
</tr>
<tr>
<td>breakdown</td>
<td>if TRUE, the function breakdown is applied to the input</td>
</tr>
</tbody>
</table>
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 Günther

References


http://lsa.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,breakdown=FALSE)
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)
- **breakdown**: if TRUE, the function breakdown is applied to the input

Details

In the traditional LSA approach, the vector $D$ for a document (or a sentence) consisting of the words $(t_1, \ldots, t_n)$ is computed as

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

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

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:

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

Value

A numeric giving the cosine between the input sentences/documents

Author(s)

Fritz Günther

References


http://lsa.colorado.edu/
distance

See Also
cosine, Cosine, multicos, 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, breakdown=FALSE)

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)
breakdown if TRUE, the function breakdown is applied to the input

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 Günther

See Also
Cosine asym
genericSummary  

**Examples**

data(wonderland)

distance("alice","rabbit",method="euclidean",t vectors=wonderland)

---

genericSummary  

Summarize a text

**Description**

Selects sentences from a text that best describe its topic

**Usage**

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

**Arguments**

text   A character vector of length(text) = 1 specifying 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
breakdown  If TRUE, the function breakdown is applied to the input
...     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 $\Sigma$, 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 operation; otherwise, increment $k$ by one, and go to Step 4.

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

Value

A character vector of the length \( k \)

Author(s)

Fritz Günther

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,breakdown=FALSE)
```

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)
- `breakdown` if TRUE, the function `breakdown` is applied to the input

Details

Submit a character vector consisting of \( n \) words to get a \( n \times n \) 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.
multicostring

Value
A matrix containing the pairwise cosines of x and y

Author(s)
Fritz Günther

References


http://lsa.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,breakdown=FALSE)

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)
breakdown if TRUE, the function breakdown is applied to the input
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. For computing the vector for the document/sentence specified in `x`, the simple Addition model is used (see `costring`).

Value

A numeric giving the cosine between the input sentences/documents

Author(s)

Fritz Günther

References


http://lsa.colorado.edu/

See Also

cosine, Cosine, multicos, multicostring

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,breakdown=FALSE)
```

**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)
- **breakdown**: if *TRUE*, the function *breakdown* is applied to the input

**Details**

In the traditional LSA approach, the vector $D$ for a document (or a sentence) consisting of the words $(t_1, \ldots, t_n)$ is computed as

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

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")

**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 Günther
MultipleChoice

References


http://lsa.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."),
  t_vectors=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,breakdown=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
tvectors the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
breakdown if TRUE, the function breakdown is applied to the input
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 warning message will be displayed if all words of one answer alternative are not found in the semantic space.

Value

The nearest option to x as a character

Author(s)

Fritz Günther

References


See Also

cosine, Cosine, costring

Examples

data(wonderland)
LSAfun:::MultipleChoice("Who does the march hare celebrate his unbirthday with?", c("Mad Hatter","Red Queen","Caterpillar","Cheshire Cat"), tvectors=wonderland,breakdown=TRUE)

default(neighbors)

Find nearest neighbors

Description

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

Usage

neighbors(x,n,tvectors=tvectors,breakdown=FALSE)
neighbors

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
- **tvectors**: the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
- **breakdown**: if TRUE, the function `breakdown` is applied to the input

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:

```r
fileName1 <- "Alice_in_Wonderland.txt"
```

```r
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 Günther

References


http://lsa.colorado.edu/

See Also

cosine, plot_neighbors, compose
normalize

Examples

```r
data(wonderland)
neighbors("cheshire", n=20, t_vectors=wonderland)

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

normalize

Normalize a vector

Description

Normalizes a character vector to a unit vector

Usage

```r
normalize(x)
```

Arguments

- `x` a numeric or integer vector

Details

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

\[
||x|| = \sqrt{\sum(x^2)}
\]

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 Günther
Examples

normalize(1:2)

## check vector norms:
x <- 1:2

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

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 home-
page 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


---

pairwise  
*Pairwise cosine computation*

**Description**

Computes pairwise cosine similarities

**Usage**

```r
pairwise(x, y, tvectors = tvectors, breakdown = FALSE)
```

**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)
- `breakdown`: if TRUE, the function `breakdown` is applied to the input

**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 Günther
References


http://lsa.colorado.edu/

See Also
cosine, Cosine, multicos,

Examples

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

---

**plausibility**  
*Compute word (or compound) plausibility*

**Description**

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

**Usage**

```r
plausibility(x, method, n=10, stem, tveectors=tveectors, breakdown=FALSE)
```

**Arguments**

- **x**: a character vector of length(x) = 1 or a numeric of length=ncol(tveectors) 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
- **tveectors**: the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
- **breakdown**: if TRUE, the function `breakdown` is applied to the input
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:

- \textbf{method} = "n\_density" The average cosine between a (word or phrase) vector and its n nearest neighbors (see \texttt{neighbors})
- \textbf{method} = "length" The length of a vector (as computed by the standard Euclidean norm)
- \textbf{method} = "proximity" The cosine similarity between a compound vector and its stem word (for example between \textit{mad hatter} and \textit{hatter} or between \textit{objectify} and \textit{object})
- \textbf{method} = "entropy" The entropy of the K-dimensional vector with the vector components \( t_1, \ldots, t_K \), as computed by

\[
entropy = \log K - \sum t_i \log t_i
\]

Value

The semantic transparency as a numeric

Author(s)

Fritz Günther

References


See Also

\texttt{Cosine, neighbors, compose}
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,breakdown=FALSE,...)

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 sen-
tence/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.
t_vectors the semantic space in which the computation is to be done (a numeric matrix where every row is a word vector)
breakdown if TRUE, the function breakdown is applied to the input
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 Scaling (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 \((t_1, \ldots, t_n)\) is computed as

\[
D = \sum_{i=1}^{n} t_i
\]

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

The format of \( x \) should be of the kind \( x \leftarrow c("this is the first text", "here is another text") \)
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 Günther, Taylor Fedechko
References


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

Examples
data(wonderland)

```r
## 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**

```r
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,breakdown=FALSE,...)
```

**Arguments**

- `x`: a character vector of length(x) = 1 or a numeric of length=ncol(tvectors)
- `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)

breakdown  if TRUE, the function breakdown is applied to the input

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
plot_neighbors

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 Günther, Taylor Fedechko

References


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",
   t vectors=t vectors, breakdown=FALSE,...)

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.
 t vectors the semantic space in which the computation is to be done (a numeric matrix
   where every row is a word vector)
 breakdown if TRUE, the function breakdown is applied to the input
 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 Günther, Taylor Fedechko

References


See Also

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

Examples

data(wonderland)

## Standard Plot

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

*Compute Vector for Predicate-Argument-Expressions*

**Description**


**Usage**

```
Predication(P, A, m, k, tvectors = tvectors, breakdown = FALSE, 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)
- **breakdown**: if TRUE, the function `breakdown` is applied to the input
- **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-neighborhood vectors without the Argument vector
The words in the \( k \)-neighborhood.

The Predicate given as input

The Argument given as input

**Author(s)**

Fritz Günther

**References**


**See Also**

cosine, neighbors, multicos, compose

**Examples**

data(wonderland)

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

---

**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

**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
```r
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 tve ctors for all the functions in the LSAfun-package.

Usage
```r
data(wonderland)
```
Format

A 1123x50 matrix with terms as rownames.

Source

Alice in Wonderland from Project Gutenberg

References


Index

*Topic Books
  oldbooks, 25
*Topic LSA space
  wonderland, 38
*Topic Synonym Test
  priming, 37
  syntest, 38

asym, 3, 12, 13, 15
bgplot3d, 29, 32, 34
breakdown, 3, 5, 6, 7, 9, 11, 12, 14–18, 20, 21, 23, 26, 27, 30, 32, 34, 36
choose.target, 6
cohere, 7
compose, 9, 23, 28, 37
cosSIM, 5, 11
Cosine, 5, 7, 8, 12, 12, 15, 18, 19, 21, 22, 27, 28
cosine, 7, 8, 15, 18, 19, 21–23, 27, 31, 33, 35, 37
costring, 8, 13, 18, 19, 21, 22
distance, 13, 15
genericSummary, 16
gsub, 6
lsa, 2, 3, 17, 38
LSAfun-package, 2

multicos, 15, 17, 19, 21, 27, 33, 35, 37
multicostring, 15, 18, 18, 19
multidocs, 20, 31
MultipleChoice, 21

neighbors, 7, 19, 22, 28, 33, 35, 37
normalize, 9, 24, 36

oldbooks, 25

pairwise, 26
plausibility, 27
plot3d, 30–33, 35
plot_doclist, 29
plot_neighbors, 23, 31, 31, 35
plot_wordlist, 31, 33, 34
Predication, 9, 10, 36
priming, 37
princomp, 31, 33, 35
rainbow, 30–32, 35
svd, 17
syntest, 38
textmatrix, 16, 17

wonderland, 3, 38

40