Package: ldaPrototype (via r-universe)

August 23, 2024

Type Package

Title Prototype of Multiple Latent Dirichlet Allocation Runs

Version 0.3.1

Date 2021-09-01

Description Determine a Prototype from a number of runs of Latent Dirichlet Allocation (LDA) measuring its similarities with S-CLOP: A procedure to select the LDA run with highest mean pairwise similarity, which is measured by S-CLOP (Similarity of multiple sets by Clustering with Local Pruning), to all other runs. LDA runs are specified by its assignments leading to estimators for distribution parameters. Repeated runs lead to different results, which we encounter by choosing the most representative LDA run as prototype.

URL https://github.com/JonasRieger/ldaPrototype

BugReports https://github.com/JonasRieger/ldaPrototype/issues

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 3.5.0)

Imports batchtools (>= 0.9.11), checkmate (>= 1.8.5), colorspace (>= 1.4-1), data.table (>= 1.11.2), dendextend, fs (>= 1.2.0), future, lda (>= 1.4.2), parallelMap, progress (>= 1.1.1), stats, utils

Suggests covr, RColorBrewer (>= 1.1-2), testthat, tosca

RoxygenNote 7.2.3

LazyData true

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

RemoteUrl https://github.com/jonasrieger/ldaprototype

RemoteRef HEAD

RemoteSha 3240581a9c8ecb1b178cd568e20f0c618bae38a9

Contents

`ldaPrototype-package` *ldaPrototype: Prototype of Multiple Latent Dirichlet Allocation Runs*

Description

Determine a Prototype from a number of runs of Latent Dirichlet Allocation (LDA) measuring its similarities with S-CLOP: A procedure to select the LDA run with highest mean pairwise similarity, which is measured by S-CLOP (Similarity of multiple sets by Clustering with Local Pruning), to all other runs. LDA runs are specified by its assignments leading to estimators for distribution parameters. Repeated runs lead to different results, which we encounter by choosing the most representative LDA run as prototype.
For bug reports and feature requests please use the issue tracker: https://github.com/JonasRieger/ldaPrototype/issues. Also have a look at the (detailed) example at https://github.com/JonasRieger/ldaPrototype.

Data

`reuters` Example Dataset (91 articles from Reuters) for testing.

Constructor

LDA LDA objects used in this package.
as.LDARep LDARep objects.
as.LDABatch LDABatch objects.

Getter

getTopics Getter for LDA objects.
getJob Getter for LDARep and LDABatch objects.
getSimilarity Getter for TopicSimilarity objects.
getSCLOP Getter for PrototypeLDA objects.
getPrototype Determine the Prototype LDA.

Performing multiple LDAs

LDARep Performing multiple LDAs locally (using parallelization).
LDABatch Performing multiple LDAs on Batch Systems.

Calculation Steps (Workflow) to determine the Prototype LDA

mergeTopics Merge topic matrices from multiple LDAs.
jaccardTopics Calculate topic similarities using the Jaccard coefficient (see Similarity Measures for other possible measures).
dendTopics Create a dendrogram from topic similarities.
SCLOP Determine various S-CLOP values.
pruneSCLOP Prune TopicDendrogram objects.

Similarity Measures

cosineTopics Cosine Similarity.
jaccardTopics Jaccard Coefficient.
jsTopics Jensen-Shannon Divergence.
rboTopics rank-biased overlap.

Shortcuts

getPrototype Shortcut which includes all calculation steps.
LDAPrototype Shortcut which performs multiple LDAs and determines their Prototype.

Author(s)

Maintainer: Jonas Rieger <jonas.rieger@tu-dortmund.de> (ORCID)

References

Rieger, Jonas (2020). "ldaPrototype: A method in R to get a Prototype of multiple Latent Dirichlet Allocations". Journal of Open Source Software, **5**(51), 2181, doi:10.21105/joss.02181.

Rieger, Jonas, Jörg Rahnenführer and Carsten Jentsch (2020). "Improving Latent Dirichlet Allocation: On Reliability of the Novel Method LDAPrototype". In: *Natural Language Processing and Information Systems, NLDB 2020.* LNCS 12089, pp. 118–125, doi:10.1007/9783030513108_11.

Rieger, Jonas, Carsten Jentsch and Jörg Rahnenführer (2022). "LDAPrototype: A Model Selection Algorithm to Improve Reliability of Latent Dirichlet Allocation". Preprint on Research Square, doi:10.21203/rs.3.rs1486359/v1.

See Also

Useful links:

- https://github.com/JonasRieger/ldaPrototype
- Report bugs at https://github.com/JonasRieger/ldaPrototype/issues

---

`as.LDABatch` *LDABatch Constructor*

Description

Constructs a `LDABatch` object for given elements `reg`, `job` and `id`.

Usage

```
as.LDABatch(reg, job, id)

is.LDABatch(obj, verbose = FALSE)
```

Arguments

`reg`	[`Registry`] Registry. See `findDone`.
`job`	[`data.frame` or `integer`] A data.frame or data.table with a column named "job.id" or a vector of integerish job ids. See `reduceResultsList`.
`id`	[`character(1)`] A name for the registry. If not passed, the folder's name is extracted from `reg`.
`obj`	[`R` object] Object to test.
`verbose`	[`logical(1)`] Should test information be given in the console?

Details

Given a `Registry` the function returns a `LDABatch` object, which can be handled using the getter functions at `getJob`.

Value

[named list] with entries id for the registry's folder name, jobs for the submitted jobs' ids and its parameter settings and reg for the registry itself.

See Also

Other constructor functions: LDA(), as.LDARep()

Other batch functions: LDABatch(), getJob(), mergeBatchTopics()

Examples

```
## Not run:
batch = LDABatch(docs = reuters_docs, vocab = reuters_vocab, K = 15, chunk.size = 20)
batch

batch2 = as.LDABatch(reg = getRegistry(batch))
batch2
head(getJob(batch2))

batch3 = as.LDABatch()
batch3

### one way of loading an existing registry ###
batchtools::loadRegistry("LDABatch")
batch = as.LDABatch()

## End(Not run)
```

---

as.LDARep *LDARep Constructor*

Description

Constructs a LDARep object for given elements lda, job and id.

Usage

```
as.LDARep(...)

## Default S3 method:
as.LDARep(lda, job, id, ...)

## S3 method for class 'LDARep'
as.LDARep(x, ...)

is.LDARep(obj, verbose = FALSE)
```

Arguments

...	additional arguments
lda	[named list] List of LDA objects, named by the corresponding "job.id" (integerish). If list is unnamed, names are set.
job	[data.frame or named vector] A data.frame or data.table with named columns (at least) "job.id" (integerish), "K", "alpha", "eta" and "num.iterations" or a named vector with entries (at least) "K", "alpha", "eta" and "num.iterations". If not passed, it is interpreted from param of each LDA.
id	[character(1)] A name for the computation. If not passed, it is set to "LDARep".
x	[named list] LDABatch or LDARep object.
obj	[R object] Object to test.
verbose	[logical(1)] Should test information be given in the console?

Details

Given a list of LDA objects the function returns a LDARep object, which can be handled using the getter functions at getJob.

Value

[named list] with entries id for computation's name, jobs for the parameter settings and lda for the results themselves.

See Also

Other constructor functions: LDA(), as.LDABatch()

Other replication functions: LDAPrototype(), LDARep(), getJob(), mergeRepTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 7, num.iterations = 20)
lda = getLDA(res)

res2 = as.LDARep(lda, id = "newName")
res2
getJob(res2)
getJob(res)

## Not run:
batch = LDABatch(docs = reuters_docs, vocab = reuters_vocab, n = 4, id = "TEMP", K = 30)
res3 = as.LDARep(batch)
res3
```

```
getJob(res3)

## End(Not run)
```

`cosineTopics` *Pairwise Cosine Similarities*

Description

Calculates the similarity of all pairwise topic combinations using the Cosine Similarity.

Usage

```
cosineTopics(topics, progress = TRUE, pm.backend, ncpus)
```

Arguments

`topics`	[named matrix] The counts of vocabularies/words (row wise) in topics (column wise).
`progress`	[logical(1)] Should a nice progress bar be shown? Turning it off, could lead to significantly faster calculation. Default is `TRUE`. If `pm.backend` is set, parallelization is done and no progress bar will be shown.
`pm.backend`	[character(1)] One of "multicore", "socket" or "mpi". If `pm.backend` is set, `parallelStart` is called before computation is started and `parallelStop` is called after.
`ncpus`	[integer(1)] Number of (physical) CPUs to use. If `pm.backend` is passed, default is determined by `availableCores`.

Details

The Cosine Similarity for two topics $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ is calculated by

$$\cos(\theta|\boldsymbol{z}_i, \boldsymbol{z}_j) = \frac{\sum_{v=1}^{V} n_i^{(v)} n_j^{(v)}}{\sqrt{\sum_{v=1}^{V} \left(n_i^{(v)}\right)^2} \sqrt{\sum_{v=1}^{V} \left(n_j^{(v)}\right)^2}}$$

with θ determining the angle between the corresponding count vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$, V is the vocabulary size and $n_k^{(v)}$ is the count of assignments of the v-th word to the k-th topic.

Value

[named list] with entries

sims [lower triangular named matrix] with all pairwise similarities of the given topics.

wordslimit [integer] = vocabulary size. See jaccardTopics for original purpose.

wordsconsidered [integer] = vocabulary size. See jaccardTopics for original purpose.

param [named list] with parameter type [character(1)] = "Cosine Similarity".

See Also

Other TopicSimilarity functions: dendTopics(), getSimilarity(), jaccardTopics(), jsTopics(), rboTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
cosine = cosineTopics(topics)
cosine

sim = getSimilarity(cosine)
dim(sim)
```

---

dendTopics *Topic Dendrogram*

Description

Builds a dendrogram for topics based on their pairwise similarities using the cluster algorithm hclust.

Usage

```
dendTopics(sims, ind, method = "complete")

## S3 method for class 'TopicDendrogram'
plot(x, pruning, pruning.par, ...)
```

Arguments

sims	[TopicSimilarity object or lower triangular named matrix] TopicSimilarity object or pairwise jaccard similarities of underlying topics as the sims element from TopicSimilarity objects. The topic names should be formatted as <*Run X*>.<*Topic Y*>, so that the name before the first dot identifies the LDA run.

ind	[integer, logical or character] An integerish vector (or logical of the same length as the number of rows and columns) for specifying the topics taken into account. Alternatively a character vector can be passed. Then, all topics are taken for which the name contain at least one of the phrases in ind (see grepl). By default all topics are considered.
method	[character(1)] The agglomeration method. See hclust.
x	an R object.
pruning	[list of dendrograms] PruningSCLOP object specifying the best possible local pruning state.
pruning.par	[list] List of parameters to mark the pruning. See section "Details" at dendTopics for default parameters. Types for marking the pruning state are "abline", "color" and "both".
...	additional arguments.

Details

The label´s colors are determined based on their Run belonging using rainbow_hcl by default. Colors can be manipulated using labels_colors. Analogously, the labels themself can be manipulated using labels. For both the function order.dendrogram is useful.

The resulting dendrogram can be plotted. In addition, it is possible to mark a pruning state in the plot, either by color or by separator lines (or both) setting pruning.par. For the default values of pruning.par call the corresponding function on any PruningSCLOP object.

Value

[dendrogram] TopicDendrogram object (and dendrogram object) of all considered topics.

See Also

Other plot functions: pruneSCLOP()

Other TopicSimilarity functions: cosineTopics(), getSimilarity(), jaccardTopics(), jsTopics(), rboTopics()

Other workflow functions: LDARep(), SCLOP(), getPrototype(), jaccardTopics(), mergeTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
jacc = jaccardTopics(topics, atLeast = 2)
sim = getSimilarity(jacc)

dend = dendTopics(jacc)
dend2 = dendTopics(sim)


plot(dend)
```

```
plot(dendTopics(jacc, ind = c("Rep2", "Rep3")))


pruned = pruneSCLOP(dend)

plot(dend, pruning = pruned)
plot(dend, pruning = pruned, pruning.par = list(type = "color"))
plot(dend, pruning = pruned, pruning.par = list(type = "both", lty = 1, lwd = 2, col = "red"))

dend2 = dendTopics(jacc, ind = c("Rep2", "Rep3"))
plot(dend2, pruning = pruneSCLOP(dend2), pruning.par = list(lwd = 2, col = "darkgrey"))
```

`getJob` *Getter and Setter for LDARep and LDABatch*

Description

Returns the job ids and its parameter set (`getJob`) or the (registry's) id (`getID`) for a `LDABatch` or `LDARep` object. `getRegistry` returns the registry itself for a `LDABatch` object. `getLDA` returns the list of `LDA` objects for a `LDABatch` or `LDARep` object. In addition, you can specify one or more LDAs by their id(s).
`setFilDir` sets the registry's file directory for a `LDABatch` object. This is useful if you move the registry´s folder, e.g. if you do your calculations on a batch system, but want to do your evaluation on your desktop computer.

Usage

```
getJob(x)

getID(x)

getRegistry(x)

getLDA(x, job, reduce, all)

setFileDir(x, file.dir)
```

Arguments

`x`	[named `list`] `LDABatch` or `LDARep` object.
`job`	[`data.frame` or `integer`] A data.frame or data.table with a column named "job.id" or a vector of integerish job ids.

reduce	[logical(1)] If the list of LDAs contains only one element, should the list be reduced and the single (unnamed) element be returned? Default is TRUE.
all	not implemented for LDABatch and LDARep object. See getLDA
file.dir	[Vector to be coerced to a fs_path object.] New file directory to overwrite the registry's old one. This can be useful if the registry is transferred from a batch system.

See Also

Other getter functions: getSCLOP(), getSimilarity(), getTopics()

Other replication functions: LDAPrototype(), LDARep(), as.LDARep(), mergeRepTopics()

Other batch functions: LDABatch(), as.LDABatch(), mergeBatchTopics()

---

getPrototype *Determine the Prototype LDA*

Description

Returns the Prototype LDA of a set of LDAs. This set is given as LDABatch object, LDARep object, or as list of LDAs. If the matrix of S-CLOP scores sclop is passed, no calculation is needed/done.

Usage

```
getPrototype(...)

## S3 method for class 'LDARep'
getPrototype(
  x,
  vocab,
  limit.rel,
  limit.abs,
  atLeast,
  progress = TRUE,
  pm.backend,
  ncpus,
  keepTopics = FALSE,
  keepSims = FALSE,
  keepLDAs = FALSE,
  sclop,
  ...
)

## S3 method for class 'LDABatch'
getPrototype(
  x,
```

```
  vocab,
  limit.rel,
  limit.abs,
  atLeast,
  progress = TRUE,
  pm.backend,
  ncpus,
  keepTopics = FALSE,
  keepSims = FALSE,
  keepLDAs = FALSE,
  sclop,
  ...
)

## Default S3 method:
getPrototype(
  lda,
  vocab,
  id,
  job,
  limit.rel,
  limit.abs,
  atLeast,
  progress = TRUE,
  pm.backend,
  ncpus,
  keepTopics = FALSE,
  keepSims = FALSE,
  keepLDAs = FALSE,
  sclop,
  ...
)
```

Arguments

`...`	additional arguments
`x`	[`named list`] `LDABatch` or `LDARep` object.
`vocab`	[`character`] Vocabularies taken into consideration for merging topic matrices. Not considered, if `sclop` is passed. Default is the vocabulary of the first LDA.
`limit.rel`	[0,1] See `jaccardTopics`. Default is `1/500`. Not considered for calculation, if `sclop` is passed. But should be passed determining the correct value for the resulting object.
`limit.abs`	[`integer(1)`] See `jaccardTopics`. Default is `10`. Not considered for calculation, if `sclop`

is passed. But should be passed determining the correct value for the resulting object.

`atLeast` [integer(1)]
See jaccardTopics. Default is 0. Not considered for calculation, if sclop is passed. But should be passed determining the correct value for the resulting object.

`progress` [logical(1)]
Should a nice progress bar be shown for the steps of mergeTopics and jaccardTopics? Turning it off, could lead to significantly faster calculation. Default ist TRUE. Not considered, if sclop is passed.

`pm.backend` [character(1)]
One of "multicore", "socket" or "mpi". If pm.backend is set, parallelStart is called before computation is started and parallelStop is called after. Not considered, if sclop is passed.

`ncpus` [integer(1)]
Number of (physical) CPUs to use. If pm.backend is passed, default is determined by availableCores. Not considered, if sclop is passed.

`keepTopics` [logical(1)]
Should the merged topic matrix from mergeTopics be kept? Not considered, if sclop is passed.

`keepSims` [logical(1)]
Should the calculated topic similarities matrix from jaccardTopics be kept? Not considered, if sclop is passed.

`keepLDAs` [logical(1)]
Should the considered LDAs be kept?

`sclop` [symmetrical named matrix]
(optional) All pairwise S-CLOP scores of the given LDA runs determined by SCLOP.pairwise. Matching of names is not implemented yet, so order matters.

`lda` [named list]
List of LDA objects, named by the corresponding "job.id".

`id` [character(1)]
A name for the computation. If not passed, it is set to "LDARep". Not considered for LDABatch or LDARep objects.

`job` [data.frame or named vector]
A data.frame or data.table with named columns (at least) "job.id" (integerish), "K", "alpha", "eta" and "num.iterations" or a named vector with entries (at least) "K", "alpha", "eta" and "num.iterations". If not passed, it is interpreted from param of each LDA. Not considered for LDABatch or LDARep objects.

Details

While LDAPrototype marks the overall shortcut for performing multiple LDA runs and choosing the Prototype of them, getPrototype just hooks up at determining the Prototype. The generation of multiple LDAs has to be done before use of this function. The function is flexible enough to use it at at least two steps/parts of the analysis: After generating the LDAs (no matter whether as LDABatch or LDARep object) or after determing the pairwise SCLOP values.

To save memory a lot of interim calculations are discarded by default.

If you use parallel computation, no progress bar is shown.

For details see the details sections of the workflow functions.

Value

[named list] with entries

id [character(1)] See above.

protoid [character(1)] Name (ID) of the determined Prototype LDA.

lda List of LDA objects of the determined Prototype LDA and - if keepLDAs is TRUE - all considered LDAs.

jobs [data.table] with parameter specifications for the LDAs.

param [named list] with parameter specifications for limit.rel [0,1], limit.abs [integer(1)] and atLeast [integer(1)]. See above for explanation.

topics [named matrix] with the count of vocabularies (row wise) in topics (column wise).

sims [lower triangular named matrix] with all pairwise jaccard similarities of the given topics.

wordslimit [integer] with counts of words determined as relevant based on limit.rel and limit.abs.

wordsconsidered [integer] with counts of considered words for similarity calculation. Could differ from wordslimit, if atLeast is greater than zero.

sclop [symmetrical named matrix] with all pairwise S-CLOP scores of the given LDA runs.

See Also

Other shortcut functions: LDAPrototype()

Other PrototypeLDA functions: LDAPrototype(), getSCLOP()

Other workflow functions: LDARep(), SCLOP(), dendTopics(), jaccardTopics(), mergeTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab,
   n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
jacc = jaccardTopics(topics, atLeast = 2)
dend = dendTopics(jacc)
sclop = SCLOP.pairwise(jacc)

getPrototype(lda = getLDA(res), sclop = sclop)

proto = getPrototype(res, vocab = reuters_vocab, keepSims = TRUE,
   limit.abs = 20, atLeast = 10)
proto
getPrototype(proto) # = getLDA(proto)
getConsideredWords(proto)
# > 10 if there is more than one word which is the 10-th often word (ties)
getRelevantWords(proto)
getSCLOP(proto)
```

getSCLOP *Getter for PrototypeLDA*

Description

Returns the corresponding element of a PrototypeLDA object.

Usage

```
getSCLOP(x)

## S3 method for class 'PrototypeLDA'
getSimilarity(x)

## S3 method for class 'PrototypeLDA'
getRelevantWords(x)

## S3 method for class 'PrototypeLDA'
getConsideredWords(x)

getMergedTopics(x)

getPrototypeID(x)

## S3 method for class 'PrototypeLDA'
getLDA(x, job, reduce = TRUE, all = FALSE)

## S3 method for class 'PrototypeLDA'
getID(x)

## S3 method for class 'PrototypeLDA'
getParam(x)

## S3 method for class 'PrototypeLDA'
getJob(x)
```

Arguments

x	[named list] PrototypeLDA object.
job	[data.frame or integer] A data.frame or data.table with a column named "job.id" or a vector of integerish job ids. Default is the (integerish) ID of the Prototype LDA.
reduce	[logical(1)] If the list of LDAs contains only one element, should the list be reduced and the single (unnamed) element be returned? Default is TRUE. Not considered, if all is TRUE.

all	[logical(1)] Shortcut for job: Should all stored LDAs be returned?

See Also

Other getter functions: getJob(), getSimilarity(), getTopics()

Other PrototypeLDA functions: LDAPrototype(), getPrototype()

---

getSimilarity *Getter for TopicSimilarity*

Description

Returns the corresponding element of a TopicSimilarity object.

Usage

```
getSimilarity(x)

getRelevantWords(x)

getConsideredWords(x)

## S3 method for class 'TopicSimilarity'
getParam(x)
```

Arguments

x	[named list] TopicSimilarity object.

See Also

Other getter functions: getJob(), getSCLOP(), getTopics()

Other TopicSimilarity functions: cosineTopics(), dendTopics(), jaccardTopics(), jsTopics(), rboTopics()

getTopics *Getter for LDA*

Description

Returns the corresponding element of a LDA object. `getEstimators` computes the estimators for `phi` and `theta`.

Usage

```
getTopics(x)

getAssignments(x)

getDocument_sums(x)

getDocument_expects(x)

getLog.likelihoods(x)

getParam(x)

getK(x)

getAlpha(x)

getEta(x)

getNum.iterations(x)

getEstimators(x)
```

Arguments

x	[named list] LDA object.

Details

The estimators for `phi` and `theta` in

$$w_n^{(m)} \mid T_n^{(m)}, \boldsymbol{\phi}_k \sim \mathsf{Discrete}(\boldsymbol{\phi}_k),$$

$$\boldsymbol{\phi}_k \sim \mathsf{Dirichlet}(\eta),$$

$$T_n^{(m)} \mid \boldsymbol{\theta}_m \sim \mathsf{Discrete}(\boldsymbol{\theta}_m),$$

$$\boldsymbol{\theta}_m \sim \mathsf{Dirichlet}(\alpha)$$

are calculated referring to Griffiths and Steyvers (2004) by

$$\hat{\phi}_{k,v} = \frac{n_k^{(v)} + \eta}{n_k + V\eta},$$

$$\hat{\theta}_{m,k} = \frac{n_k^{(m)} + \alpha}{N^{(m)} + K\alpha}$$

with V is the vocabulary size, K is the number of modeled topics; $n_k^{(v)}$ is the count of assignments of the v-th word to the k-th topic. Analogously, $n_k^{(m)}$ is the count of assignments of the m-th text to the k-th topic. $N^{(m)}$ is the total number of assigned tokens in text m and n_k the total number of assigned tokens to topic k.

References

Griffiths, Thomas L. and Mark Steyvers (2004). "Finding scientific topics". In: *Proceedings of the National Academy of Sciences* **101** (suppl 1), pp.5228–5235, doi:10.1073/pnas.0307752101.

See Also

Other getter functions: `getJob()`, `getSCLOP()`, `getSimilarity()`

Other LDA functions: `LDABatch()`, `LDARep()`, `LDA()`

`jaccardTopics` *Pairwise Jaccard Coefficients*

Description

Calculates the similarity of all pairwise topic combinations using a modified Jaccard Coefficient.

Usage

```
jaccardTopics(
  topics,
  limit.rel,
  limit.abs,
  atLeast,
  progress = TRUE,
  pm.backend,
  ncpus
)
```

Arguments

topics	[named matrix] The counts of vocabularies/words (row wise) in topics (column wise).
limit.rel	[0,1] A relative lower bound limit for which words are taken into account. Those words are taken as relevant for a topic that have a count higher than limit.rel multiplied by the total count of the given topic. Default is 1/500.
limit.abs	[integer(1)] An absolute lower bound limit for which words are taken into account. All words are taken as relevant for a topic that have a count higher than limit.abs. Default is 10.
atLeast	[integer(1)] An absolute count of how many words are at least considered as relevant for a topic. Default is 0.
progress	[logical(1)] Should a nice progress bar be shown? Turning it off, could lead to significantly faster calculation. Default is TRUE. If pm.backend is set, parallelization is done and no progress bar will be shown.
pm.backend	[character(1)] One of "multicore", "socket" or "mpi". If pm.backend is set, parallelStart is called before computation is started and parallelStop is called after.
ncpus	[integer(1)] Number of (physical) CPUs to use. If pm.backend is passed, default is determined by availableCores.

Details

The modified Jaccard Coefficient for two topics $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ is calculated by

$$J_m(\boldsymbol{z}_i, \boldsymbol{z}_j \mid \boldsymbol{c}) = \frac{\sum_{v=1}^{V} 1_{\left\{n_i^{(v)} > c_i \,\wedge\, n_j^{(v)} > c_j\right\}} \left(n_i^{(v)}, n_j^{(v)}\right)}{\sum_{v=1}^{V} 1_{\left\{n_i^{(v)} > c_i \,\vee\, n_j^{(v)} > c_j\right\}} \left(n_i^{(v)}, n_j^{(v)}\right)}$$

with V is the vocabulary size and $n_k^{(v)}$ is the count of assignments of the v-th word to the k-th topic. The threshold vector $\boldsymbol{c}$ is determined by the maximum threshold of the user given lower bounds limit.rel and limit.abs. In addition, at least atLeast words per topic are considered for calculation. According to this, if there are less than atLeast words considered as relevant after applying limit.rel and limit.abs the atLeast most common words per topic are taken to determine topic similarities.

The procedure of determining relevant words is executed for each topic individually. The values wordslimit and wordsconsidered describes the number of relevant words per topic.

Value

[named list] with entries

`sims` [`lower triangular named matrix`] with all pairwise jaccard similarities of the given topics.

`wordslimit` [`integer`] with counts of words determined as relevant based on `limit.rel` and `limit.abs`.

`wordsconsidered` [`integer`] with counts of considered words for similarity calculation. Could differ from `wordslimit`, if `atLeast` is greater than zero.

`param` [`named list`] with parameter specifications for `type` [`character(1)`] = `"Jaccard Coefficient"`, `limit.rel` [0,1], `limit.abs` [`integer(1)`] and `atLeast` [`integer(1)`]. See above for explanation.

See Also

Other TopicSimilarity functions: `cosineTopics()`, `dendTopics()`, `getSimilarity()`, `jsTopics()`, `rboTopics()`

Other workflow functions: `LDARep()`, `SCLOP()`, `dendTopics()`, `getPrototype()`, `mergeTopics()`

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
jacc = jaccardTopics(topics, atLeast = 2)
jacc

n1 = getConsideredWords(jacc)
n2 = getRelevantWords(jacc)
(n1 - n2)[n1 - n2 != 0]

sim = getSimilarity(jacc)
dim(sim)

# Comparison to Cosine and Jensen-Shannon (more interesting on large datasets)
cosine = cosineTopics(topics)
js = jsTopics(topics)

sims = list(jaccard = sim, cosine = getSimilarity(cosine), js = getSimilarity(js))
pairs(do.call(cbind, lapply(sims, as.vector)))
```

---

jsTopics *Pairwise Jensen-Shannon Similarities (Divergences)*

Description

Calculates the similarity of all pairwise topic combinations using the Jensen-Shannon Divergence.

Usage

```
jsTopics(topics, epsilon = 1e-06, progress = TRUE, pm.backend, ncpus)
```

Arguments

topics	[named matrix] The counts of vocabularies/words (row wise) in topics (column wise).
epsilon	[numeric(1)] Numerical value added to topics to ensure computability. See details. Default is 1e-06.
progress	[logical(1)] Should a nice progress bar be shown? Turning it off, could lead to significantly faster calculation. Default is TRUE. If pm.backend is set, parallelization is done and no progress bar will be shown.
pm.backend	[character(1)] One of "multicore", "socket" or "mpi". If pm.backend is set, parallelStart is called before computation is started and parallelStop is called after.
ncpus	[integer(1)] Number of (physical) CPUs to use. If pm.backend is passed, default is determined by availableCores.

Details

The Jensen-Shannon Similarity for two topics $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ is calculated by

$$JS(\boldsymbol{z}_i, \boldsymbol{z}_j) = 1 - \left(KLD\left(\boldsymbol{p}_i, \frac{\boldsymbol{p}_i + \boldsymbol{p}_j}{2}\right) + KLD\left(\boldsymbol{p}_j, \frac{\boldsymbol{p}_i + \boldsymbol{p}_j}{2}\right)\right)/2$$

$$= 1 - KLD(\boldsymbol{p}_i, \boldsymbol{p}_i + \boldsymbol{p}_j)/2 - KLD(\boldsymbol{p}_j, \boldsymbol{p}_i + \boldsymbol{p}_j)/2 - \log(2)$$

with V is the vocabulary size, $\boldsymbol{p}_k = \left(p_k^{(1)}, ..., p_k^{(V)}\right)$, and $p_k^{(v)}$ is the proportion of assignments of the v-th word to the k-th topic. KLD defines the Kullback-Leibler Divergence calculated by

$$KLD(\boldsymbol{p}_k, \boldsymbol{p}_\Sigma) = \sum_{v=1}^{V} p_k^{(v)} \log \frac{p_k^{(v)}}{p_\Sigma^{(v)}}.$$

There is an epsilon added to every $n_k^{(v)}$, the count (not proportion) of assignments to ensure computability with respect to zeros.

Value

[named list] with entries

sims [lower triangular named matrix] with all pairwise similarities of the given topics.

wordslimit [integer] = vocabulary size. See jaccardTopics for original purpose.

wordsconsidered [integer] = vocabulary size. See jaccardTopics for original purpose.

param [named list] with parameter specifications for type [character(1)] = "Cosine Similarity" and epsilon [numeric(1)]. See above for explanation.

See Also

Other TopicSimilarity functions: cosineTopics(), dendTopics(), getSimilarity(), jaccardTopics(), rboTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
js = jsTopics(topics)
js

sim = getSimilarity(js)
dim(sim)

js1 = jsTopics(topics, epsilon = 1)
sim1 = getSimilarity(js1)
summary((sim1-sim)[lower.tri(sim)])
plot(sim, sim1, xlab = "epsilon = 1e-6", ylab = "epsilon = 1")
```

LDA *LDA Object*

Description

Constructor for LDA objects used in this package.

Usage

```
LDA(
  x,
  param,
  assignments,
  topics,
  document_sums,
  document_expects,
  log.likelihoods
)

as.LDA(
  x,
  param,
  assignments,
  topics,
  document_sums,
  document_expects,
  log.likelihoods
)
```

```
is.LDA(obj, verbose = FALSE)
```

Arguments

x	[named list] Output from lda.collapsed.gibbs.sampler. Alternatively each element can be passed for individual results. Individually set elements overwrite elements from x.
param	[named list] Parameters of the function call lda.collapsed.gibbs.sampler. List always should contain names "K", "alpha", "eta" and "num.iterations".
assignments	Individual element for LDA object.
topics	Individual element for LDA object.
document_sums	Individual element for LDA object.
document_expects	Individual element for LDA object.
log.likelihoods	Individual element for LDA object.
obj	[R object] Object to test.
verbose	[logical(1)] Should test information be given in the console?

Details

The functions LDA and as.LDA do exactly the same. If you call LDA on an object x which already is of the structure of an LDA object (in particular a LDA object itself), the additional arguments param, assignments, ... may be used to override the specific elements.

Value

[named list] LDA object.

See Also

Other constructor functions: as.LDABatch(), as.LDARep()

Other LDA functions: LDABatch(), LDARep(), getTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 1, K = 10)
lda = getLDA(res)

LDA(lda)
# does not change anything

LDA(lda, assignments = NULL)
```

```
# creates a new LDA object without the assignments element

LDA(param = getParam(lda), topics = getTopics(lda))
# creates a new LDA object with elements param and topics
```

LDABatch *LDA Replications on a Batch System*

Description

Performs multiple runs of Latent Dirichlet Allocation on a batch system using the batchtools-package.

Usage

```
LDABatch(
  docs,
  vocab,
  n = 100,
  seeds,
  id = "LDABatch",
  load = FALSE,
  chunk.size = 1,
  resources,
  ...
)
```

Arguments

docs	[list] Documents as received from LDAprep.
vocab	[character] Vocabularies passed to lda.collapsed.gibbs.sampler. For additional (and necessary) arguments passed, see ellipsis (three-dot argument).
n	[integer(1)] Number of Replications.
seeds	[integer(n)] Random Seeds for each Replication.
id	[character(1)] Name for the registry's folder.
load	[logical(1)] If a folder with name id exists: should the existing registry be loaded?
chunk.size	[integer(1)] Requested chunk size for each single chunk. See chunk.
resources	[named list] Computational resources for the jobs to submit. See submitJobs.

... additional arguments passed to lda.collapsed.gibbs.sampler. Arguments will be coerced to a vector of length n. Default parameters are alpha = eta = 1/K and num.iterations = 200. There is no default for K.

Details

The function generates multiple LDA runs with the possibility of using a batch system. The integration is done by the batchtools-package. After all jobs of the corresponding registry are terminated, the whole registry can be ported to your local computer for further analysis.

The function returns a LDABatch object. You can receive results and all other elements of this object with getter functions (see getJob).

Value

[named list] with entries id for the registry's folder name, jobs for the submitted jobs' ids and its parameter settings and reg for the registry itself.

See Also

Other batch functions: as.LDABatch(), getJob(), mergeBatchTopics()

Other LDA functions: LDARep(), LDA(), getTopics()

Examples

```
## Not run:
batch = LDABatch(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 15)
batch
getRegistry(batch)
getJob(batch)
getLDA(batch, 2)

batch2 = LDABatch(docs = reuters_docs, vocab = reuters_vocab, K = 15, chunk.size = 20)
batch2
head(getJob(batch2))

## End(Not run)
```

LDAPrototype *Determine the Prototype LDA*

Description

Performs multiple runs of LDA and computes the Prototype LDA of this set of LDAs.

Usage

```
LDAPrototype(
  docs,
  vocabLDA,
  vocabMerge = vocabLDA,
  n = 100,
  seeds,
  id = "LDARep",
  pm.backend,
  ncpus,
  limit.rel,
  limit.abs,
  atLeast,
  progress = TRUE,
  keepTopics = FALSE,
  keepSims = FALSE,
  keepLDAs = FALSE,
  ...
)
```

Arguments

docs	[list] Documents as received from LDAprep.
vocabLDA	[character] Vocabularies passed to lda.collapsed.gibbs.sampler. For additional (and necessary) arguments passed, see ellipsis (three-dot argument).
vocabMerge	[character] Vocabularies taken into consideration for merging topic matrices.
n	[integer(1)] Number of Replications.
seeds	[integer(n)] Random Seeds for each Replication.
id	[character(1)] Name for the computation.
pm.backend	[character(1)] One of "multicore", "socket" or "mpi". If pm.backend is set, parallelStart is called before computation is started and parallelStop is called after.
ncpus	[integer(1)] Number of (physical) CPUs to use. If pm.backend is passed, default is determined by availableCores.
limit.rel	[0,1] See jaccardTopics. Default is 1/500.
limit.abs	[integer(1)] See jaccardTopics. Default is 10.

atLeast	[integer(1)] See jaccardTopics. Default is 0.
progress	[logical(1)] Should a nice progress bar be shown for the steps of mergeTopics and jaccardTopics? Turning it off, could lead to significantly faster calculation. Default ist TRUE.
keepTopics	[logical(1)] Should the merged topic matrix from mergeTopics be kept?
keepSims	[logical(1)] Should the calculated topic similarities matrix from jaccardTopics be kept?
keepLDAs	[logical(1)] Should the considered LDAs be kept?
...	additional arguments passed to lda.collapsed.gibbs.sampler. Arguments will be coerced to a vector of length n. Default parameters are alpha = eta = 1/K and num.iterations = 200. There is no default for K.

Details

While LDAPrototype marks the overall shortcut for performing multiple LDA runs and choosing the Prototype of them, getPrototype just hooks up at determining the Prototype. The generation of multiple LDAs has to be done before use of getPrototype.

To save memory a lot of interim calculations are discarded by default.

If you use parallel computation, no progress bar is shown.

For details see the details sections of the workflow functions at getPrototype.

Value

[named list] with entries

id [character(1)] See above.

protoid [character(1)] Name (ID) of the determined Prototype LDA.

lda List of LDA objects of the determined Prototype LDA and - if keepLDAs is TRUE - all considered LDAs.

jobs [data.table] with parameter specifications for the LDAs.

param [named list] with parameter specifications for limit.rel [0,1], limit.abs [integer(1)] and atLeast [integer(1)]. See above for explanation.

topics [named matrix] with the count of vocabularies (row wise) in topics (column wise).

sims [lower triangular named matrix] with all pairwise jaccard similarities of the given topics.

wordslimit [integer] with counts of words determined as relevant based on limit.rel and limit.abs.

wordsconsidered [integer] with counts of considered words for similarity calculation. Could differ from wordslimit, if atLeast is greater than zero.

sclop [symmetrical named matrix] with all pairwise S-CLOP scores of the given LDA runs.

See Also

Other shortcut functions: `getPrototype()`

Other PrototypeLDA functions: `getPrototype()`, `getSCLOP()`

Other replication functions: `LDARep()`, `as.LDARep()`, `getJob()`, `mergeRepTopics()`

Examples

```
res = LDAPrototype(docs = reuters_docs, vocabLDA = reuters_vocab,
   n = 4, K = 10, num.iterations = 30)
res
getPrototype(res) # = getLDA(res)
getSCLOP(res)

res = LDAPrototype(docs = reuters_docs, vocabLDA = reuters_vocab,
   n = 4, K = 10, num.iterations = 30, keepLDAs = TRUE)
res
getLDA(res, all = TRUE)
getPrototypeID(res)
getParam(res)
```

LDARep *LDA Replications*

Description

Performs multiple runs of Latent Dirichlet Allocation.

Usage

```
LDARep(docs, vocab, n = 100, seeds, id = "LDARep", pm.backend, ncpus, ...)
```

Arguments

docs	[list] Documents as received from `LDAprep`.
vocab	[character] Vocabularies passed to `lda.collapsed.gibbs.sampler`. For additional (and necessary) arguments passed, see ellipsis (three-dot argument).
n	[integer(1)] Number of Replications.
seeds	[integer(n)] Random Seeds for each Replication.
id	[character(1)] Name for the computation.

pm.backend [character(1)]
One of "multicore", "socket" or "mpi". If pm.backend is set, parallelStart is called before computation is started and parallelStop is called after.

ncpus [integer(1)]
Number of (physical) CPUs to use. If pm.backend is passed, default is determined by availableCores.

... additional arguments passed to lda.collapsed.gibbs.sampler. Arguments will be coerced to a vector of length n. Default parameters are alpha = eta = 1/K and num.iterations = 200. There is no default for K.

Details

The function generates multiple LDA runs with the possibility of using parallelization. The integration is done by the parallelMap-package.

The function returns a LDARep object. You can receive results and all other elements of this object with getter functions (see getJob).

Value

[named list] with entries id for computation's name, jobs for the parameter settings and lda for the results itself.

See Also

Other replication functions: LDAPrototype(), as.LDARep(), getJob(), mergeRepTopics()

Other LDA functions: LDABatch(), LDA(), getTopics()

Other workflow functions: SCLOP(), dendTopics(), getPrototype(), jaccardTopics(), mergeTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, seeds = 1:4,
   id = "myComputation", K = 7:10, alpha = 1, eta = 0.01, num.iterations = 20)
res
getJob(res)
getID(res)
getLDA(res, 4)


LDARep(docs = reuters_docs, vocab = reuters_vocab,
   K = 10, num.iterations = 100, pm.backend = "socket")
```

`mergeBatchTopics` *Merge LDA Topic Matrices*

Description

Collects LDA results from a given registry and merges their topic matrices for a given set of vocabularies.

Usage

```
mergeBatchTopics(...)

## S3 method for class 'LDABatch'
mergeBatchTopics(x, vocab, progress = TRUE, ...)

## Default S3 method:
mergeBatchTopics(vocab, reg, job, id, progress = TRUE, ...)
```

Arguments

`...`	additional arguments
`x`	[named `list`] `LDABatch` object. Alternatively `job`, `reg` and `id` can be passed or their defaults are taken.
`vocab`	[`character`] Vocabularies taken into consideration for merging topic matrices. Default is the vocabulary of the first LDA.
`progress`	[`logical(1)`] Should a nice progress bar be shown? Turning it off, could lead to significantly faster calculation. Default ist `TRUE`.
`reg`	[`Registry`] Registry. See `reduceResultsList`.
`job`	[`data.frame` or `integer`] A data.frame or data.table with a column named "job.id" or a vector of integerish job ids. See `reduceResultsList`.
`id`	[`character(1)`] A name for the registry. If not passed, the folder's name is extracted from `reg`.

Details

For details and examples see `mergeTopics`.

Value

[named `matrix`] with the count of vocabularies (row wise) in topics (column wise).

See Also

Other merge functions: `mergeRepTopics()`, `mergeTopics()`

Other batch functions: `LDABatch()`, `as.LDABatch()`, `getJob()`

`mergeRepTopics` *Merge LDA Topic Matrices*

Description

Collects LDA results from a list of replicated runs and merges their topic matrices for a given set of vocabularies.

Usage

```
mergeRepTopics(...)

## S3 method for class 'LDARep'
mergeRepTopics(x, vocab, progress = TRUE, ...)

## Default S3 method:
mergeRepTopics(lda, vocab, id, progress = TRUE, ...)
```

Arguments

`...`	additional arguments
`x`	[named `list`] `LDARep` object. Alternatively `lda` and `id` can be passed.
`vocab`	[`character`] Vocabularies taken into consideration for merging topic matrices. Default is the vocabulary of the first LDA.
`progress`	[`logical(1)`] Should a nice progress bar be shown? Turning it off, could lead to significantly faster calculation. Default ist `TRUE`.
`lda`	[named `list`] List of `LDA` objects, named by the corresponding "job.id".
`id`	[`character(1)`] Name for the computation. Default is "LDARep".

Details

For details and examples see `mergeTopics`.

Value

[named `matrix`] with the count of vocabularies (row wise) in topics (column wise).

See Also

Other merge functions: `mergeBatchTopics()`, `mergeTopics()`

Other replication functions: `LDAPrototype()`, `LDARep()`, `as.LDARep()`, `getJob()`

`mergeTopics` *Merge LDA Topic Matrices*

Description

Generic function, which collects LDA results and merges their topic matrices for a given set of vocabularies.

Usage

```
mergeTopics(x, vocab, progress = TRUE)
```

Arguments

`x`	[`named list`] `LDARep` or `LDABatch` object.
`vocab`	[`character`] Vocabularies taken into consideration for merging topic matrices.
`progress`	[`logical(1)`] Should a nice progress bar be shown? Turning it off, could lead to significantly faster calculation. Default ist `TRUE`.

Details

This function uses the function `mergeRepTopics` or `mergeBatchTopics`. The topic matrices are transponed and cbinded, so that the resulting matrix contains the counts of vocabularies/words (row wise) in topics (column wise).

Value

[`named matrix`] with the count of vocabularies (row wise) in topics (column wise).

See Also

Other merge functions: `mergeBatchTopics()`, `mergeRepTopics()`

Other workflow functions: `LDARep()`, `SCLOP()`, `dendTopics()`, `getPrototype()`, `jaccardTopics()`

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
dim(topics)
length(reuters_vocab)

## Not run:
res = LDABatch(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
dim(topics)
length(reuters_vocab)

## End(Not run)
```

---

pruneSCLOP *Local Pruning State of Topic Dendrograms*

Description

The function SCLOP calculates the S-CLOP value for the best possible local pruning state of a dendrogram from dendTopics. The function pruneSCLOP supplies the corresponding pruning state itself.

Usage

```
pruneSCLOP(dend)

## S3 method for class 'PruningSCLOP'
plot(x, dend, pruning.par, ...)

pruning.par(pruning)
```

Arguments

dend	[dendrogram] TopicDendrogram (and dendrogram) object of all considered topics as the output from dendTopics.
x	an R object.
pruning.par	[list] List of parameters to mark the pruning. See section "Details" at dendTopics for default parameters. Types for marking the pruning state are "abline", "color" and "both".
...	additional arguments.
pruning	[list of dendrograms] PruningSCLOP object specifying the best possible local pruning state.

Details

For details of computing the S-CLOP values see SCLOP.

For details and examples of plotting the pruning state see dendTopics.

Value

[list of dendrograms] PruningSCLOP object specifying the best possible local pruning state.

See Also

Other plot functions: dendTopics()

Other SCLOP functions: SCLOP()

---

rboTopics *Pairwise RBO Similarities*

Description

Calculates the similarity of all pairwise topic combinations using the rank-biased overlap (RBO) Similarity.

Usage

```
rboTopics(topics, k, p, progress = TRUE, pm.backend, ncpus)
```

Arguments

topics	[named matrix] The counts of vocabularies/words (row wise) in topics (column wise).
k	[integer(1)] Maximum depth for evaluation. Words down to this rank are considered for the calculation of similarities.
p	[0,1] Weighting parameter. Lower values emphasizes top ranked words while values that go towards 1 correspond to equal weights for each evaluation depth.
progress	[logical(1)] Should a nice progress bar be shown? Turning it off, could lead to significantly faster calculation. Default is TRUE. If pm.backend is set, parallelization is done and no progress bar will be shown.
pm.backend	[character(1)] One of "multicore", "socket" or "mpi". If pm.backend is set, parallelStart is called before computation is started and parallelStop is called after.
ncpus	[integer(1)] Number of (physical) CPUs to use. If pm.backend is passed, default is determined by availableCores.

Details

The RBO Similarity for two topics $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ is calculated by

$$RBO(\boldsymbol{z}_i, \boldsymbol{z}_j \mid k, p) = 2p^k \frac{\left|Z_i^{(k)} \cap Z_j^{(k)}\right|}{\left|Z_i^{(k)}\right| + \left|Z_j^{(k)}\right|} + \frac{1-p}{p} \sum_{d=1}^{k} 2p^d \frac{\left|Z_i^{(d)} \cap Z_j^{(d)}\right|}{\left|Z_i^{(d)}\right| + \left|Z_j^{(d)}\right|}$$

with $Z_i^{(d)}$ is the vocabulary set of topic $\boldsymbol{z}_i$ down to rank d. Ties in ranks are resolved by taking the minimum.

The value `wordsconsidered` describes the number of words per topic ranked at rank k or above.

Value

[`named list`] with entries

`sims` [`lower triangular named matrix`] with all pairwise similarities of the given topics.

`wordslimit` [`integer`] = vocabulary size. See `jaccardTopics` for original purpose.

`wordsconsidered` [`integer`] = vocabulary size. See `jaccardTopics` for original purpose.

`param` [`named list`] with parameter `type` [`character(1)`] = `"RBO Similarity"`, `k` [`integer(1)`] and `p` [0,1]. See above for explanation.

References

Webber, William, Alistair Moffat and Justin Zobel (2010). "A similarity measure for indefinite rankings". In: *ACM Transations on Information Systems* 28(4), p.20:1—20:38, DOI 10.1145/1852102.1852106, URL https://doi.acm.org/10.1145/1852102.1852106

See Also

Other TopicSimilarity functions: `cosineTopics()`, `dendTopics()`, `getSimilarity()`, `jaccardTopics()`, `jsTopics()`

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
rbo = rboTopics(topics, k = 12, p = 0.9)
rbo

sim = getSimilarity(rbo)
dim(sim)
```

reuters — *A Snippet of the Reuters Dataset*

Description

Example Dataset from Reuters consisting of 91 articles. It can be used to familiarize with the bunch of functions offered by this package.

Usage

```
data(reuters_docs)

data(reuters_vocab)
```

Format

`reuters_docs` is a list of documents of length 91 prepared by `LDAprep`.

`reuters_vocab` is

An object of class `character` of length 2141.

Source

temporarily unavailable: http://ronaldo.cs.tcd.ie/esslli07/data/reuters21578-xml/

References

Lewis, David (1997). *Reuters-21578 Text Categorization Collection Distribution 1.0*. http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html

Luz, Saturnino. *XML-encoded version of Reuters-21578*. http://ronaldo.cs.tcd.ie/esslli07/data/reuters21578-xml/ (temporarily unavailable)

SCLOP — *Similarity/Stability of multiple sets of Objects using Clustering with Local Pruning*

Description

The function `SCLOP` calculates the S-CLOP value for the best possible local pruning state of a dendrogram from `dendTopics`. The function `pruneSCLOP` supplies the corresponding pruning state itself.
To get all pairwise S-CLOP scores of two LDA runs, the function `SCLOP.pairwise` can be used. It returns a matrix of the pairwise S-CLOP scores.
All three functions use the function `disparitySum` to calculate the least possible sum of disparities (on the best possible local pruning state) on a given dendrogram.

Usage

```
SCLOP(dend)

disparitySum(dend)

SCLOP.pairwise(sims)
```

Arguments

dend	[dendrogram] Output from dendTopics.
sims	[TopicSimilarity object or lower triangular named matrix] TopicSimilarity object or pairwise jaccard similarities of underlying topics as the sims element from TopicSimilarity objects. The topic names should be formatted as *<Run X>.<Topic Y>*, so that the name before the first dot identifies the LDA run.

Details

For one specific cluster g and R LDA Runs the disparity is calculated by

$$U(g) := \frac{1}{R}\sum_{r=1}^{R}|t_r^{(g)} - 1| \cdot \sum_{r=1}^{R} t_r^{(g)},$$

while $\boldsymbol{t}^{(g)} = (t_1^{(g)}, ..., t_R^{(g)})^T$ contains the number of topics that belong to the different LDA runs and that occur in cluster g.

The function disparitySum returns the least possible sum of disparities $U_\Sigma(G^*)$ for the best possible pruning state G^* with $U_\Sigma(G) = \sum_{g\in G} U(g) \to \min$. The highest possible value for $U_\Sigma(G^*)$ is limited by

$$U_{\Sigma,\text{max}} := \sum_{g\in\tilde{G}} U(g) = N \cdot \frac{R-1}{R},$$

with $\tilde{G}$ denotes the corresponding worst case pruning state. This worst case scenario is useful for normalizing the SCLOP scores.

The function SCLOP then calculates the value

$$\text{S-CLOP}(G^*) := 1 - \frac{1}{U_{\Sigma,\text{max}}} \cdot \sum_{g\in G^*} U(g) \ \in [0,1],$$

where $\sum_{g\in G^*} U(g) = U_\Sigma(G^*)$.

Value

SCLOP [0,1] value specifying the S-CLOP for the best possible local pruning state of the given dendrogram.

disparitySum [numeric(1)] value specifying the least possible sum of disparities on the given dendrogram.

SCLOP.pairwise [symmetrical named matrix] with all pairwise S-CLOP scores of the given LDA runs.

See Also

Other SCLOP functions: pruneSCLOP()

Other workflow functions: LDARep(), dendTopics(), getPrototype(), jaccardTopics(), mergeTopics()

Examples

```
res = LDARep(docs = reuters_docs, vocab = reuters_vocab, n = 4, K = 10, num.iterations = 30)
topics = mergeTopics(res, vocab = reuters_vocab)
jacc = jaccardTopics(topics, atLeast = 2)
dend = dendTopics(jacc)

SCLOP(dend)
disparitySum(dend)

SCLOP.pairwise(jacc)
SCLOP.pairwise(getSimilarity(jacc))
```

Index