

# The lsa Package

November 25, 2005

**Title** Latent Semantic Analysis

**Version** 0.57

**Date** 2005-11-23

**Author** Fridolin Wild

**Description** The basic idea of latent semantic analysis (LSA) is, that text do have a higher order (=latent semantic) structure which, however, is obscured by word usage (e.g. through the use of synonyms or polysemy). By using conceptual indices that are derived statistically via a truncated singular value decomposition (a two-mode factor analysis) over a given document-term matrix, this variability problem can be overcome.

**Depends** Rstem

**Maintainer** Fridolin Wild <fridolin.wild@wu-wien.ac.at>

**License** GPL version 2 or later.

## R topics documented:

as.textmatrix . . . . .	2
cosine . . . . .	3
dimcalc . . . . .	4
fold_in . . . . .	6
lsa . . . . .	7
print.textmatrix . . . . .	9
query . . . . .	10
stopwords . . . . .	11
summary.textmatrix . . . . .	12
textmatrix . . . . .	13
triples . . . . .	14
weightings . . . . .	16

<b>Index</b>	<b>18</b>
--------------	-----------

---

as.textmatrix	<i>Display a latent semantic space generated by Latent Semantic Analysis (LSA)</i>
---------------	--

---

### Description

Returns a latent semantic space (created by createLSAspace) in textmatrix format: rows are terms, columns are documents.

### Usage

```
as.textmatrix( LSAspace )
```

### Arguments

LSAspace      a latent semantic space generated by createLSAspace.

### Details

To allow comparisons between terms and documents, the internal format of the latent semantic space needs to be converted to a classical document-term matrix (just like the ones generated by `textmatrix()` that are of class 'textmatrix').

Remark: There are other ways to compare documents and terms using the partial matrices from an LSA space directly. See (Berry, 1995) for more information.

### Value

textmatrix      a textmatrix representation of the latent semantic space.

### Author(s)

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

### References

Berry, M., Dumais, S., and O'Brien, G (1995) *Using Linear Algebra for Intelligent Information Retrieval*. In: SIAM Review, Vol. 37(4), pp.573–595.

### See Also

[textmatrix](#), [lsa](#), [fold\\_in](#)

### Examples

```
# create some files
td = tempfile()
dir.create(td)
write( c("dog", "cat", "mouse"), file=paste(td, "D1", sep="/"))
write( c("hamster", "mouse", "sushi"), file=paste(td, "D2", sep="/"))
write( c("dog", "monster", "monster"), file=paste(td, "D3", sep="/"))
write( c("dog", "mouse", "dog"), file=paste(td, "D4", sep="/"))
```

```
# read files into a document-term matrix
myMatrix = textmatrix(td, minWordLength=1)

# create the latent semantic space
myLSAspace = lsa(myMatrix, dims=dimcalc_raw())

# display it as a textmatrix again
round(as.textmatrix(myLSAspace),2) # should give the original

# create the latent semantic space
myLSAspace = lsa(myMatrix, dims=dimcalc_share())

# display it as a textmatrix again
myNewMatrix = as.textmatrix(myLSAspace)
myNewMatrix # should look be different!

# compare two terms with the cosine measure
cosine(myNewMatrix["dog",], myNewMatrix["cat",])

# compare two documents with pearson
cor(myNewMatrix[,1], myNewMatrix[,2], method="pearson")

# clean up
unlink(td, recursive=TRUE)
```

---

cosine

*Cosine Measure (Matrices)*

---

## Description

Calculates the cosine measure between two vectors or between all column vectors of a matrix.

## Usage

```
cosine(x, y = NULL)
```

## Arguments

x	A vector or a matrix (e.g., a document-term matrix).
y	Optional: a vector with compatible dimensions to x. If 'NULL', all column vectors of x are correlated.

## Details

`cosine()` calculates a similarity matrix between all column vectors of a matrix x. This matrix might be a document-term matrix, so columns would be expected to be documents and rows to be terms.

When executed on two vectors x and y, `cosine()` calculates the cosine similarity between them.

## Value

Returns a  $n * n$  similarity matrix of cosine values, comparing all  $n$  column vectors against each other. Executed on two vectors, their cosine similarity value is returned.

**Note**

The cosine measure is nearly identical with the pearson correlation coefficient (besides a constant factor) `cor(method="pearson")`. For an investigation on the differences in the context of textmining see (Leydesdorff, 2005).

**Author(s)**

Fridolin Wild (fridolin.wild@wu-wien.ac.at)

**References**

Leydesdorff, L. (2005) *Similarity Measures, Author Cocitation Analysis, and Information Theory*. In: JASIST 56(7), pp.769-772.

**See Also**

[cor](#)

**Examples**

```
## the cosinus measure between two vectors

vec1 = c( 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 )
vec2 = c( 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0 )
cosine(vec1,vec2)

## the cosine measure for all document vectors of a matrix

vec3 = c( 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0 )
matrix = cbind(vec1,vec2, vec3)
cosine(matrix)
```

---

dimcalc

*Dimensionality Calculation Routines (LSA)*

---

**Description**

Methods for choosing a ‘good’ number of singular values for the dimensionality reduction in LSA.

**Usage**

```
dimcalc_share(share=0.5)
dimcalc_ndocs(ndocs)
dimcalc_kaiser()
dimcalc_raw()
```

**Arguments**

share	Optional: a fraction of the sum of the selected singular values to the sum of all singular values (default: 0.5). Only needed by <code>dimcalc_share</code> .
ndocs	Optional: the number of documents (only needed for <code>dimcalc_ndocs()</code> ).

## Details

In an LSA process, the diagonal matrix of the singular value decomposition is usually reduced to a specific number of dimensions (also ‘factors’ or ‘singular values’).

The functions `dimcalc_share()`, `dimcalc_ndocs()`, `dimcalc_kaiser()` and also the redundant function `dimcalc_raw()` offer methods to calculate a useful number of singular values (based on the distribution and values of the given sequence of singular values).

All of them are tightly coupled to the core LSA functions: they generates a function to be executed by the calling (higher-level) function `lsa()`. The output function contains only one parameter, namely `s`, which is expected to be the sequence of singular values. In `lsa()`, the code returned is executed, the mandatory singular values are provided as a parameter within `lsa()`.

The dimensionality calculation methods, however, can still be called directly by adding a second, separate parameter set: e.g.

```
dimcalc_share(share=0.2)(mysingularvalues)
```

The method `dimcalc_share()` finds the first position in the descending sequence of singular values `s` where their sum (divided by the sum of all values) meets or exceeds the specified share.

The method `dimcalc_ndocs()` calculates the first position in the descending sequence of singular values where their sum meets or exceeds the number of documents.

The method `dimcalc_kaiser()` calculates the number of singular values according to the Kaiser-Criterium, i.e. from the descending order of values all values with  $s[n] > 1$  will be taken. The number of dimensions is returned accordingly.

The method `dimcalc_raw()` return the maximum number of singular values (= the length of `s`). It is here only for completeness.

## Value

Returns a function that takes the singular values as a parameter to return the recommended number of dimensions. The expected parameter of this function is

`s`                      A sequence of singular values (as produced by the SVD). Only needed when calling the dimensionality calculation routines directly.

## Author(s)

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

## References

Wild, F., Stahl, C., Stermsek, G., Neumann, G., Playa, Y. (2005) *Parameters Driving Effectiveness of Automated Essay Scoring with LSA*. In: Proceedings of the 9th CAA, pp.485-494, Loughborough

## See Also

[lsa](#)

## Examples

```
## create some data
vec1 = c( 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 )
vec2 = c( 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0 )
vec3 = c( 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0 )
```

```

matrix = cbind(vec1,vec2, vec3)
s = svd(matrix)$d

# standard share of 0.5
dimcalc_share()(s)

# specific share of 0.9
dimcalc_share(share=0.9)(s)

# meeting the number of documents (here: 3)
n = ncol(matrix)
dimcalc_ndocs(n)(s)

```

---

fold_in	<i>Ex-post folding-in of textmatrices into an existing latent semantic space</i>
---------	--

---

### Description

Additional documents can be mapped into a pre-existing latent semantic space without influencing the factor distribution of the space. Applied, when additional documents must not influence the calculated existing latent semantic factor structure.

### Usage

```
fold_in( docvecs, LSAspace )
```

### Arguments

LSAspace	a latent semantic space generated by createLSAspace.
docvecs	a textmatrix.

### Details

To keep additional documents from influencing the factor distribution calculated previously from a particular text basis, they can be folded-in after the singular value decomposition performed in `lsa()`.

Background Information: For folding-in, a pseudo document vector  $m_i$  of the new documents is calculated into as shown in the equations (1) and (2) (cf. Berry et al., 1995):

$$(1) \hat{d} = v^T T_k S_k^{-1}$$

$$(2) \hat{m} = T_k S_k \hat{d}$$

The document vector  $v^T$  in equation (1) is identical to an additional column of an input textmatrix  $M$  with the term frequencies of the essay to be folded-in.  $T_k$  and  $S_k$  are the truncated matrices from the SVD applied through `lsa()` on a given text collection to construct the latent semantic space. The resulting vector  $\hat{m}$  from equation (2) is identical to an additional column in the textmatrix representation of the latent semantic space (as produced by `as.textmatrix()`).

### Value

textmatrix	a textmatrix representation of the additional documents in the latent semantic space.
------------	---

**Author(s)**

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

**See Also**

[textmatrix](#), [lsa](#), [as.textmatrix](#)

**Examples**

```
# create a first textmatrix with some files
td = tempfile()
dir.create(td)
write( c("dog", "cat", "mouse"), file=paste(td, "D1", sep="/") )
write( c("hamster", "mouse", "sushi"), file=paste(td, "D2", sep="/") )
write( c("dog", "monster", "monster"), file=paste(td, "D3", sep="/") )
matrix1 = textmatrix(td, minWordLength=1)
unlink(td, recursive=TRUE)

# create a second textmatrix with some more files
td = tempfile()
dir.create(td)
write( c("cat", "mouse", "mouse"), file=paste(td, "A1", sep="/") )
write( c("nothing", "mouse", "monster"), file=paste(td, "A2", sep="/") )
write( c("cat", "monster", "monster"), file=paste(td, "A3", sep="/") )
matrix2 = textmatrix(td, vocabulary=rownames(matrix1), minWordLength=1)
unlink(td, recursive=TRUE)

# create an LSA space from matrix1
spacel = lsa(matrix1, dims=dimcalc_share())
as.textmatrix(spacel)

# fold matrix2 into the space generated by matrix1
fold_in( matrix2, spacel)
```

---

 lsa

---

*Create a vector space with Latent Semantic Analysis (LSA)*


---

**Description**

Calculates a latent semantic space from a given document-term matrix.

**Usage**

```
lsa( x, dims=dimcalc_share() )
```

**Arguments**

<code>x</code>	a document-term matrix (recommended to be of class <code>textmatrix</code> ), containing documents in columns, terms in rows and occurrence frequencies in the cells.
<code>dims</code>	either the number of dimensions or a configuring function.

## Details

LSA combines the classical vector space model — well known in textmining — with a Singular Value Decomposition (SVD), a two-mode factor analysis. Thereby, bag-of-words representations of texts can be mapped into a modified vector space that is assumed to reflect semantic structure.

With `lsa()` a new latent semantic space can be constructed over a given document-term matrix. To ease comparisons of terms and documents with common correlation measures, the space can be converted into a `textmatrix` of the same format as `y` by calling `as.textmatrix()`.

To add more documents or queries to this latent semantic space in order to keep them from influencing the original factor distribution (i.e., the latent semantic structure calculated from a primary text corpus), they can be ‘folded-in’ later on (with the function `fold_in()`).

Background information (see also Deerwester et al., 1990):

A document-term matrix  $M$  is constructed with `textmatrix()` from a given text base of  $n$  documents containing  $m$  terms. This matrix  $M$  of the size  $m \times n$  is then decomposed via a singular value decomposition into: term vector matrix  $T$  (constituting left singular vectors), the document vector matrix  $D$  (constituting right singular vectors) being both orthonormal, and the diagonal matrix  $S$  (constituting singular values).

$$M = TSD^T$$

These matrices are then reduced to the given number of dimensions  $k = \text{dims}$  to result into truncated matrices  $T_k$ ,  $S_k$  and  $D_k$  — the latent semantic space.

$$M_k = \sum_{i=1}^k t_i \cdot s_i \cdot d_i^T$$

If these matrices  $T_k$ ,  $S_k$ ,  $D_k$  were multiplied, they would give a new matrix  $M_k$  (of the same format as  $M$ , i.e., rows are the same terms, columns are the same documents), which is the least-squares best fit approximation of  $M$  with  $k$  singular values.

In the case of folding-in, i.e., multiplying new documents into a given latent semantic space, the matrices  $T_k$  and  $S_k$  remain unchanged and an additional  $D_k$  is created (without replacing the old one). All three are multiplied together to return a (new and appendable) document-term matrix  $\hat{M}$  in the term-order of  $M$ .

## Value

`LSAspace` a list with components  $(T_k, S_k, D_k)$ , representing the latent semantic space.

## Author(s)

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

## References

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and Harshman, R. (1990) *Indexing by Latent Semantic Analysis*. In: *Journal of the American Society for Information Science* 41(6), pp. 391–407.

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

## See Also

[as.textmatrix](#), [fold\\_in](#), [textmatrix](#), [gw\\_idf](#), [dimcalc\\_share](#)

## Examples

```
# create some files
td = tempfile()
dir.create(td)
write( c("dog", "cat", "mouse"), file=paste(td, "D1", sep="/") )
write( c("ham", "mouse", "sushi"), file=paste(td, "D2", sep="/") )
write( c("dog", "pet", "pet"), file=paste(td, "D3", sep="/") )

# LSA
data(stopwords_en)
myMatrix = textmatrix(td, stopwords=stopwords_en)
myMatrix = lw_logtf(myMatrix) * gw_idf(myMatrix)
myLSAspace = lsa(myMatrix, dims=dimcalc_share())
as.textmatrix(myLSAspace)

# clean up
unlink(td, recursive=TRUE)
```

---

print.textmatrix    *Print a textmatrix (Matrices)*

---

## Description

Display a one screen short version of a textmatrix.

## Usage

```
print.textmatrix( x, bag_lines, bag_cols, ... )
```

## Arguments

x	A textmatrix.
bag_lines	The number of lines per bag.
bag_cols	The number of columns per bag.
...	

## Details

Document-term matrices are often very large and cannot be displayed completely on one screen. Therefore, the textmatrix print method displays only clippings ('bags') from this matrix.

Clippings are taken vertically and horizontally from beginning, middle, and end of the matrix. bag\_lines lines and bag\_cols columns are printed to the screen.

To keep document titles from blowing up the display, the legend is printed below, referencing the symbols used in the table.

## Value

**Author(s)**

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

**See Also**

[textmatrix](#)

**Examples**

```
# fake a matrix
m = matrix(ncol=800, nrow=400)
m[1:length(m)] = 1:length(m)
colnames(m) = paste("D", 1:ncol(m), sep=" ")
rownames(m) = paste("W", 1:nrow(m), sep=" ")
class(m) = "textmatrix"

# show a short form of the matrix
print(m, bag_cols=5)
```

---

query

*Query (Matrices)*

---

**Description**

Create a query in the format of a given textmatrix.

**Usage**

```
query ( qtext, termlist, stemming=FALSE, language="german" )
```

**Arguments**

termlist	the termlist of the background latent-semantic space.
language	specifies a language for stemming / stop-word-removal.
stemming	boolean, specifies whether all terms will be reduced to their wordstems.
qtext	the query string, words are separated by blanks.

**Details**

Create queries, i.e., an additional term vector to be used for query-to-document comparisons, in the format of a given textmatrix.

**Value**

query returns the query vector (based on the given vocabulary) as matrix.

**Author(s)**

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

**See Also**

[wordStem](#), [textmatrix](#)

**Examples**

```
# prepare some files
td = tempfile()
dir.create(td)
write( c("dog", "cat", "mouse"), file=paste(td,"D1", sep="/") )
write( c("hamster", "mouse", "sushi"), file=paste(td,"D2", sep="/") )
write( c("dog", "monster", "monster"), file=paste(td,"D3", sep="/") )

# demonstrate generation of a query
dtm = textmatrix(td)
query("monster", rownames(dtm))
query("monster dog", rownames(dtm))

# clean up
unlink(td, TRUE)
```

---

stopwords

*Stopwordlists in German and English*

---

**Description**

This data sets contain very common lists of words that want to be ignored when building up a document-term matrix. The stop word lists can be loaded by calling `data(stopwords_en)` and `data(stopwords_de)`. The objects `stopwords_de` and `stopwords_en` must already exist *before* being handed over to `textmatrix()`.

**Usage**

```
data(stopwords_de)
data(stopwords_en)
```

**Format**

A vector containing 424 (370) English (German) stop words (e.g. 'he', 'she', 'a').

**Author(s)**

Fridolin Wild (fridolin.wild@wu-wien.ac.at)

summary.textmatrix *Summary of a textmatrix (Matrices)*

---

### Description

Return a summary with some statistical infos about a given textmatrix.

### Usage

```
summary.textmatrix( object, ... )
```

### Arguments

object            A textmatrix.  
...

### Details

Returns some statistical infos about the textmatrix `x`: number of terms, number of documents, maximum length of a term, number of values not 0, number of terms containing strange characters.

### Value

matrix            Returns a matrix.

### Author(s)

Fridolin Wild (fridolin.wild@wu-wien.ac.at)

### See Also

[textmatrix](#)

### Examples

```
# fake a matrix
m = matrix(ncol=800, nrow=400)
m[1:length(m)] = 1:length(m)
colnames(m) = paste("D",1:ncol(m),sep="")
rownames(m) = paste("W",1:nrow(m),sep="")
class(m) = "textmatrix"

# show a short form of the matrix
summary(m)
```

---

textmatrix	<i>Textmatrix (Matrices)</i>
------------	------------------------------

---

### Description

Creates a document-term matrix from all textfiles in a given directory.

### Usage

```
textmatrix( mydir, stemming=FALSE, language="german",
            minWordLength=2, minDocFreq=1, stopwords=NULL, vocabulary=NULL )
textvector( file, stemming=FALSE, language="german",
            minWordLength=2, minDocFreq=1, stopwords=NULL, vocabulary=NULL )
```

### Arguments

file	filename (may include path).
mydir	the directory path (e.g., "corpus/texts/").
stemming	boolean indicating whether to reduce all terms to their wordstem.
language	specifies language for the stemming / stop-word-removal.
minWordLength	words with less than minWordLength characters will be ignored.
minDocFreq	words of a document appearing less than minDocFreq within that document will be ignored.
stopwords	a stopword list that contains terms the will be ignored.
vocabulary	if specified, only words in this term list will be used for building the matrix ('controlled vocabulary').

### Details

All documents in the specified directory are read and a matrix is composed. The matrix contains in every cell the exact number of appearances (i.e., the term frequency) of every word for all documents. If specified, simple text preprocessing mechanisms are applied (stemming, stopword filtering, wordlength cutoffs).

Stemming thereby uses porter's snowball stemmer (from package `Rstem`).

There are two stopword lists included (for english and for german), which are loaded on demand into the variables `stopwords_de` and `stopwords_en`. They can be activated by calling `data(stopwords_de)` or `data(stopwords_en)`. Attention: the stopword lists have to be already loaded when `textmatrix()` is called.

`textvector()` is a support function that creates a list of term-in-document occurrences.

For every generated matrix, an own environment is added as an attribute which holds the triples that are stored by `setTriple()` and can be retrieved with `getTriple()`.

### Value

textmatrix	the document-term matrix (incl. row and column names).
------------	--

**Author(s)**

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

**See Also**

[wordStem](#), [stopwords\\_de](#), [stopwords\\_en](#), [setTriple](#), [getTriple](#)

**Examples**

```
# create some files
td = tempfile()
dir.create(td)
write( c("dog", "cat", "mouse"), file=paste(td, "D1", sep="/") )
write( c("hamster", "mouse", "sushi"), file=paste(td, "D2", sep="/") )
write( c("dog", "monster", "monster"), file=paste(td, "D3", sep="/") )

# read them, create a document-term matrix
textmatrix(td)

# read them, drop german stopwords
data(stopwords_de)
textmatrix(td, stopwords=stopwords_de)

# read them based on a controlled vocabulary
voc = c("dog", "mouse")
textmatrix(td, vocabulary=voc, minWordLength=1)

# clean up
unlink(td, recursive=TRUE)
```

---

triples

*Bind Triples to a Textmatrix*

---

**Description**

Allows to store, manage and retrieve SPO-triples (subject, predicate, object) bound to the document columns of a document term matrix.

**Usage**

```
getTriple( M, subject, predicate )
setTriple( M, subject, predicate, object )
delTriple( M, subject, predicate, object )
getSubjectId( M, subject )
```

**Arguments**

M	the document term matrix (see <code>textmatrix</code> ).
subject	column number or column name (e.g., "doc3" or 3).
predicate	predicate of the triple sentence (e.g., "has_category" or "has_grade").
object	value of the triple sentence (e.g., "14" or 14).

## Details

SPO-Triples are simple facts of the uniform structure (subject, predicate, object). A subject is typically a document in the given document-term matrix *M*, i.e. its document title (as in the column names) or its column position. A key-value pair (the ‘predicate’ and the ‘object’) can be bound to this ‘subject’.

This can be used, for example, to store classification information about the documents of the text base used.

The triple data is stored in the environment of *M* constructed by `textmatrix()`.

Whenever a matrix has to be used which has not been generated by this function, its class should be set to ‘textmatrix’ and an environment has to be added manually via:

```
class(mymatrix) = "textmatrix"
environment(mymatrix) = new.env()
```

Alternatively, `as.matrix()` can be used to convert a matrix to a textmatrix. To spare memory, the manual method might be of advantage.

In `getTriple()`, the arguments ‘subject’ and ‘predicate’ are optional.

## Value

`textmatrix` the document-term matrix (including row and column names).

## Author(s)

Fridolin Wild <fridolin.wild@wu-wien.ac.at>

## See Also

[textmatrix](#)

## Examples

```
x = matrix(2,2,3) # we fake a document term matrix
rownames(x) = c("dog","mouse") # fake term names
colnames(x) = c("doc1","doc2","doc3") # fake doc titles
class(x) = "textmatrix" # usually done by textmatrix()
environment(x) = new.env() # usually done by textmatrix()

setTriple(x, "doc1", "has_category", "15")
setTriple(x, "doc2", "has_category", "7")
setTriple(x, "doc1", "has_grade", "5")
setTriple(x, "doc1", "has_category", "11")

getTriple(x, "doc1")
getTriple(x, "doc1")[[2]]
getTriple(x, "doc1", "has_category") # -> [1] "15" "11"

delTriple(x, "doc1", "has_category", "15")
getTriple(x, "doc1", "has_category") # -> [1] "11"
```

**Description**

Calculates a weighted document-term matrix according to the chosen local and/or global weighting scheme.

**Usage**

```
lw_tf(m)
lw_logtf(m)
lw_bintf(m)
gw_normalisation(m)
gw_idf(m)
gw_gfidf(m)
entropy(m)
gw_entropy(m)
```

**Arguments**

`m` a document-term matrix.

**Details**

When combining a local and a global weighting scheme to be applied on a given textmatrix  $m$  via  $dtm = lw(m) \cdot gw(m)$ , where

- $m$  is the given document-term matrix,
- $lw(m)$  is one of the local weight functions `lw_tf()`, `lw_logtf()`, `lw_bintf()`, and
- $gw(m)$  is one of the global weight functions `gw_normalisation()`, `gw_idf()`, `gw_gfidf()`, `entropy()`, `gw_entropy()`.

This set of weighting schemes includes the local weightings (lw) raw, log, binary and the global weightings (gw) normalisation, two versions of the inverse document frequency (idf), and entropy in both the original Shannon as well as in a slightly modified, more common version:

`lw_tf()` returns a completely unmodified  $n \times m$  matrix (placebo function).

`lw_logtf()` returns the logarithmised  $n \times m$  matrix.  $\log(m_{i,j} + 1)$  is applied on every cell.

`lw_bintf()` returns binary values of the  $n \times m$  matrix. Every cell is assigned 1, iff the term frequency is not equal to 0.

`gw_normalisation()` returns a normalised  $n \times m$  matrix. Every cell equals 1 divided by the square root of the document vector length.

`gw_idf()` returns the inverse document frequency in a  $n \times m$  matrix. Every cell is 1 plus the logarithmus of the number of documents divided by the number of documents where the term appears.

`gw_gfidf()` returns the global frequency multiplied with idf. Every cell equals the sum of the frequencies of one term divided by the number of documents where the term shows up.

`entropy()` returns the entropy (as defined by Shannon).

`gw_entropy()` returns one plus entropy.

**Value**

Returns the weighted textmatrix of the same size and format as the input matrix.

**Author(s)**

Fridolin Wild (fridolin.wild@wu-wien.ac.at)

**References**

Dumais, S. (1992) *Enhancing Performance in Latent Semantic Indexing (LSI) Retrieval*. Technical Report, Bellcore.

Nakov, P., Popova, A., and Mateev, P. (2001) *Weight functions impact on LSA performance*. In: Proceedings of the Recent Advances in Natural language processing, Bulgaria, pp.187-193.

Shannon, C. (1948) *A Mathematical Theory of Communication*. In: The Bell System Technical Journal 27(July), pp.379–423.

**Examples**

```
## use the logarithmised term frequency as local weight and
## the inverse document frequency as global weight.
```

```
vec1 = c( 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 )
```

```
vec2 = c( 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0 )
```

```
vec3 = c( 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0 )
```

```
matrix = cbind(vec1,vec2, vec3)
```

```
weighted = lw_logtf(matrix)*gw_idf(matrix)
```

# Index

- \*Topic **algebra**
  - as.textmatrix, 1
  - dimcalc, 4
  - fold\_in, 6
  - lsa, 7
- \*Topic **array**
  - print.textmatrix, 9
  - query, 10
  - stopwords, 11
  - summary.textmatrix, 11
  - textmatrix, 12
  - triples, 14
  - weightings, 15
- \*Topic **datasets**
  - stopwords, 11
- \*Topic **multivariate**
  - cosine, 3
- \*Topic **univar**
  - cosine, 3
  
- as.textmatrix, 1, 6, 8
  
- cor, 3
- cosine, 3
  
- delTriple (*triples*), 14
- dimcalc, 4
- dimcalc\_kaiser (*dimcalc*), 4
- dimcalc\_ndocs (*dimcalc*), 4
- dimcalc\_raw (*dimcalc*), 4
- dimcalc\_share, 8
- dimcalc\_share (*dimcalc*), 4
  
- entropy (*weightings*), 15
  
- fold\_in, 2, 6, 8
  
- getSubjectId (*triples*), 14
- getTriple, 13
- getTriple (*triples*), 14
- gw\_entropy (*weightings*), 15
- gw\_gfidf (*weightings*), 15
- gw\_idf, 8
- gw\_idf (*weightings*), 15
- gw\_normalisation (*weightings*), 15
  
- lsa, 2, 5, 6, 7
- lw\_bintf (*weightings*), 15
- lw\_logtf (*weightings*), 15
- lw\_tf (*weightings*), 15
  
- print.textmatrix, 9
  
- query, 10
  
- setTriple, 13
- setTriple (*triples*), 14
- stopwords, 11
- stopwords\_de, 13
- stopwords\_de (*stopwords*), 11
- stopwords\_en, 13
- stopwords\_en (*stopwords*), 11
- stopwords\_english (*stopwords*), 11
- stopwords\_german (*stopwords*), 11
- summary.textmatrix, 11
  
- textmatrix, 2, 6, 8–10, 12, 12, 15
- textvector (*textmatrix*), 12
- triples, 14
  
- weightings, 15
- wordStem, 10, 13