Package 'koRpus'

February 2, 2021

Type Package

Title Text Analysis with Emphasis on POS Tagging, Readability, and Lexical Diversity

Description A set of tools to analyze texts. Includes, amongst others, functions for automatic language detection, hyphenation, several indices of lexical diversity (e.g., type token ratio, HD-D/vocd-D, MTLD) and readability (e.g., Flesch, SMOG, LIX, Dale-Chall). Basic import functions for language corpora are also provided, to enable frequency analyses (supports Celex and Leipzig Corpora Collection file formats) and measures like tf-idf. Note: For full functionality a local installation of TreeTagger is recommended. It is also recommended to not load this package directly, but by loading one of the available language support packages from the 'l10n' repository <https://undocumeantit.github.io/repos/l10n/>. 'koRpus' also includes a plugin for the R GUI and IDE RKWard, providing graphical dialogs for its basic features. The respective R package 'rkward' cannot be installed directly from a repository, as it is a part of RKWard. To make full use of this feature, please install RKWard from <https://rkward.kde.org> (plugins are detected automatically). Due to some restrictions on CRAN, the full package sources are only available from the project homepage. To ask for help, report bugs, request features, or discuss the development of the package, please subscribe to the koRpus-dev mailing list (<https://korpusml.reaktanz.de>).

Depends R (>= 3.0.0),sylly (>= 0.1-6)

Imports data.table,methods,Matrix

Enhances rkward

Suggests testthat,tm,SnowballC,shiny,knitr,rmarkdown,koRpus.lang.de,koRpus.lang.en,koRpus.lang.es,koRpus.lang.fr,koRpus.lang

VignetteBuilder knitr

URL https://reaktanz.de/?c=hacking&s=koRpus

BugReports https://github.com/unDocUMeantIt/koRpus/issues

Additional_repositories https://undocumeantit.github.io/repos/l10n

License GPL (>= 3)

Encoding UTF-8

LazyLoad yes

Version 0.13-5

Date 2021-02-02

RoxygenNote 7.1.1

Collate '01_class_01_kRp.text.R' '02_method_filterByClass.R'
'koRpus-internal.R' '00_environment.R' '01_class_02_kRp.TTR.R'
'01_class_03_kRp.corp.freq.R' '01_class_04_kRp.lang.R'
'01_class_05_kRp.readability.R'
'01_class_81_kRp.connection_union.R'
'02_method_get_set_kRp.text.R'
'01_class_90_deprecated_classes.R' '02_method_cTest.R'
'02_method_clozeDelete.R' '02_method_correct.R'
'02_method_docTermMatrix.R' '02_method_freq.analysis.R'
'02_method_hyphen.R' '02_method_jumbleWords.R'
'02_method_lex.div.R' '02_method_pasteText.R'
'02_method_plot.kRp.text.R' '02_method_query.R'
'02_method_read.corp.custom.R' '02_method_readTagged.R'
'02_method_readability.R' '02_method_show.kRp.lang.R'
'02_method_show.kRp.TTR.R' '02_method_show.kRp.corp.freq.R'
'02_method_show.kRp.readability.R' '02_method_show.kRp.text.R'
'02_method_split_by_doc_id.R' '02_method_summary.kRp.lang.R'
'02_method_summary.kRp.TTR.R'
'02_method_summary.kRp.readability.R'
'02_method_summary.kRp.text.R' '02_method_textTransform.R'
'02_method_tokenize.R' '02_method_treetag.R'
'02_method_types_tokens.R' 'available.koRpus.lang.R'
'get.kRp.env.R' 'guess.lang.R' 'install.koRpus.lang.R'
'kRp.POS.tags.R' 'kRp.cluster.R'
'koRpus-internal.freq.analysis.R' 'koRpus-internal.import.R'
'koRpus-internal.lexdiv.formulae.R'
'koRpus-internal.rdb.formulae.R'
'koRpus-internal.rdb.params.grades.R'
'koRpus-internal.read.corp.custom.R' 'koRpus-package.R'
'lex.div.num.R' 'read.BAWL.R' 'read.corp.LCC.R'
'read.corp.celex.R' 'readability.num.R' 'segment.optimizer.R'
'set.kRp.env.R' 'set.lang.support.R' 'textFeatures.R'
'wrapper_functions_lex.div.R' 'wrapper_functions_readability.R'

NeedsCompilation no

Author Meik Michalke [aut, cre],
Earl Brown [ctb],
Alberto Mirisola [ctb],
Alexandre Brulet [ctb],
Laura Hauser [ctb]

Maintainer Meik Michalke <meik.michalke@hhu.de>

Repository CRAN

Date/Publication 2021-02-02 16:00:02 UTC

R topics documented:

koRpus-package *Text Analysis with Emphasis on POS Tagging, Readability, and Lexical Diversity*

Description

A set of tools to analyze texts. Includes, amongst others, functions for automatic language detection, hyphenation, several indices of lexical diversity (e.g., type token ratio, HD-D/vocd-D, MTLD) and readability (e.g., Flesch, SMOG, LIX, Dale-Chall). Basic import functions for language corpora are also provided, to enable frequency analyses (supports Celex and Leipzig Corpora Collection file formats) and measures like tf-idf. Note: For full functionality a local installation of TreeTagger is recommended. It is also recommended to not load this package directly, but by loading one of the available language support packages from the 'l10n' repository <https://undocumeantit.github.io/repos/l10n/>. 'koRpus' also includes a plugin for the R GUI and IDE RKWard, providing graphical dialogs for its basic features. The respective R package 'rkward' cannot be installed directly from a repository, as it is a part of RKWard. To make full use of this feature, please install RKWard from <https://rkward.kde.org> (plugins are detected automatically). Due to some restrictions on CRAN, the full package sources are only available from the project homepage. To ask for help, report bugs, request features, or discuss the development of the package, please subscribe to the koRpus-dev mailing list (<https://korpusml.reaktanz.de>).

Details

The DESCRIPTION file:

Package:	koRpus
Type:	Package
Version:	0.13-5
Date:	2021-02-02
Depends:	R (>= 3.0.0),sylly (>= 0.1-6)
Enhances:	rkward
Encoding:	UTF-8
License:	GPL (>= 3)
LazyLoad:	yes
URL:	https://reaktanz.de/?c=hacking&s=koRpus

Author(s)

NA

Maintainer: NA

See Also

Useful links:

- https://reaktanz.de/?c=hacking&s=koRpus

- Report bugs at https://github.com/unDocUMeantIt/koRpus/issues

---

ARI *Readability: Automated Readability Index (ARI)*

Description

This is just a convenient wrapper function for readability.

Usage

```
ARI(txt.file, parameters = c(asl = 0.5, awl = 4.71, const = 21.43), ...)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for the index.
...	Further valid options for the main function, see readability for details.

Details

Calculates the Automated Readability Index (ARI). In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

If parameters="NRI", the simplified parameters from the Navy Readability Indexes are used, if set to ARI="simple", the simplified formula is calculated.

This formula doesn't need syllable count.

Value

An object of class kRp.readability.

References

DuBay, W.H. (2004). *The Principles of Readability*. Costa Mesa: Impact Information. WWW: http://www.impact-information.com/impactinfo/readability02.pdf; 22.03.2011.

Smith, E.A. & Senter, R.J. (1967). *Automated readability index*. AMRL-TR-66-22. Wright-Paterson AFB, Ohio: Aerospace Medical Division.

Examples

```
## Not run:
ARI(tagged.text)

## End(Not run)
```

`available.koRpus.lang` *List available language packages*

Description

Get a list of all currently available language packages for koRpus from the official l10n repository.

Usage

```
available.koRpus.lang(repos = "https://undocumeantit.github.io/repos/l10n/")
```

Arguments

`repos`	The URL to additional repositories to query. You should probably leave this to the default, but if you would like to use a third party repository, you're free to do so. The value is temporarily appended to the repos currently returned by `getOption("repos")`.

Details

koRpus' language support is modular by design, meaning you can (and must) load an extension package for each language you want to work with in a given session. These language support packages are named `koRpus.lang.**`, where `**` is replaced by a valid language identifier (like `en` for English or `de` for German). See `set.lang.support` for more details.

This function downloads the package list from (also) the official localization repository for koRpus and lists all currently available language packages that you could install and load. Apart from than it does not download or install anything.

You can install the packages by either calling the convenient wrapper function `install.koRpus.lang`, or `install.packages` (see examples).

Value

Returns an invisible character vector with all available language packages.

See Also

`install.koRpus.lang`

Examples

```
## Not run:
# see all available language packages
available.koRpus.lang()

# install support for German
install.koRpus.lang("de")
# alternatively, you could call install.packages directly
install.packages("koRpus.lang.de", repos="https://undocumeantit.github.io/repos/l10n/")
```

```
## End(Not run)
```

`bormuth` *Readability: Bormuth's Mean Cloze and Grade Placement*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
bormuth(txt.file, word.list, clz=35,
   meanc=c(const=0.886593, awl=0.08364, afw=0.161911,
     asl1=0.021401, asl2=0.000577, asl3=0.000005),
   grade=c(const=4.275, m1=12.881, m2=34.934, m3=20.388,
     c1=26.194, c2=2.046, c3=11.767, mc1=44.285, mc2=97.62,
     mc3=59.538), ...)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`word.list`	A vector or matrix (with exactly one column) which defines familiar words. For valid results the long Dale-Chall list with 3000 words should be used.
`clz`	Integer, the cloze criterion score in percent.
`meanc`	A numeric vector with named magic numbers, defining the relevant parameters for Mean Cloze calculation.
`grade`	A numeric vector with named magic numbers, defining the relevant parameters for Grade Placement calculation. If omitted, Grade Placement will not be calculated.
`...`	Further valid options for the main function, see `readability` for details.

Details

Calculates Bormuth's Mean Cloze and estimted grade placement. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

Examples

```
## Not run:
  bormuth(tagged.text, word.list=new.dale.chall.wl)

## End(Not run)
```

`C.ld` *Lexical diversity: Herdan's C*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
C.ld(txt, char = FALSE, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

Calculates Herdan's C. In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the C value, and characteristics are off by default.

Value

An object of class `kRp.TTR`.

See Also

`kRp.POS.tags`, `kRp.text`, `kRp.TTR`

Examples

```
## Not run:
C.ld(tagged.text)

## End(Not run)
```

clozeDelete *Transform text into cloze test format*

Description

If you feed a tagged text object to this function, its text will be transformed into a format used for cloze deletion tests. That is, by default every fifth word (or as specified by `every`) will be replaced by a line. You can also set an offset value to specify where to begin.

Usage

```
clozeDelete(obj, ...)

## S4 method for signature 'kRp.text'
clozeDelete(obj, every = 5, offset = 0, replace.by = "_", fixed = 10)
```

Arguments

`obj`	An object of class `kRp.text`.
`...`	Additional arguments to the method (as described in this document).
`every`	Integer numeric, setting the frequency of words to be manipulated. By default, every fifth word is being transformed.
`offset`	Either an integer numeric, sets the number of words to offset the transformations. Or the special keyword `"all"`, which will cause the method to iterate through all possible offset values and not return an object, but print the results (including the list with changed words).
`replace.by`	Character, will be used as the replacement for the removed words.
`fixed`	Integer numberic, defines the length of the replacement (`replace.by` will be repeated this much times). If set to 0, the replacement wil be as long as the replaced word.

Details

The option `offset="all"` will not return one single object, but print the results after iterating through all possible offset values.

Value

An object of class `kRp.text` with the added feature `diff`.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
```

```
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  tokenized.obj <- clozeDelete(tokenized.obj)
  pasteText(tokenized.obj)

  # diff stats are now part of the object
  hasFeature(tokenized.obj)
  diffText(tokenized.obj)
} else {}
```

`coleman` *Readability: Coleman's Formulas*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
coleman(
  txt.file,
  hyphen = NULL,
  parameters = c(syll = 1),
  clz1 = c(word = 1.29, const = 38.45),
  clz2 = c(word = 1.16, sntc = 1.48, const = 37.95),
  clz3 = c(word = 1.07, sntc = 1.18, pron = 0.76, const = 34.02),
  clz4 = c(word = 1.04, sntc = 1.06, pron = 0.56, prep = 0.36, const = 26.01),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`hyphen`	An object of class kRp.hyphen. If `NULL`, the text will be hyphenated automatically.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for all formulas of the index.
`clz1`	A numeric vector with named magic numbers for the first formula.
`clz2`	A numeric vector with named magic numbers for the second formula.
`clz3`	A numeric vector with named magic numbers for the third formula.
`clz4`	A numeric vector with named magic numbers for the fourth formula.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates the four readability formulas by Coleman. In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class kRp.readability.

Examples

```
## Not run:
coleman(tagged.text)

## End(Not run)
```

`coleman.liau` *Readability: Coleman-Liau Index*

Description

This is just a convenient wrapper function for readability.

Usage

```
coleman.liau(
  txt.file,
  ecp = c(const = 141.8401, char = 0.21459, sntc = 1.079812),
  grade = c(ecp = -27.4004, const = 23.06395),
  short = c(awl = 5.88, spw = 29.6, const = 15.8),
  ...
)
```

Arguments

`txt.file`	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
`ecp`	A numeric vector with named magic numbers, defining the relevant parameters for the cloze percentage estimate.
`grade`	A numeric vector with named magic numbers, defining the relevant parameters to calculate grade equvalent for ECP values.
`short`	A numeric vector with named magic numbers, defining the relevant parameters for the short form of the formula.
`...`	Further valid options for the main function, see readability for details.

Details

Calculates the Coleman-Liau index. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

Examples

```
## Not run:
coleman.liau(tagged.text)

## End(Not run)
```

`correct.tag` *Methods to correct koRpus objects*

Description

The method `correct.tag` can be used to alter objects of class `kRp.text`.

Usage

```
correct.tag(
  obj,
  row,
  tag = NULL,
  lemma = NULL,
  check.token = NULL,
  quiet = TRUE
)

## S4 method for signature 'kRp.text'
correct.tag(
  obj,
  row,
  tag = NULL,
  lemma = NULL,
  check.token = NULL,
  quiet = TRUE
)
```

Arguments

obj	An object of class kRp.text.
row	Integer, the row number of the entry to be changed. Can be an integer vector to change several rows in one go.
tag	A character string with a valid POS tag to replace the current tag entry. If NULL (the default) the entry remains unchanged.
lemma	A character string naming the lemma to to replace the current lemma entry. If NULL (the default) the entry remains unchanged.
check.token	A character string naming the token you expect to be in this row. If not NULL, correct will stop with an error if this values don't match.
quiet	If FALSE, messages about all applied changes are shown.

Details

Although automatic POS tagging and lemmatization are remarkably accurate, the algorithms do ususally produce some errors. If you want to correct for these flaws, this method can be of help, because it might prevent you from introducing new errors. That is, it will do some sanitiy checks before the object is actually manipulated and returned.

correct.tag will read the lang slot from the given object and check whether the tag provided is actually valid. If so, it will not only change the tag field in the object, but also update wclass and desc accordingly.

If check.token is set it must also match token in the given row(s). Note that no check is done on the lemmata.

Value

An object of the same class as obj.

See Also

kRp.text, treetag, kRp.POS.tags.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  tokenized.obj <- correct.tag(tokenized.obj, row=6, tag="NN")
} else {}
```

cTest *Transform text into C-Test-like format*

Description

If you feed a tagged text object to this function, its text will be transformed into a format used for C-Tests:

- the first and last sentence will be left untouched (except if the `start` and `stop` values of the `intact` parameter are changed
- of all other sentences, the second half of every 2nd word (or as specified by `every`) will be replaced by a line
- words must have at least `min.length` characters, otherwise they are skipped
- words an uneven number of characters will be replaced after the next character, i.e., a word with five characters will keep the first three and have the last two replaced

Usage

```
cTest(obj, ...)

## S4 method for signature 'kRp.text'
cTest(
  obj,
  every = 2,
  min.length = 3,
  intact = c(start = 1, end = 1),
  replace.by = "_"
)
```

Arguments

`obj`	An object of class `kRp.text`.
`...`	Additional arguments to the method (as described in this document).
`every`	Integer numeric, setting the frequency of words to be manipulated. By default, every other word is being transformed.
`min.length`	Integer numeric, sets the minimum length of words to be considered (in letters).
`intact`	Named vector with the elements `start` and `end`. both must be integer values and define, which sentences are to be left untouched, counted in sentences from beginning and end of the text. The default is to ignore the first and last sentence.
`replace.by`	Character, will be used as the replacement for the removed word halves.

Value

An object of class `kRp.text` with the added feature `diff`.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  tokenized.obj <- cTest(tokenized.obj)
  pasteText(tokenized.obj)

  # diff stats are now part of the object
  hasFeature(tokenized.obj)
  diffText(tokenized.obj)
} else {}
```

CTTR *Lexical diversity: Carroll's corrected TTR (CTTR)*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
CTTR(txt, char = FALSE, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

Calculates Carroll's corrected TTR (CTTR). In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the CTTR value, and characteristics are off by default.

Value

An object of class `kRp.TTR`.

See Also

`kRp.POS.tags`, `kRp.text`, `kRp.TTR`

Examples

```
## Not run:
CTTR(tagged.text)

## End(Not run)
```

`dale.chall` *Readability: Dale-Chall Readability Formula*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
dale.chall(
  txt.file,
  word.list,
  parameters = c(const = 64, dword = 0.95, asl = 0.69),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`word.list`	A vector or matrix (with exactly one column) which defines familiar words. For valid results the long Dale-Chall list with about 3000 words should be used.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`...`	Further valid options for the main function, see `readability` for details.

Details

Calculates the New Dale-Chall Readability Formula. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

If `parameters="PSK"`, the parameters by Powers-Sumner-Kearl (1958) are used, and if `parameters="old"`, the original parameters by Dale-Chall (1948), respectively.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

Examples

```
## Not run:
dale.chall(tagged.text, word.list=new.dale.chall.wl)

## End(Not run)
```

`danielson.bryan` *Readability: Danielson-Bryan*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
danielson.bryan(
  txt.file,
  db1 = c(cpb = 1.0364, cps = 0.0194, const = 0.6059),
  db2 = c(const = 131.059, cpb = 10.364, cps = 0.194),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`db1`	A numeric vector with named magic numbers, defining the relevant parameters for the first formula (regression).
`db2`	A numeric vector with named magic numbers, defining the relevant parameters for the second formula (cloze equivalent).
`...`	Further valid options for the main function, see `readability` for details.

Details

Calculates the two Danielson-Bryan formulas. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

Examples

```
## Not run:
  danielson.bryan(tagged.text)

## End(Not run)
```

`dickes.steiwer` *Readability: Dickes-Steiwer Handformel*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
dickes.steiwer(
  txt.file,
  parameters = c(const = 235.95993, awl = 73.021, asl = 12.56438, ttr = 50.03293),
  case.sens = FALSE,
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`case.sens`	Logical, whether types should be counted case sensitive.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates the shortcut formula by Dickes-Steiwer. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

Examples

```
## Not run:
  dickes.steiwer(tagged.text)

## End(Not run)
```

`docTermMatrix` *Generate a document-term matrix*

Description

Returns a sparse document-term matrix calculated from a given TIF[1] compliant token data frame or object of class `kRp.text`. You can also calculate the term frequency inverted document frequency value (tf-idf) for each term.

Usage

```
docTermMatrix(obj, terms = "token", case.sens = FALSE, tfidf = FALSE, ...)

## S4 method for signature 'data.frame'
docTermMatrix(obj, terms = "token", case.sens = FALSE,
      tfidf = FALSE)

## S4 method for signature 'kRp.text'
docTermMatrix(obj, terms = "token", case.sens = FALSE, tfidf = FALSE)
```

Arguments

`obj`	Either an object of class `kRp.text`, or a TIF[1] compliant token data frame.
`terms`	A character string defining the `tokens` column to be used for calculating the matrix.
`case.sens`	Logical, whether terms should be counted case sensitive.
`tfidf`	Logical, if `TRUE` calculates term frequency–inverse document frequency (tf-idf) values instead of absolute frequency.
`...`	Additional arguments depending on the particular method.

Details

This is usually more interesting if done with more than one single text. If you're interested in full corpus analysis, the `tm.plugin.koRpus` package should be worth checking out. Alternatively, a data frame with multiple `doc_id` entries can be used.

See the examples to learn how to limit the analysis to desired word classes.

Value

A sparse matrix of class `dgCMatrix`.

References

[1] Text Interchange Formats (https://github.com/ropensci/tif) [2] tm.plugin.koRpus: https://CRAN.R-project.org/package=tm.plugin.koRpus

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  # of course this makes more sense with a corpus of
  # multiple texts, see the tm.plugin.koRpus[2] package
  # for that
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  # get the document-term frequencies in a sparse matrix
  myDTMatrix <- docTermMatrix(tokenized.obj)

  # combine with filterByClass() to, e.g.,  exclude all punctuation
  myDTMatrix <- docTermMatrix(filterByClass(tokenized.obj))

  # instead of absolute frequencies, get the tf-idf values
  myDTMatrix <- docTermMatrix(
    filterByClass(tokenized.obj),
    tfidf=TRUE
  )
} else {}
```

DRP *Readability: Degrees of Reading Power (DRP)*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
DRP(txt.file, word.list, ...)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`word.list`	A vector or matrix (with exactly one column) which defines familiar words. For valid results the long Dale-Chall list with 3000 words should be used.
`...`	Further valid options for the main function, see `readability` for details.

Details

Calculates the Degrees of Reading Power, using the Bormuth Mean Cloze Score. In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

This formula doesn't need syllable count.

Value

An object of class kRp.readability.

Examples

```
## Not run:
  DRP(tagged.text, word.list=new.dale.chall.wl)

## End(Not run)
```

ELF *Readability: Fang's Easy Listening Formula (ELF)*

Description

This is just a convenient wrapper function for readability.

Usage

```
ELF(txt.file, hyphen = NULL, parameters = c(syll = 1), ...)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for the index.
...	Further valid options for the main function, see readability for details.

Details

This function calculates Fang's Easy Listening Formula (ELF). In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class kRp.readability.

References

DuBay, W.H. (2004). *The Principles of Readability*. Costa Mesa: Impact Information. WWW: http://www.impact-information.com/impactinfo/readability02.pdf; 22.03.2011.

Examples

```
## Not run:
  ELF(tagged.text)

## End(Not run)
```

`farr.jenkins.paterson` *Readability: Farr-Jenkins-Paterson Index*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
farr.jenkins.paterson(
  txt.file,
  hyphen = NULL,
  parameters = c(const = -31.517, asl = 1.015, monsy = 1.599),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`hyphen`	An object of class kRp.hyphen. If `NULL`, the text will be hyphenated automatically.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index, or `"PSK"`.
`...`	Further valid options for the main function, see `readability` for details.

Details

Calculates the Farr-Jenkins-Paterson index, a simplified version of Flesch Reading Ease. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

If `parameters="PSK"`, the revised parameters by Powers-Sumner-Kearl (1958) are used.

Value

An object of class kRp.readability.

References

Farr, J.N., Jenkins, J.J. & Paterson, D.G. (1951). Simplification of Flesch Reading Ease formula. *Journal of Applied Psychology*, 35(5), 333–337.

Powers, R.D, Sumner, W.A, & Kearl, B.E. (1958). A recalculation of four adult readability formulas, *Journal of Educational Psychology*, 49(2), 99–105.

See Also

flesch

Examples

```
## Not run:
farr.jenkins.paterson(tagged.text)

## End(Not run)
```

---

filterByClass *Remove word classes*

Description

This method strips off defined word classes of tagged text objects.

Usage

```
filterByClass(txt, ...)

## S4 method for signature 'kRp.text'
filterByClass(
  txt,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c(),
  as.vector = FALSE,
  update.desc = TRUE
)
```

Arguments

txt	An object of class kRp.text.
...	Additional options, currently unused.

corp.rm.class	A character vector with word classes which should be removed. The default value "nonpunct" has special meaning and will cause the result of kRp.POS.tags(lang,tags=c("punct to be used. Another valid value is "stopword" to remove all detected stopwords.
corp.rm.tag	A character vector with valid POS tags which should be removed.
as.vector	Logical. If TRUE, results will be returned as a character vector containing only the text parts which survived the filtering.
update.desc	Logical. If TRUE, the desc slot of the tagged object will be fully recalculated using the filtered text. If FALSE, the desc slot will be copied from the original object. Finally, if NULL, the desc slot remains empty.

Value

An object of the input class. If as.vector=TRUE, returns only a character vector.

See Also

kRp.POS.tags

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  filterByClass(tokenized.obj)
} else {}
```

---

flesch *Readability: Flesch Readability Ease*

Description

This is just a convenient wrapper function for readability.

Usage

```
flesch(
  txt.file,
  hyphen = NULL,
  parameters = c(const = 206.835, asl = 1.015, asw = 84.6),
  ...
)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.
parameters	Either a numeric vector with named magic numbers, defining the relevant parameters for the index, or a valid character string naming a preset for implemented languages ("de", "es", "es-s", "nl", "nl-b", "fr").
...	Further valid options for the main function, see readability for details.

Details

Calculates the Flesch Readability Ease index. In contrast to readability, which by default calculates all possible indices, this function will only calculate the Flesch RE value.

Certain internationalisations of the parameters are also implemented. They can be used by setting parameters to "es" (Fernandez-Huerta), "es-s" (Szigriszt), "nl" (Douma), "nl-b" (Brouwer), "de" (Amstad) or "fr" (Kandel-Moles). If parameters="PSK", the revised parameters by Powers-Sumner-Kearl (1958) are used to calculate a grade level.

Value

An object of class kRp.readability.

See Also

flesch.kincaid for grade levels, farr.jenkins.paterson for a simplified Flesch formula.

Examples

```
## Not run:
flesch(german.tagged.text, parameters="de")

## End(Not run)
```

---

flesch.kincaid *Readability: Flesch-Kincaid Grade Level*

Description

This is just a convenient wrapper function for readability.

Usage

```
flesch.kincaid(
  txt.file,
  hyphen = NULL,
  parameters = c(asl = 0.39, asw = 11.8, const = 15.59),
  ...
)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for the index.
...	Further valid options for the main function, see readability for details.

Details

Calculates the Flesch-Kincaid grade level. In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class kRp.readability.

Examples

```
## Not run:
flesch.kincaid(tagged.text)

## End(Not run)
```

FOG *Readability: Gunning FOG Index*

Description

This is just a convenient wrapper function for readability.

Usage

```
FOG(
  txt.file,
  hyphen = NULL,
  parameters = list(syll = 3, const = 0.4, suffix = c("es", "ed", "ing")),
  ...
)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.
parameters	A list with named magic numbers and a vector with verb suffixes, defining the relevant parameters for the index, or one of "PSK" or "NRI".
...	Further valid options for the main function, see readability for details.

Details

Calculates the Gunning FOG index. In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

If parameters="PSK", the revised parameters by Powers-Sumner-Kearl (1958) are used, and if parameters="NRI", the simplified parameters from the Navy Readability Indexes, respectively.

Value

An object of class kRp.readability.

References

DuBay, W.H. (2004). *The Principles of Readability*. Costa Mesa: Impact Information. WWW: http://www.impact-information.com/impactinfo/readability02.pdf; 22.03.2011.

Powers, R.D, Sumner, W.A, & Kearl, B.E. (1958). A recalculation of four adult readability formulas, *Journal of Educational Psychology*, 49(2), 99–105.

Examples

```
## Not run:
FOG(tagged.text)

## End(Not run)
```

FORCAST *Readability: FORCAST Index*

Description

This is just a convenient wrapper function for readability.

Usage

```
FORCAST(
  txt.file,
  hyphen = NULL,
  parameters = c(syll = 1, mult = 0.1, const = 20),
  ...
)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for the index, or "RGL".
...	Further valid options for the main function, see readability for details.

Details

Calculates the FORCAST index (both grade level and reading age). In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

If parameters="RGL", the parameters for the precise Reading Grade Level are used.

Value

An object of class kRp.readability.

References

Klare, G.R. (1975). Assessing readability. *Reading Research Quarterly*, 10(1), 62–102.

Examples

```
## Not run:
FORCAST(tagged.text)

## End(Not run)
```

`freq.analysis` *Analyze word frequencies*

Description

The function `freq.analysis` analyzes texts regarding frequencies of tokens, word classes etc.

Usage

```
freq.analysis(txt.file, ...)

## S4 method for signature 'kRp.text'
freq.analysis(
  txt.file,
  corp.freq = NULL,
  desc.stat = TRUE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c()
)
```

Arguments

`txt.file`	An object of class `kRp.text`.
`...`	Additional options for the generic.
`corp.freq`	An object of class `kRp.corp.freq`.
`desc.stat`	Logical, whether an updated descriptive statistical analysis should be conducted.
`corp.rm.class`	A character vector with word classes which should be ignored for frequency analysis. The default value `"nonpunct"` has special meaning and will cause the result of `kRp.POS.tags(lang,tags=c("punct","sentc"),list.classes=TRUE)` to be used.
`corp.rm.tag`	A character vector with POS tags which should be ignored for frequency analysis.

Details

It adds new columns with frequency information to the `tokens` data frame of the input data, describing how often the particular token is used in the additionally provided corpus frequency object.

To get the results, you can use `taggedText` to get the `tokens` slot, `describe` to get the raw descriptive statistics (only updated if `desc.stat=TRUE`), and `corpusFreq` to get the data from the added `freq` feature.

If `corp.freq` provides appropriate idf values for the types in `txt.file`, the term frequency–inverse document frequency statistic (tf-idf) will also be computed. Missing idf values will result in `NA`.

Value

An updated object of class kRp.text with the added feature freq, which is a list with information on the word frequencies of the analyzed text. Use corpusFreq to get that slot.

See Also

get.kRp.env, kRp.text, kRp.corp.freq

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  # call freq.analysis() on a tokenized text
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  # the token slot before frequency analysis
  head(taggedText(tokenized.obj))

  # instead of data from a larger corpus, we'll
  # use the token frequencies of the text itself
  tokenized.obj <- freq.analysis(
    tokenized.obj,
    corp.freq=read.corp.custom(tokenized.obj)
  )
  # compare the columns after the anylsis
  head(taggedText(tokenized.obj))

  # the object now has further statistics in a
  # new feature slot called freq
  hasFeature(tokenized.obj)
  corpusFreq(tokenized.obj)
} else {}
```

---

fucks *Readability: Fucks' Stilcharakteristik*

Description

This is just a convenient wrapper function for readability.

Usage

```
fucks(txt.file, ...)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
...	Further valid options for the main function, see readability for details.

Details

Calculates Fucks' Stilcharakteristik ("characteristics of style"; Fucks, 1955, as cited in Briest, 1974). In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class kRp.readability.

References

Briest, W. (1974). Kann man Verständlichkeit messen? *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 27, 543–563.

Examples

```
## Not run:
  fucks(tagged.text)

## End(Not run)
```

---

get.kRp.env *Get koRpus session settings*

Description

The function get.kRp.env returns information on your session environment regarding the koRpus package, e.g. where your local TreeTagger installation resides, if it was set before using set.kRp.env.

Usage

```
get.kRp.env(..., errorIfUnset = TRUE)
```

Arguments

...	Named parameters to get from the koRpus environment. Valid arguments are: **TT.cmd** Logical, whether the set tagger command should be returned. **lang** Logical, whether the set language should be returned. **TT.options** Logical, whether the set TT.options for `treetag` should be returned. **hyph.cache.file** Logical, whether the set hyphenation cache file for `hyphen` should be returned. **add.desc** Logical, whether tag descriptions should be added directly to tagged text objects.
`errorIfUnset`	Logical, if `TRUE` and the desired property is not set at all, the function will fail with an error message.

Details

For the most part, `get.kRp.env` is a convenient wrapper for `getOption`.

Value

A character string or list, possibly including:

`TT.cmd`	Path information for the TreeTagger command
`lang`	The specified language
`TT.options`	A list with options for `treetag`
`hyph.cache.file`	The specified hyphenation cache file for `hyphen`

See Also

`set.kRp.env`

Examples

```
set.kRp.env(lang="en")
get.kRp.env(lang=TRUE)
```

---

guess.lang *Guess language a text is written in*

Description

This function tries to guess the language a text is written in.

Usage

```
guess.lang(
  txt.file,
  udhr.path,
  comp.length = 300,
  keep.udhr = FALSE,
  quiet = TRUE,
  in.mem = TRUE,
  format = "file"
)
```

Arguments

txt.file	A character vector pointing to the file with the text to be analyzed.
udhr.path	A character string, either pointing to the directory where you unzipped the translations of the Universal Declaration of Human Rights, or to the ZIP file containing them.
comp.length	Numeric value, giving the number of characters to be used of txt to estimate the language.
keep.udhr	Logical, whether all the UDHR translations should be kept in the resulting object.
quiet	Logical. If FALSE, short status messages will be shown.
in.mem	Logical. If TRUE, the gzip compression will remain in memory (using memCompress), which is probably the faster method. Otherwise temporary files are created and automatically removed on exit.
format	Either "file" or "obj". If the latter, txt.file is not interpreted as a file path but the text to analyze itself.

Details

To accomplish the task, the method described by Benedetto, Caglioti & Loreto (2002) is used, utilizing both gzip compression and tranlations of the Universal Declaration of Human Rights[1]. The latter holds the world record for being translated into the most different languages, and is publicly available.

Value

An object of class kRp.lang.

Note

For this implementation the documents provided by the "UDHR in Unicode" project[2] have been used. Their translations are *not part of this package* and must be downloaded seperately to use guess.lang! You need the ZIP archive containing *all the plain text files* from https://unicode.org/udhr/downloads.html.

References

Benedetto, D., Caglioti, E. & Loreto, V. (2002). Language trees and zipping. *Physical Review Letters*, 88(4), 048702.

[1] http://www.ohchr.org/EN/UDHR/Pages/UDHRIndex.aspx

[2] https://unicode.org/udhr/

Examples

```
## Not run:
  # using the still zipped bulk file
  guess.lang(
    file.path("~","data","some.txt"),
    udhr.path=file.path("~","data","udhr_txt.zip")
  )
  # using the unzipped UDHR archive
  guess.lang(
    file.path("~","data","some.txt"),
    udhr.path=file.path("~","data","udhr_txt")
  )

## End(Not run)
```

---

harris.jacobson *Readability: Harris-Jacobson indices*

Description

This is just a convenient wrapper function for readability.

Usage

```
harris.jacobson(
  txt.file,
  word.list,
  parameters = c(char = 6),
  hj1 = c(dword = 0.094, asl = 0.168, const = 0.502),
  hj2 = c(dword = 0.14, asl = 0.153, const = 0.56),
  hj3 = c(asl = 0.158, lword = 0.055, const = 0.355),
  hj4 = c(dword = 0.07, asl = 0.125, lword = 0.037, const = 0.497),
  hj5 = c(dword = 0.118, asl = 0.134, lword = 0.032, const = 0.424),
  ...
)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
word.list	A vector or matrix (with exactly one column) which defines familiar words. For valid results the short Harris-Jacobson word list for grades 1 and 2 (english) should be used.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for all formulas of the index.
hj1	A numeric vector with named magic numbers for the first of the formulas.
hj2	A numeric vector with named magic numbers for the second of the formulas.
hj3	A numeric vector with named magic numbers for the third of the formulas.
hj4	A numeric vector with named magic numbers for the fourth of the formulas.
hj5	A numeric vector with named magic numbers for the fifth of the formulas.
...	Further valid options for the main function, see readability for details.

Details

This function calculates the revised Harris-Jacobson readability formulas (1 to 5), as described in their paper for the 18th Annual Meeting of the College Reading Association (Harris & Jacobson, 1974). In contrast to readability, which by default calculates all possible indices, this function will only calculate the index values.

This formula doesn't need syllable count.

Value

An object of class kRp.readability.

References

Harris, A.J. & Jacobson, M.D. (1974). Revised Harris-Jacobson readability formulas. In *18th Annual Meeting of the College Reading Association*, Bethesda.

Examples

```
## Not run:
harris.jacobson(tagged.text, word.list=harris.jacobson.wl)

## End(Not run)
```

HDD *Lexical diversity: HD-D (vocd-d)*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
HDD(txt, rand.sample = 42, char = FALSE, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`rand.sample`	An integer value, how many tokens should be assumed to be drawn for calculating HD-D.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

This function calculates HD-D, an idealized version of vocd-d (see McCarthy & Jarvis, 2007). In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the HD-D value, and characteristics are off by default.

Value

An object of class `kRp.TTR`.

References

McCarthy, P.M. & Jarvis, S. (2007). vocd: A theoretical and empirical evaluation. *Language Testing*, 24(4), 459–488.

See Also

`kRp.POS.tags`, `kRp.text`, `kRp.TTR`

Examples

```
## Not run:
HDD(tagged.text)

## End(Not run)
```

hyphen,kRp.text-method *Automatic hyphenation*

Description

These methods implement word hyphenation, based on Liang's algorithm. For details, please refer to the documentation for the generic hyphen method in the sylly package.

Usage

```
## S4 method for signature 'kRp.text'
hyphen(
  words,
  hyph.pattern = NULL,
  min.length = 4,
  rm.hyph = TRUE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c(),
  quiet = FALSE,
  cache = TRUE,
  as = "kRp.hyphen",
  as.feature = FALSE
)

## S4 method for signature 'kRp.text'
hyphen_df(
  words,
  hyph.pattern = NULL,
  min.length = 4,
  rm.hyph = TRUE,
  quiet = FALSE,
  cache = TRUE
)

## S4 method for signature 'kRp.text'
hyphen_c(
  words,
  hyph.pattern = NULL,
  min.length = 4,
  rm.hyph = TRUE,
  quiet = FALSE,
  cache = TRUE
)
```

Arguments

words	Either an object of class kRp.text, or a character vector with words to be hyphenated.
hyph.pattern	Either an object of class kRp.hyph.pat, or a valid character string naming the language of the patterns to be used. See details.
min.length	Integer, number of letters a word must have for considering a hyphenation. hyphen will not split words after the first or before the last letter, so values smaller than 4 are not useful.
rm.hyph	Logical, whether appearing hyphens in words should be removed before pattern matching.
corp.rm.class	A character vector with word classes which should be ignored. The default value "nonpunct" has special meaning and will cause the result of kRp.POS.tags(lang,tags=c("punct","se to be used. Relevant only if words is a valid koRpus object.
corp.rm.tag	A character vector with POS tags which should be ignored. Relevant only if words is a valid koRpus object.
quiet	Logical. If FALSE, short status messages will be shown.
cache	Logical. hyphen() can cache results to speed up the process. If this option is set to TRUE, the current cache will be queried and new tokens also be added. Caches are language-specific and reside in an environment, i.e., they are cleaned at the end of a session. If you want to save these for later use, see the option hyph.cache.file in set.kRp.env.
as	A character string defining the class of the object to be returned. Defaults to "kRp.hyphen", but can also be set to "data.frame" or "numeric", returning only the central data.frame or the numeric vector of counted syllables, respectively. For the latter two options, you can alternatively use the shortcut methods hyphen_df or hyphen_c. Ignored if as.feature=TRUE.
as.feature	Logical, whether the output should be just the analysis results or the input object with the results added as a feature. Use corpusHyphen to get the results from such an aggregated object. If set to TRUE, as="kRp.hyphen" is automatically set, overwriting other setting of as with a warning.

Value

An object of class kRp.text, kRp.hyphen, data.frame or a numeric vector, depending on the values of the as and as.feature arguments.

References

Liang, F.M. (1983). *Word Hy-phen-a-tion by Com-put-er*. Dissertation, Stanford University, Dept. of Computer Science.

[1] http://tug.ctan.org/tex-archive/language/hyph-utf8/tex/generic/hyph-utf8/patterns/

[2] http://www.ctan.org/tex-archive/macros/latex/base/lppl.txt

See Also

read.hyph.pat, manage.hyph.pat

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  # call hyphen on a given english word
  # "quiet=TRUE" suppresses the progress bar
  hyphen(
    "interference",
    hyph.pattern="en",
    quiet=TRUE
  )

  # call hyphen() on a tokenized text
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  # language definition is defined in the object
  # if you call hyphen() without arguments,
  # you will get its results directly
  hyphen(tokenized.obj)

  # alternatively, you can also store those results as a
  # feature in the object itself
  tokenized.obj <- hyphen(
    tokenized.obj,
    as.feature=TRUE
  )
  # results are now part of the object
  hasFeature(tokenized.obj)
  corpusHyphen(tokenized.obj)
} else {}
```

`install.koRpus.lang` *Install language support packages*

Description

This is a wrapper for `install.packages`, making it more convenient to install additional language support packages for koRpus.

Usage

```
install.koRpus.lang(
  lang,
  repos = "https://undocumeantit.github.io/repos/l10n/",
  ...
)
```

Arguments

lang	Character vector, one or more valid language identifiers (like en for English or de for German).
repos	The URL to additional repositories to query. You should probably leave this to the default, but if you would like to use a third party repository, you're free to do so. The value is temporarily appended to the repos currently returned by getOption("repos").
...	Additional options for install.packages.

Details

For a list of currently available language packages see available.koRpus.lang. See set.lang.support for more details on koRpus' language support in general.

Value

Does not return any useful objects, just calls install.packages.

See Also

install.packages, available.koRpus.lang

Examples

```
## Not run:
# install support for German
install.koRpus.lang("de")
# load the package
library("koRpus.lang.de")

## End(Not run)
```

---

jumbleWords *Produce jumbled words*

Description

This method either takes a character vector or objects inheriting class kRp.text (i.e., text tokenized by koRpus), and jumbles the words. This usually means that the first and last letter of each word is left intact, while all characters inbetween are being randomized.

Usage

```
jumbleWords(words, ...)

## S4 method for signature 'kRp.text'
jumbleWords(words, min.length = 3, intact = c(start = 1, end = 1))

## S4 method for signature 'character'
jumbleWords(words, min.length = 3, intact = c(start = 1, end = 1))
```

Arguments

words	Either a character vector or an object inheriting from class kRp.text.
...	Additional options, currently unused.
min.length	An integer value, defining the minimum word length. Words with less characters will not be changed. Grapheme clusters are counted as one.
intact	A named vector with the two integer values named start and stop. These define how many characters of each relevant words will be left unchanged at its start and its end, respectively.

Value

Depending on the class of words, either a character vector or an object of class kRp.text with the added feature diff.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  tokenized.obj <- jumbleWords(tokenized.obj)
  pasteText(tokenized.obj)

  # diff stats are now part of the object
  hasFeature(tokenized.obj)
  diffText(tokenized.obj)
} else {}
```

K.ld *Lexical diversity: Yule's K*

Description

This is just a convenient wrapper function for lex.div.

Usage

```
K.ld(txt, char = FALSE, ...)
```

Arguments

txt	An object of class kRp.text containing the tagged text to be analyzed.
char	Logical, defining whether data for plotting characteristic curves should be calculated.
...	Further valid options for the main function, see lex.div for details.

Details

This function calculates Yule's K. In contrast to lex.div, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the K value, and characteristics are off by default.

Value

An object of class kRp.TTR.

See Also

kRp.POS.tags, kRp.text, kRp.TTR

Examples

```
## Not run:
K.ld(tagged.text)

## End(Not run)
```

`koRpus-deprecated` *Deprecated object classes*

Description

These classes are no longer used by the `koRpus` package and will be removed in a later version. They are kept here for the time being so you can still load old objects and convert them into new objects using the `fixObject` method.

These functions will be removed soon and should no longer ne used.

Usage

```
kRp.filter.wclass(...)

kRp.text.paste(...)

read.tagged(...)

kRp.text.transform(...)
```

Arguments

`...`	Parameters to be passed to the replacement of the function

Slots

`lang` A character string, naming the language that is assumed for the tokenized text in this object.

`desc` Descriptive statistics of the tagged text.

`TT.res` Results of the called tokenizer and POS tagger. The data.frame usually has eleven columns:

- `doc_id`: Factor, optional document identifier.
- `token`: Character, the tokenized text.
- `tag`: Factor, POS tags for each token.
- `lemma`: Character, lemma for each token.
- `lttr`: Integer, number of letters.
- `wclass`: Factor, word class.
- `desc`: Factor, a short description of the POS tag.
- `stop`: Logical, `TRUE` if token is a stopword.
- `stem`: Character, stemmed token.
- `idx`: Integer, index number of token in this document.
- `sntc`: Integer, number of sentence in this document.

This data.frame structure adheres to the "Text Interchange Formats" guidelines set out by rOpenSci[1].

`freq.analysis` A list with information on the word frequencies of the analyzed text.

`diff` A list with mostly atomic vectors, describing the amount of diffences between both text variants (percentage):

`all.tokens`**:** Percentage of all tokens, including punctuation, that were altered.

`words`**:** Percentage of altered words only.

`all.chars`**:** Percentage of all characters, including punctuation, that were altered.

`letters`**:** Percentage of altered letters in words only.

`transfmt`**:** Character vector documenting the transformation(s) done to the tokens.

`transfmt.equal`**:** Data frame documenting which token was changed in which transformational step. Only available if more than one transformation was done.

`transfmt.normalize`**:** A list documenting steps of normalization that were done to the object, one element per transformation. Each entry holds the name of the method, the query parameters, and the effective replacement value.

`lex.div` Information on lexical diversity

S4 Class `kRp.tagged`

This was used for objects returned by `treetag` or `tokenize`. It was replaced by `kRp.text`.

S4 Class `kRp.txt.freq`

This was used for objects returned by `freq.analysis`. It was replaced by `kRp.text`.

S4 Class `kRp.txt.trans`

This was used for objects returned by `textTransform`, `clozeDelete`, `cTest`, and `jumbleWords`. It was replaced by `kRp.text`.

S4 Class `kRp.analysis`

This was used for objects returned by `kRp.text.analysis`. The function is also deprecated, functionality can be replicated by combining `treetag`,`freq.analysis` and `lex.div`.

References

[1] Text Interchange Formats (https://github.com/ropensci/tif)

`kRp.cluster` *Work in (early) progress. Probably don't even look at it. Consider it pure magic that is not to be tempered with.*

Description

In some future release, this might evolve into a function to help comparing several texts by features like average sentece length, word length, lexical diversity, and so forth. The idea behind it is to conduct a cluster analysis, to discover which texts out of several are similar to (or very different from) each other. This can be useful, e.g., if you need texts for an experiment which are different in content, but similar regarding syntactic features, like listed above.

Usage

```
kRp.cluster(txts, lang, TT.path, TT.preset)
```

Arguments

txts	A character vector with paths to texts to analyze.
lang	A character string with a valid Language identifier.
TT.path	A character string, path to TreeTagger installation.
TT.preset	A character string naming the TreeTagger preset to use.

Details

It is included in this package not really to be used, but to maybe inspire you, to toy around with the code and help me to come up with something useful in the end...

---

kRp.corp.freq,-class *S4 Class kRp.corp.freq*

Description

This class is used for objects that are returned by read.corp.LCC and read.corp.celex.

Details

The slot meta simply contains all information from the "meta.txt" of the LCC[1] data and remains empty for data from a Celex[2] DB.

Slots

meta Metadata on the corpora (see details).

words Absolute word frequencies. It has at least the following columns:

- num: Some word ID from the DB, integer
- word: The word itself
- lemma: The lemma of the word
- tag: A part-of-speech tag
- wclass: The word class
- lttr: The number of characters
- freq: The frequency of that word in the corpus DB
- pct: Percentage of appearance in DB
- pmio: Appearance per million words in DB
- log10: Base 10 logarithm of word frequency
- rank.avg: Rank in corpus data, rank ties method "average"
- rank.min: Rank in corpus data, rank ties method "min"
- rank.rel.avg: Relative rank, i.e. percentile of "rank.avg"

`rank.rel.min`: Relative rank, i.e. percentile of `"rank.min"`

`inDocs`: The absolute number of documents in the corpus containing the word

`idf`: The inverse document frequency

The slot might have additional columns, depending on the input material.

`desc` Descriptive information. It contains six numbers from the `meta` information, for convenient accessibility:

`tokens`: Number of running word forms

`types`: Number of distinct word forms

`words.p.sntc`: Average sentence length in words

`chars.p.sntc`: Average sentence length in characters

`chars.p.wform`: Average word form length

`chars.p.word`: Average running word length

The slot might have additional columns, depending on the input material.

`bigrams` A data.frame listing all tokens that co-occurred next to each other in the corpus:

`token1`: The first token

`token2`: The second token that appeared right next to the first

`freq`: How often the co-occurrance was present

`sig`: Log-likelihood significance of the co-occurrende

`cooccur` Similar to `bigrams`, but listing co-occurrences anywhere in one sentence:

`token1`: The first token

`token2`: The second token that appeared in the same sentence

`freq`: How often the co-occurrance was present

`sig`: Log-likelihood significance of the co-occurrende

`caseSens` A single logical value, whether the frequency statistics were calculated case sensitive or not.

Contructor function

Should you need to manually generate objects of this class (which should rarely be the case), the contructor function `kRp_corp_freq(...)` can be used instead of `new("kRp.corp.freq",...)`.

References

[1] https://wortschatz.uni-leipzig.de/en/download/ [2] http://celex.mpi.nl

kRp.lang,-class *S4 Class kRp.lang*

Description

This class is used for objects that are returned by `guess.lang`.

Slots

`lang` A character string, naming the language (by its ISO 639-3 identifier) that was estimated for the analized text in this object.

`lang.name` A character string, full name of the estimated language.

`txt` A character string containing the analized part of the text.

`txt.full` A character string containing the full text.

`udhr` A data.frame with full analysis results for each language tried.

Contructor function

Should you need to manually generate objects of this class (which should rarely be the case), the contructor function `kRp_lang(...)` can be used instead of `new("kRp.lang",...)`.

kRp.POS.tags *Get elaborated word tag definitions*

Description

This function can be used to get a set of part-of-speech (POS) tags for a given language. These tag sets should conform with the ones used by TreeTagger.

Usage

```
kRp.POS.tags(
  lang = get.kRp.env(lang = TRUE),
  list.classes = FALSE,
  list.tags = FALSE,
  tags = c("words", "punct", "sentc")
)
```

Arguments

`lang`	A character string defining a language (see details for valid choices).
`list.classes`	Logical, if `TRUE` only the known word classes for the chosen language will me returned.
`list.tags`	Logical, if `TRUE` only the POS tags for the chosen language will me returned.
`tags`	A character vector with at least one of "words", "punct" or "sentc".

Details

Use available.koRpus.lang to get a list of all supported languages. Language support packages must be installed an loaded to be usable with kRp.POS.tags. For the internal tokenizer a small subset of tags is also defined, available through lang="kRp". Finally, the Universal POS Tags[1] are automatically appended if no matching tag was already defined. If you don't know the language your text was written in, the function guess.lang should be able to detect it.

With the element tags you can specify if you want all tag definitions, or a subset, e.g. tags only for punctuation and sentence endings (that is, you need to call for both "punct" and "sentc" to get all punctuation tags).

The function is not so much intended to be used directly, but it is called by several other functions internally. However, it can still be useful to directly examine available POS tags.

Value

If list.classes=FALSE and list.tags=FALSE returns a matrix with word tag definitions of the given language. The matrix has three columns:

tag: Word tag

class: Respective word class

desc: "Human readable" description of what the tag stands for

Otherwise a vector with the known word classes or POS tags for the chosen language (and probably tag subset) will be returned. If both list.classes and list.tags are TRUE, still only the POS tags will be returned.

References

[1] https://universaldependencies.org/u/pos/index.html

See Also

get.kRp.env, available.koRpus.lang, install.koRpus.lang

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  tags.internal <- kRp.POS.tags("kRp")
  tags.en <- kRp.POS.tags("en")
} else {}
```

kRp.readability,-class *S4 Class kRp.readability*

Description

This class is used for objects that are returned by readability and its wrapper functions (e.g., Flesch, FOG or LIX).

Slots

lang A character string, naming the language that is assumed for the text in this object.

tokens The tokenized and POS-tagged text. See kRp.text for details.

desc Descriptive measures which were computed from the text:

- sentences: Number of sentences.
- words: Number of words.
- letters: Named vector with total number of letters ("all") and possibly several entries called "l<digit>", giving the number of words with <digit> letters.
- all.chars: Number of all characters, including spaces.
- syllables: Named vector with the number of syllables, simlar to letters, but entries are called "s<digit>" (NA if hyphenation was skipped).
- lttr.distrib: Distribution of letters: Absolute numbers, cumulative sum, inversed cumulative sum, percent, cumulative percent, and inversed cumulative percent.
- syll.distrib: Distribution of syllables (see lttr.distrib, NA if hyphenation was skipped).
- syll.uniq.distrib: Distribution of unique syllables (see lttr.distrib, NA if hyphenation was skipped).
- punct: Number of punctuation characters.
- conjunctions: Number of conjunctions.
- prepositions: Number of prepositions.
- pronouns: Number of pronouns.
- foreign: Number of foreign words.
- TTR: Type-token ratio.
- avg.sentc.length: Average number of words per sentence.
- avg.word.length: Average number of characters per word.
- avg.syll.word: Average number of syllables per word (NA if hyphenation was skipped).
- sntc.per.word: Number of sentences per word.
- sntc.per100: Number of sentences per 100 words.
- lett.per100: Number of letters per 100 words.
- syll.per100: Number of syllables per 100 words (NA if hyphenation was skipped).
- FOG.hard.words: Number of hard words, counted according to FOG (NULL if measure was not computed).
- Bormuth.NOL: Number of words not on the Bormuth word list (NULL if measure was not computed).

`Dale.Chall.NOL`: Number of words not on the Dale-Chall word list (`NULL` if measure was not computed).

`Harris.Jacobson.NOL`: Number of words not on the Harris-Jacobson word list (`NULL` if measure was not computed).

`Spache.NOL`: Number of words not on the Spache word list (`NULL` if measure was not computed).

`hyphen` The hyphenated text that was actually analyzed (i.e. without certain word classes, if they were to be removed).

`param` Relevant parameters of the given analysis, as given to the function call. See `readability` for detailed onformation.

`ARI` The "flavour" of the parameter settings and the calculated value of the ARI level. NA if not calculated.

`ARI.NRI` See "ARI".

`ARI.simple` See "ARI".

`Bormuth` The "flavour" of the parameter settings and the calculated value of Bormuth's Mean Cloze and grade level. NA if not calculated.

`Coleman` The "flavour" of the parameter settings and the calculated value of the four Coleman formulas. NA if not calculated.

`Coleman.Liau` The "flavour" of the parameter settings and the calculated value of the Coleman-Liau index. NA if not calculated.

`Dale.Chall` The "flavour" of the parameter settings and the calculated value of the Dale-Chall Readability Formula. NA if not calculated.

`Dale.Chall.PSK` See "Dale.Chall".

`Dale.Chall.old` See "Dale.Chall".

`Danielson.Bryan` The "flavour" of the parameter settings and the calculated value of the Danielson-Bryan Formula. NA if not calculated.

`Dickes.Steiwer` The "flavour" of the parameter settings and the calculated value of Dickes-Steiwer's shortcut formula. NA if not calculated.

`DRP` The "flavour" of the parameter settings and the calculated value of the Degrees of Reading Power. NA if not calculated.

`ELF` The "flavour" of the parameter settings and the calculated value of the Easy Listening Formula. NA if not calculated.

`Farr.Jenkins.Paterson` The "flavour" of the parameter settings and the calculated value of the Farr-Jenkins-Paterson index. NA if not calculated.

`Farr.Jenkins.Paterson.PSK` See "Farr.Jenkins.Paterson".

`Flesch` The "flavour" of the parameter settings and the calculated value of Flesch Reading Ease. NA if not calculated.

`Flesch.PSK` See "Flesch".

`Flesch.Brouwer` See "Flesch".

`Flesch.Szigriszt` See "Flesch".

`Flesch.de` See "Flesch".

`Flesch.es` See "Flesch".

`Flesch.fr` See "Flesch".

`Flesch.nl` See "Flesch".

`Flesch.Kincaid` The "flavour" of the parameter settings and the calculated value of the Flesch-Kincaid Grade Level. NA if not calculated.

`FOG` The "flavour" of the parameter settings, a list of proper nouns, combined words and verbs that were not counted as hard words (`"dropped"`), the considered number of hard words, and the calculated value of Gunning's FOG index. NA if not calculated.

`FOG.PSK` See "FOG".

`FOG.NRI` See "FOG".

`FORCAST` The "flavour" of the parameter settings and the calculated value of the FORCAST grade level. NA if not calculated.

`FORCAST.RGL` See "FORCAST".

`Fucks` The calculated value of Fucks' Stilcharakteristik. NA if not calculated.

`Linsear.Write` The "flavour" of the parameter settings and the calculated value of the Linsear Write index. NA if not calculated.

`LIX` The "flavour" of the parameter settings and the calculated value of the LIX index. NA if not calculated.

`RIX` The "flavour" of the parameter settings and the calculated value of the RIX index. NA if not calculated.

`SMOG` The "flavour" of the parameter settings and the calculated value of the SMOG grade level. NA if not calculated.

`SMOG.de` See "SMOG".

`SMOG.C` See "SMOG".

`SMOG.simple` See "SMOG".

`Spache` The "flavour" of the parameter settings and the calculated value of the Spache formula. NA if not calculated.

`Spache.old` See "Spache".

`Strain` The "flavour" of the parameter settings and the calculated value of the Strain index. NA if not calculated.

`Traenkle.Bailer` The "flavour" of the parameter settings, percentages of prepositions and conjunctions, and the calculated values of both Tr\"ankle-Bailer formulae. NA if not calculated.

`TRI` The calculated value of Kuntzsch' Text-Redundanz-Index. NA if not calculated.

`Tuldava` The calculated value of the Tuldava text difficulty formula. NA if not calculated.

`Wheeler.Smith` The "flavour" of the parameter settings and the calculated value of the Wheeler-Smith index. NA if not calculated.

`Wheeler.Smith.de` See "Wheeler.Smith"

`Wiener.STF` The "flavour" of the parameter settings and the calculated value of the Wiener Sachtextformel. NA if not calculated.

Contructor function

Should you need to manually generate objects of this class (which should rarely be the case), the contructor function `kRp_readability(...)` can be used instead of `new("kRp.readability",...)`.

kRp.text,-class *S4 Class kRp.text*

Description

This class is used for objects that are returned by treetag or tokenize.

Slots

lang A character string, naming the language that is assumed for the tokenized text in this object.

desc Descriptive statistics of the tagged text.

tokens Results of the called tokenizer and POS tagger. The data.frame usually has eleven columns:

doc_id: Factor, optional document identifier.

token: Character, the tokenized text.

tag: Factor, POS tags for each token.

lemma: Character, lemma for each token.

lttr: Integer, number of letters.

wclass: Factor, word class.

desc: Factor, a short description of the POS tag.

stop: Logical, TRUE if token is a stopword.

stem: Character, stemmed token.

idx: Integer, index number of token in this document.

sntc: Integer, number of sentence in this document.

This data.frame structure adheres to the "Text Interchange Formats" guidelines set out by rOpenSci[1].

features A named logical vector, indicating which features are available in this object's feat_list slot. Common features are listed in the description of the feat_list slot.

feat_list A named list with optional analysis results or other content as used by the defined features:

- hyphen A named list of objects of class kRp.hyphen.
- readability A named list of objects of class kRp.readability.
- lex_div A named list of objects of class kRp.TTR.
- freq A list with additional results of freq.analysis.
- corp_freq An object of class kRp.corp.freq, e.g., results of a call to read.corp.custom.
- diff Additional results of calls to a method like textTransform.
- doc_term_matrix A sparse document-term matrix, as produced by docTermMatrix.

See the getter and setter methods for easy access to these sub-slots. There can actually be any number of additional features, the above is just a list of those already defined by this package.

Contructor function

Should you need to manually generate objects of this class (which should rarely be the case), the contructor function `kRp_text(...)` can be used instead of `new("kRp.text",...)`.

Note

There is also `as()` methods to transform objects from other koRpus classes into kRp.text.

References

[1] Text Interchange Formats (https://github.com/ropensci/tif)

---

kRp.TTR,-class *S4 Class kRp.TTR*

Description

This class is used for objects that are returned by `lex.div` and its wrapper functions (like `TTR`, `MSTTR`, `MTLD`, etc.).

Slots

`param` Relevant parameters of the given analysis, as given to the function call, see `lex.div` for details.

`tt` The analyzed text in tokenized form, with eight elements ("tokens", "types", "lemmas", "type.in.txt", "type.in.result", "num.tokens", "num.types", "num.lemmas").

`TTR` Value of the classic type-token ratio. NA if not calculated.

`MSTTR` Mean segmental type-token ratio, including the actual "MSTTR", TTR values of each segment ("TTR.seg"), and the number of dropped words due to segment size ("dropped"). NA if not calculated.

`MATTR` Moving-average type-token ratio, including the actual "MATTR", TTR values of each window ("TTR.win"), and standard deviation of TTRs ("sd"). NA if not calculated.

`C.ld` Herdan's C. NA if not calculated.

`R.ld` Guiraud's R. NA if not calculated.

`CTTR` Carroll's CTTR. NA if not calculated.

`U.ld` Uber Index. NA if not calculated.

`S.ld` Summer's S. NA if not calculated.

`K.ld` Yule's K. NA if not calculated.

`Maas` Maas' a. NA if not calculated.

`lgV0` Maas' $\lg V_0$. NA if not calculated.

`lgeV0` Maas' $\lg_e V_0$. NA if not calculated.

`Maas.grw` Maas' relative type growth V'. NA if not calculated.

`HDD` The actual HD-D value ("HDD"), a vector with the probabilies for each type ("type.probs"), a "summary" on these probabilities and their standard deviation "sd".

`MTLD` Measure of textual lexical diversity, including the actual "MTLD", two matrices with detailed information on forward and backward factorization ("all.forw" & "all.back"), a named vector holding both calculated factors and their mean value ("factors"), and a named list with information on the number or tokens in each factor, both forward and backward, as well as their mean and standard deviation ("lengths"). NA if not calculated.

`MTLDMA` Moving-average MTLD, including the actual "MTLDMA", its standard deviation, a list ("all") with detailed information on factorization, the step size, and a named list with information on the number or tokens in each factor, as well as their mean and standard deviation ("lengths"). NA if not calculated.

`TTR.char` TTR values, starting with the first steplength of tokens, then adding the next one, progressing until the whole text is analyzed. The matrix has two colums, one for the respective step ("token") and one for the actual values ("value"). Can be used to plot TTR characteristic curves. NA if not calculated.

`MATTR.char` Equivalent to TTR.char, but calculated using MATTR algorithm. NA if not calculated.

`C.char` Equivalent to TTR.char, but calculated using Herdan's C algorithm. NA if not calculated.

`R.char` Equivalent to TTR.char, but calculated using Guiraud's R algorithm. NA if not calculated.

`CTTR.char` Equivalent to TTR.char, but calculated using Carroll's CTTR algorithm. NA if not calculated.

`U.char` Equivalent to TTR.char, but calculated using the Uber Index algorithm. NA if not calculated.

`S.char` Equivalent to TTR.char, but calculated using Summer's S algorithm. NA if not calculated.

`K.char` Equivalent to TTR.char, but calculated using Yule's K algorithm. NA if not calculated.

`Maas.char` Equivalent to TTR.char, but calculated using Maas' a algorithm. NA if not calculated.

`lgV0.char` Equivalent to TTR.char, but calculated using Maas' $\lg V_0$ algorithm. NA if not calculated.

`lgeV0.char` Equivalent to TTR.char, but calculated using Maas' $\lg_e V_0$ algorithm. NA if not calculated.

`HDD.char` Equivalent to TTR.char, but calculated using the HD-D algorithm. NA if not calculated.

`MTLD.char` Equivalent to TTR.char, but calculated using the MTLD algorithm. NA if not calculated.

`MTLDMA.char` Equivalent to TTR.char, but calculated using the moving-average MTLD algorithm. NA if not calculated.

Contructor function

Should you need to manually generate objects of this class (which should rarely be the case), the contructor function `kRp_TTR(...)` can be used instead of `new("kRp.TTR",...)`.

lex.div *Analyze lexical diversity*

Description

These methods analyze the lexical diversity/complexity of a text corpus.

Usage

```
lex.div(txt, ...)

## S4 method for signature 'kRp.text'
lex.div(
  txt,
  segment = 100,
  factor.size = 0.72,
  min.tokens = 9,
  MTLDMA.steps = 1,
  rand.sample = 42,
  window = 100,
  case.sens = FALSE,
  lemmatize = FALSE,
  detailed = FALSE,
 measure = c("TTR", "MSTTR", "MATTR", "C", "R", "CTTR", "U", "S", "K", "Maas", "HD-D",
    "MTLD", "MTLD-MA"),
 char = c("TTR", "MATTR", "C", "R", "CTTR", "U", "S", "K", "Maas", "HD-D", "MTLD",
    "MTLD-MA"),
  char.steps = 5,
  log.base = 10,
  force.lang = NULL,
  keep.tokens = FALSE,
  type.index = FALSE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c(),
  as.feature = FALSE,
  quiet = FALSE
)

## S4 method for signature 'character'
lex.div(
  txt,
  segment = 100,
  factor.size = 0.72,
  min.tokens = 9,
  MTLDMA.steps = 1,
  rand.sample = 42,
  window = 100,
```

```
  case.sens = FALSE,
  lemmatize = FALSE,
  detailed = FALSE,
 measure = c("TTR", "MSTTR", "MATTR", "C", "R", "CTTR", "U", "S", "K", "Maas", "HD-D",
    "MTLD", "MTLD-MA"),
 char = c("TTR", "MATTR", "C", "R", "CTTR", "U", "S", "K", "Maas", "HD-D", "MTLD",
    "MTLD-MA"),
  char.steps = 5,
  log.base = 10,
  force.lang = NULL,
  keep.tokens = FALSE,
  type.index = FALSE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c(),
  quiet = FALSE
)

## S4 method for signature 'missing'
lex.div(txt, measure)

## S4 method for signature 'kRp.TTR,ANY,ANY,ANY'
x[i]

## S4 method for signature 'kRp.TTR'
x[[i]]
```

Arguments

txt	An object of class kRp.text, containing the tagged text to be analyzed. If txt is of class character, it is assumed to be the raw text to be analyzed.
...	Only used for the method generic.
segment	An integer value for MSTTR, defining how many tokens should form one segment.
factor.size	A real number between 0 and 1, defining the MTLD factor size.
min.tokens	An integer value, how many tokens a full factor must at least have to be considered for the MTLD-MA result.
MTLDMA.steps	An integer value for MTLD-MA, defining the step size for the moving window, in tokens. The original proposal uses an incremet of 1. If you increase this value, computation will be faster, but your value can only remain a good estimate if the text is long enough.
rand.sample	An integer value, how many tokens should be assumed to be drawn for calculating HD-D.
window	An integer value for MATTR, defining how many tokens the moving window should include.
case.sens	Logical, whether types should be counted case sensitive.
lemmatize	Logical, whether analysis should be carried out on the lemmatized tokens rather than all running word forms.

detailed	Logical, whether full details of the analysis should be calculated. This currently affects MTLD and MTLD-MA, defining if all factors should be kept in the object. This slows down calculations considerably.
measure	A character vector defining the measures which should be calculated. Valid elements are "TTR", "MSTTR", "MATTR", "C", "R", "CTTR", "U", "S", "K", "Maas", "HD-D", "MTLD" and "MTLD-MA". You can also set it to "validation" to get information on the current status of validation.
char	A character vector defining whether data for plotting characteristic curves should be calculated. Valid elements are "TTR", "MATTR", "C", "R", "CTTR", "U", "S", "K", "Maas", "HD-D", "MTLD" and "MTLD-MA".
char.steps	An integer value defining the step size for characteristic curves, in tokens.
log.base	A numeric value defining the base of the logarithm. See log for details.
force.lang	A character string defining the language to be assumed for the text, by force. See details.
keep.tokens	Logical. If TRUE, all raw tokens and types will be preserved in the resulting object, in a slot called tt. For the types, also their frequency in the analyzed text will be listed.
type.index	Logical. If TRUE, the tt slot will contain two named lists of all types with the indices where that particular type is to be found in the original tagged text (type.in.txt) or the list of tokens in these results (type.in.result), respectively.
corp.rm.class	A character vector with word classes which should be dropped. The default value "nonpunct" has special meaning and will cause the result of kRp.POS.tags(lang,tags=c("punct to be used.
corp.rm.tag	A character vector with POS tags which should be dropped.
as.feature	Logical, whether the output should be just the analysis results or the input object with the results added as a feature. Use corpusLexDiv to get the results from such an aggregated object.
quiet	Logical. If FALSE, short status messages will be shown. TRUE will also suppress all potential warnings regarding the validation status of measures.
x	An object of class kRp.TTR.
i	Defines the row selector ([) or the name to match ([[).

Details

lex.div calculates a variety of proposed indices for lexical diversity. In the following formulae, N refers to the total number of tokens, and V to the number of types:

"TTR": The ordinary *Type-Token Ratio*:

$$TTR = \frac{V}{N}$$

Wrapper function: TTR

"MSTTR": For the *Mean Segmental Type-Token Ratio* (sometimes referred to as *Split TTR*) tokens are split up into segments of the given size, TTR for each segment is calculated and the mean

of these values returned. Tokens at the end which do not make a full segment are ignored. The number of dropped tokens is reported.

Wrapper function: MSTTR

"MATTR": The *Moving-Average Type-Token Ratio* (Covington & McFall, 2010) calculates TTRs for a defined number of tokens (called the "window"), starting at the beginning of the text and moving this window over the text, until the last token is reached. The mean of these TTRs is the MATTR.

Wrapper function: MATTR

"C": Herdan's *C* (Herdan, 1960, as cited in Tweedie & Baayen, 1998; sometimes referred to as *LogTTR*):

$$C = \frac{\lg V}{\lg N}$$

Wrapper function: C.ld

"R": Guiraud's *Root TTR* (Guiraud, 1954, as cited in Tweedie & Baayen, 1998):

$$R = \frac{V}{\sqrt{N}}$$

Wrapper function: R.ld

"CTTR": Carroll's *Corrected TTR*:

$$CTTR = \frac{V}{\sqrt{2N}}$$

Wrapper function: CTTR

"U": Dugast's *Uber Index* (Dugast, 1978, as cited in Tweedie & Baayen, 1998):

$$U = \frac{(\lg N)^2}{\lg N - \lg V}$$

Wrapper function: U.ld

"S": Summer's index:

$$S = \frac{\lg \lg V}{\lg \lg N}$$

Wrapper function: S.ld

"K": Yule's *K* (Yule, 1944, as cited in Tweedie & Baayen, 1998) is calculated by:

$$K = 10^4 \times \frac{(\sum_{X=1}^{X} f_X X^2) - N}{N^2}$$

where N is the number of tokens, X is a vector with the frequencies of each type, and f_X is the frequencies for each X.

Wrapper function: K.ld

"Maas": Maas' indices (a, $\lg V_0$ & $\lg {}_e V_0$):

$$a^2 = \frac{\lg N - \lg V}{\lg N^2}$$

$$\lg V_0 = \frac{\lg V}{\sqrt{1 - \frac{\lg V}{\lg N}^2}}$$

Earlier versions (koRpus < 0.04-12) reported a^2, and not a. The measure was derived from a formula by M\"uller (1969, as cited in Maas, 1972). $\lg_e V_0$ is equivalent to $\lg V_0$, only with e as the base for the logarithms. Also calculated are a, $\lg V_0$ (both not the same as before) and V' as measures of relative vocabulary growth while the text progresses. To calculate these measures, the first half of the text and the full text will be examined (see Maas, 1972, p. 67 ff. for details).

Wrapper function: `maas`

`"MTLD"`: For the *Measure of Textual Lexical Diversity* (McCarthy & Jarvis, 2010) so called factors are counted. Each factor is a subsequent stream of tokens which ends (and is then counted as a full factor) when the TTR value falls below the given factor size. The value of remaining partial factors is estimated by the ratio of their current TTR to the factor size threshold. The MTLD is the total number of tokens divided by the number of factors. The procedure is done twice, both forward and backward for all tokens, and the mean of both calculations is the final MTLD result.

Wrapper function: `MTLD`

`"MTLD-MA"`: The *Moving-Average Measure of Textual Lexical Diversity* (Jarvis, no year) combines factor counting and a moving window similar to MATTR: After each full factor the the next one is calculated from one token after the last starting point. This is repeated until the end of text is reached for the first time. The average of all full factor lengths is the final MTLD-MA result. Factors below the `min.tokens` threshold are dropped.

Wrapper function: `MTLD`

`"HD-D"`: The *HD-D* value can be interpreted as the idealized version of *vocd-D* (see McCarthy & Jarvis, 2007). For each type, the probability is computed (using the hypergeometric distribution) of drawing it at least one time when drawing randomly a certain number of tokens from the text – 42 by default. The sum of these probabilities make up the HD-D value. The sum of probabilities relative to the drawn sample size (ATTR) is also reported.

Wrapper function: `HDD`

By default, if the text has to be tagged yet, the language definition is queried by calling `get.kRp.env(lang=TRUE)` internally. Or, if `txt` has already been tagged, by default the language definition of that tagged object is read and used. Set `force.lang=get.kRp.env(lang=TRUE)` or to any other valid value, if you want to forcibly overwrite this default behaviour, and only then. See `kRp.POS.tags` for all supported languages.

Value

Depending on `as.feature`, either an object of class `kRp.TTR`, or an object of class `kRp.text` with the added feature `lex_div` containing it.

References

Covington, M.A. & McFall, J.D. (2010). Cutting the Gordian Knot: The Moving-Average Type-Token Ratio (MATTR). *Journal of Quantitative Linguistics*, 17(2), 94–100.

Maas, H.-D., (1972). \"Uber den Zusammenhang zwischen Wortschatzumfang und L\"ange eines Textes. *Zeitschrift f\"ur Literaturwissenschaft und Linguistik*, 2(8), 73–96.

McCarthy, P.M. & Jarvis, S. (2007). vocd: A theoretical and empirical evaluation. *Language Testing*, 24(4), 459–488.

McCarthy, P.M. & Jarvis, S. (2010). MTLD, vocd-D, and HD-D: A validation study of sophisticated approaces to lexical diversity assessment. *Behaviour Research Methods*, 42(2), 381–392.

Tweedie. F.J. & Baayen, R.H. (1998). How Variable May a Constant Be? Measures of Lexical Richness in Perspective. *Computers and the Humanities*, 32(5), 323–352.

See Also

kRp.POS.tags, kRp.text, kRp.TTR

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  # call lex.div() on a tokenized text
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  # if you call lex.div() without arguments,
  # you will get its results directly
  ld.results <- lex.div(tokenized.obj, char=c())

  # there are [ and [[ methods for these objects
  ld.results[["MSTTR"]]

  # alternatively, you can also store those results as a
  # feature in the object itself
  tokenized.obj <- lex.div(
    tokenized.obj,
    char=c(),
    as.feature=TRUE
  )
  # results are now part of the object
  hasFeature(tokenized.obj)
  corpusLexDiv(tokenized.obj)
} else {}
```

---

lex.div.num *Calculate lexical diversity*

Description

This function is a stripped down version of `lex.div`. It does not analyze text, but takes the numbers of tokens and types directly to calculate measures for which this information is sufficient:

- `"TTR"` The classic *Type-Token Ratio*
- `"C"` Herdan's *C*
- `"R"` Guiraud's *Root TTR*
- `"CTTR"` Carroll's *Corrected TTR*
- `"U"` Dugast's *Uber Index*
- `"S"` Summer's index
- `"Maas"` Maas' (a^2)

See `lex.div` for further details on the formulae.

Usage

```
lex.div.num(
  num.tokens,
  num.types,
  measure = c("TTR", "C", "R", "CTTR", "U", "S", "Maas"),
  log.base = 10,
  quiet = FALSE
)
```

Arguments

`num.tokens`	Numeric, the number of tokens.
`num.types`	Numeric, the number of types.
`measure`	A character vector defining the measures to calculate.
`log.base`	A numeric value defining the base of the logarithm. See `log` for details.
`quiet`	Logical. If `FALSE`, short status messages will be shown. `TRUE` will also suppress all potential warnings regarding the validation status of measures.

Value

An object of class `kRp.TTR`.

References

Maas, H.-D., (1972). \"Uber den Zusammenhang zwischen Wortschatzumfang und L\"ange eines Textes. *Zeitschrift f\"ur Literaturwissenschaft und Linguistik*, 2(8), 73–96.

Tweedie. F.J. & Baayen, R.H. (1998). How Variable May a Constant Be? Measures of Lexical Richness in Perspective. *Computers and the Humanities*, 32(5), 323–352.

See Also

`lex.div`

Examples

```
lex.div.num(
  num.tokens=104,
  num.types=43
)
```

`linsear.write` *Readability: Linsear Write Index*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
linsear.write(
  txt.file,
  hyphen = NULL,
  parameters = c(short.syll = 2, long.syll = 3, thrs = 20),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`hyphen`	An object of class kRp.hyphen. If `NULL`, the text will be hyphenated automatically.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates the Linsear Write index. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class `kRp.readability`.

Examples

```
## Not run:
linsear.write(tagged.text)

## End(Not run)
```

LIX *Readability: Bj\"ornsson's L\"asbarhetsindex (LIX)*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
LIX(txt.file, parameters = c(char = 6, const = 100), ...)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates the readability index ("l\"asbarhetsindex") by Bj\"ornsson. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

References

Anderson, J. (1981). Analysing the readability of english and non-english texts in the classroom with Lix. In *Annual Meeting of the Australian Reading Association*, Darwin, Australia.

Anderson, J. (1983). Lix and Rix: Variations on a little-known readability index. *Journal of Reading*, 26(6), 490–496.

Examples

```
## Not run:
  LIX(tagged.text)

## End(Not run)
```

maas *Lexical diversity: Maas' indices*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
maas(txt, char = FALSE, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

This function calculates Maas' indices (a^2 & $\lg V_0$). In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the index values, and characteristics are off by default.

Value

An object of class `kRp.TTR`.

See Also

`kRp.POS.tags`, `kRp.text`, `kRp.TTR`

Examples

```
## Not run:
maas(tagged.text)

## End(Not run)
```

MATTR *Lexical diversity: Moving-Average Type-Token Ratio (MATTR)*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
MATTR(txt, window = 100, char = FALSE, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`window`	An integer value for MATTR, defining how many tokens the moving window should include.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

This function calculates the moving-average type-token ratio (MATTR). In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the MATTR value.

Value

An object of class `kRp.TTR`.

References

Covington, M.A. & McFall, J.D. (2010). Cutting the Gordian Knot: The Moving-Average Type-Token Ratio (MATTR). *Journal of Quantitative Linguistics*, 17(2), 94–100.

See Also

`kRp.POS.tags`, `kRp.text`, `kRp.TTR`

Examples

```
## Not run:
MATTR(tagged.text)

## End(Not run)
```

MSTTR *Lexical diversity: Mean Segmental Type-Token Ratio (MSTTR)*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
MSTTR(txt, segment = 100, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`segment`	An integer value, defining how many tokens should form one segment.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

This function calculates the mean segmental type-token ratio (MSTTR). In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the MSTTR value.

Value

An object of class `kRp.TTR`.

See Also

`kRp.POS.tags`, `kRp.text`, `kRp.TTR`

Examples

```
## Not run:
MSTTR(tagged.text)

## End(Not run)
```

MTLD *Lexical diversity: Measure of Textual Lexical Diversity (MTLD)*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
MTLD(
  txt,
  factor.size = 0.72,
  min.tokens = 9,
  detailed = FALSE,
  char = FALSE,
  MA = FALSE,
  steps = 1,
  ...
)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`factor.size`	A real number between 0 and 1, defining the MTLD factor size.
`min.tokens`	An integer value, how many tokens a full factor must at least have to be considered for the MTLD-MA result.
`detailed`	Logical, whether full details of the analysis should be calculated. It defines if all factors should be kept in the object. This slows down calculations considerably.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`MA`	Logical, defining whether the newer moving-average algorithm (MTLD-MA) should be calculated.
`steps`	An integer value for MTLD-MA, defining the step size for the moving window, in tokens. The original proposal uses an incremet of 1. If you increase this value, computation will be faster, but your value can only remain a good estimate if the text is long enough.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

This function calculates the measure of textual lexical diversity (MTLD; see McCarthy & Jarvis, 2010). In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the MTLD value, and characteristics are off by default.

If you set `MA=TRUE`, the newer MTLD-MA (moving-average method) is used instead of the classic MTLD.

Value

An object of class kRp.TTR.

References

McCarthy, P. M. & Jarvis, S. (2010). MTLD, vocd-D, and HD-D: A validation study of sophisticated approaces to lexical diversity assessment. *Behaviour Research Methods*, 42(2), 381–392.

See Also

kRp.POS.tags, kRp.text, kRp.TTR

Examples

```
## Not run:
MTLD(tagged.text)

## End(Not run)
```

---

nWS *Readability: Neue Wiener Sachtextformeln*

Description

This is just a convenient wrapper function for readability.

Usage

```
nWS(
  txt.file,
  hyphen = NULL,
  parameters = c(ms.syll = 3, iw.char = 6, es.syll = 1),
  nws1 = c(ms = 19.35, sl = 0.1672, iw = 12.97, es = 3.27, const = 0.875),
  nws2 = c(ms = 20.07, sl = 0.1682, iw = 13.73, const = 2.779),
  nws3 = c(ms = 29.63, sl = 0.1905, const = 1.1144),
  nws4 = c(ms = 27.44, sl = 0.2656, const = 1.693),
  ...
)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.

parameters	A numeric vector with named magic numbers, defining the relevant parameters for all formulas of the index.
nws1	A numeric vector with named magic numbers for the first of the formulas.
nws2	A numeric vector with named magic numbers for the second of the formulas.
nws3	A numeric vector with named magic numbers for the third of the formulas.
nws4	A numeric vector with named magic numbers for the fourth of the formulas.
...	Further valid options for the main function, see readability for details.

Details

This function calculates the new Wiener Sachtextformeln (formulas 1 to 4). In contrast to readability, which by default calculates all possible indices, this function will only calculate the index values.

Value

An object of class kRp.readability.

References

Bamberger, R. & Vanecek, E. (1984). *Lesen–Verstehen–Lernen–Schreiben*. Wien: Jugend und Volk.

Examples

```
## Not run:
nWS(tagged.text)

## End(Not run)
```

---

pasteText *Paste koRpus objects*

Description

Paste the text in koRpus objects.

Usage

```
pasteText(txt, ...)

## S4 method for signature 'kRp.text'
pasteText(
  txt,
  replace = c(hon.kRp = "", hoff.kRp = "\n\n", p.kRp = "\n\n")
)
```

Arguments

txt	An object of class kRp.text.
...	Additional options, currently unused.
replace	A named character vector to define replacements for koRpus' internal headline and paragraph tags.

Details

This function takes objects of class kRp.text and pastes only the actual text as is.

Value

An atomic character vector.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  tokenized.obj <- jumbleWords(tokenized.obj)
  pasteText(tokenized.obj)
} else {}
```

---

plot *Plot method for objects of class kRp.text*

Description

Plot method for S4 objects of class kRp.text, plots the frequencies of tagged word classes.

Usage

```
plot(x, y, ...)

## S4 method for signature 'kRp.text,missing'
plot(x, what = "wclass", ...)
```

Arguments

x	An object of class kRp.text
y	From the generic plot function, ignored for koRpus class objects.
...	Any other argument suitable for plot()
what	Character string, valid options are: "wclass": Barplot of distribution of word classes "letters": Line plot of distribution of word length in letters

See Also

kRp.text

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  plot(tokenized.obj)
} else {}
```

---

query *A method to get information out of koRpus objects*

Description

The method query returns query information from objects of classes kRp.corp.freq and kRp.text.

Usage

```
query(obj, ...)

## S4 method for signature 'kRp.corp.freq'
query(
  obj,
  var = NULL,
  query,
  rel = "eq",
  as.df = TRUE,
  ignore.case = TRUE,
  perl = FALSE,
```

```
  regexp_var = "word"
)

## S4 method for signature 'kRp.text'
query(
  obj,
  var,
  query,
  rel = "eq",
  as.df = TRUE,
  ignore.case = TRUE,
  perl = FALSE,
  regexp_var = "token"
)

## S4 method for signature 'data.frame'
query(
  obj,
  var,
  query,
  rel = "eq",
  as.df = TRUE,
  ignore.case = TRUE,
  perl = FALSE,
  regexp_var = "token"
)
```

Arguments

obj	An object of class kRp.corp.freq, kRp.text, or data.frame.
...	Optional arguments, see above.
var	A character string naming a variable in the object (i.e., colname). If set to "regexp", grepl is called on the column specified by regexp_var.
query	A character vector (for words), regular expression, or single number naming values to be matched in the variable. Can also be a vector of two numbers to query a range of frequency data, or a list of named lists for multiple queries (see "Query lists" section in details).
rel	A character string defining the relation of the queried value and desired results. Must either be "eq" (equal, the default), "gt" (greater than), "ge" (greater of equal), "lt" (less than) or "le" (less or equal). If var="word", is always interpreted as "eq"
as.df	Logical, if TRUE, returns a data.frame, otherwise an object of the input class. Ignored if obj is a data frame already.
ignore.case	Logical, passed through to grepl if var="regexp".
perl	Logical, passed through to grepl if var="regexp".
regexp_var	A character string naming the column to query if var="regexp".

Details

kRp.corp.freq: Depending on the setting of the `var` parameter, will return entries with a matching character (`var="word"`), or all entries of the desired frequency (see the examples). A special case is the need for a range of frequencies, which can be achieved by providing a nomerical vector of two values as the `query` value, for start and end of the range, respectively. In these cases, if `rel` is set to `"gt"` or `"lt"`, the given range borders are excluded, otherwise they will be included as true matches.

kRp.text: `var` can be any of the variables in slot `tokens`. If `rel="num"`, a vector with the row numbers in which the query was found is returned.

Value

Depending on the arguments, might include whole objects, lists, single values etc.

Query lists

You can combine an arbitrary number of queries in a simple way by providing a list of named lists to the `query` parameter, where each list contains one query request. In each list, the first element name represents the `var` value of the request, and its value is taken as the `query` argument. You can also assign `rel`, `ignore.case` and `perl` for each request individually, and if you don't, the settings of the main query call are taken as default (`as.df` only applies to the final query). The filters will be applied in the order given, i.e., the second query will be made to the results of the first.

This method calls `subset`, which might actually be even more flexible if you need more control.

See Also

`kRp.corp.freq`, `subset`

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  en_corp <- read.corp.custom(
    tokenized.obj,
    caseSens=FALSE
  )

  # look up frequencies for the word "winner"
  query(en_corp, var="word", query="winner")

  # show all entries with a frequency of exactly 3 in the corpus
  query(en_corp, "freq", 3)
```

```
  # now, which tokens appear more than 40000 times in a million?
  query(en_corp, "pmio", 40000, "gt")

  # example for a range request: tokens with a log10 between 4.2 and 4.7
  # (including these two values)
  query(en_corp, "log10", c(4.2, 4.7))
  # (and without them)
  query(en_corp, "log10", c(4.2, 4.7), "gt")

  # example for a list of queries: get words with a frequency between
  # 10000 and 25000 per million and at least four letters
  query(en_corp, query=list(
    list(pmio=c(10000, 25000)),
    list(lttr=4, rel="ge"))
  )

  # get all instances of "the" in a tokenized text object
  query(tokenized.obj, "token", "the")
} else {}
```

`R.ld` *Lexical diversity: Guiraud's R*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
R.ld(txt, char = FALSE, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

This function calculates Guiraud's R. In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the R value, and characteristics are off by default.

Value

An object of class `kRp.TTR`.

See Also

kRp.POS.tags, kRp.text, kRp.TTR

Examples

```
## Not run:
R.ld(tagged.text)

## End(Not run)
```

---

read.BAWL *Import BAWL-R data*

Description

Read the Berlin Affective Word List – Reloaded (V\"o, Conrad, Kuchinke, Hartfeld, Hofmann & Jacobs, 2009; [1]) into a valid object of class kRp.corp.freq.

Usage

```
read.BAWL(csv, fileEncoding = NULL)
```

Arguments

csv	A character string, path to the BAWL-R in CSV2 format.
fileEncoding	A character string naming the encoding of the file, if necessary.

Details

To use this function, you must first export the BAWL-R list into CSV format: Use comma for decimal values and semicolon as value separator (often referred to as CSV2). Once you have successfully imported the word list, you can use the object to perform frequency analysis.

Value

An object of class kRp.corp.freq.

References

V\"o, M. L.-H., Conrad, M., Kuchinke, L., Hartfeld, K., Hofmann, M.F. & Jacobs, A.M. (2009). The Berlin Affective Word List Reloaded (BAWL-R). *Behavior Research Methods*, 41(2), 534–538. doi: 10.3758/BRM.41.2.534

[1] https://www.ewi-psy.fu-berlin.de/einrichtungen/arbeitsbereiche/allgpsy/Download/BAWL/index.html

See Also

kRp.corp.freq, query

Examples

```
## Not run:
bawl.corp <- read.BAWL(
  file.path("~","mydata","valence","BAWL-R.csv")
)

# you can now use query() now to create subsets of the word list,
# e.g., only nound with 5 letters and an valence rating of >= 1
bawl.stimulus <- query(bawl.corp,
  query=list(
    list(wclass="noun"),
    list(lttr=5),
    list("EMO_MEAN"=1, rel="ge")
  )
)

## End(Not run)
```

`read.corp.celex` *Import Celex data*

Description

Read data from Celex[1] formatted corpora.

Usage

```
read.corp.celex(
  celex.path,
  running.words,
  fileEncoding = "ISO_8859-1",
  n = -1,
  caseSens = TRUE
)
```

Arguments

`celex.path`	A character string, path to a frequency file in Celex format to read.
`running.words`	An integer value, number of running words in the Celex data corpus to be read.
`fileEncoding`	A character string naming the encoding of the Celex files.
`n`	An integer value defining how many lines of data should be read if `format="flatfile"`. Reads all at -1.
`caseSens`	Logical, if `FALSE` forces all frequency statistics to be calculated regardless of the tokens' case. Otherwise, if the imported database supports it, you will get different frequencies for the same tokens in different cases (e.\,g., "one" and "One").

Value

An object of class kRp.corp.freq.

References

[1] http://celex.mpi.nl

See Also

kRp.corp.freq

Examples

```
## Not run:
my.Celex.data <- read.corp.celex(
  file.path("~","mydata","Celex","GERMAN","GFW","GFW.CD"),
  running.words=5952000
)
freq.analysis(
  tokenized.obj,
  corp.freq=my.Celex.data
)

## End(Not run)
```

---

read.corp.custom *Import custom corpus data*

Description

Read data from a custom corpus into a valid object of class kRp.corp.freq.

Usage

```
read.corp.custom(corpus, caseSens = TRUE, log.base = 10, ...)

## S4 method for signature 'kRp.text'
read.corp.custom(
  corpus,
  caseSens = TRUE,
  log.base = 10,
  dtm = docTermMatrix(obj = corpus, case.sens = caseSens),
  as.feature = FALSE
)
```

Arguments

corpus	An object of class kRp.text (then the column "token" of the tokens slot is used).
caseSens	Logical. If FALSE, all tokens will be matched in their lower case form.
log.base	A numeric value defining the base of the logarithm used for inverse document frequency (idf). See log for details.
...	Additional options for methods of the generic.
dtm	A document term matrix of the corpus object as generated by docTermMatrix. This argument merely exists for cases where you want to re-use an already existing matrix. By default, it is being created from the corpus object.
as.feature	Logical, whether the output should be just the analysis results or the input object with the results added as a feature. Use corpusCorpFreq to get the results from such an aggregated object.

Details

The methods should enable you to perform a basic text corpus frequency analysis. That is, not just to import analysis results like LCC files, but to import the corpus material itself. The resulting object is of class kRp.corp.freq, so it can be used for frequency analysis by other functions and methods of this package.

Value

An object of class kRp.corp.freq.

Depending on as.feature, either an object of class kRp.corp.freq, or an object of class kRp.text with the added feature corp_freq containing it.

See Also

kRp.corp.freq

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  # call read.corp.custom() on a tokenized text
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  # if you call read.corp.custom() without arguments,
  # you will get its results directly
  en_corp <- read.corp.custom(
    tokenized.obj,
    caseSens=FALSE
```

```
  )

  # alternatively, you can also store those results as a
  # feature in the object itself
  tokenized.obj <- read.corp.custom(
    tokenized.obj,
    caseSens=FALSE,
    as.feature=TRUE
  )
  # results are now part of the object
  hasFeature(tokenized.obj)
  corpusCorpFreq(tokenized.obj)
} else {}
```

`read.corp.LCC` *Import LCC data*

Description

Read data from LCC[1] formatted corpora (Quasthoff, Richter & Biemann, 2006).

Usage

```
read.corp.LCC(
  LCC.path,
  format = "flatfile",
  fileEncoding = "UTF-8",
  n = -1,
  keep.temp = FALSE,
  prefix = NULL,
  bigrams = FALSE,
  cooccurence = FALSE,
  caseSens = TRUE
)
```

Arguments

`LCC.path`	A character string, either path to a .tar/.tar.gz/.zip file in LCC format (flatfile), or the path to the directory with the unpacked archive.
`format`	Either "flatfile" or "MySQL", depending on the type of LCC data.
`fileEncoding`	A character string naming the encoding of the LCC files. Old zip archives used "ISO_8859-1". This option will only influence the reading of meta information, as the actual database encoding is derived from there.
`n`	An integer value defining how many lines of data should be read if `format="flatfile"`. Reads all at -1.
`keep.temp`	Logical. If `LCC.path` is a tarred/zipped archive, setting `keep.temp=TRUE` will keep the temporarily unpacked files for further use. By default all temporary files will be removed when the function ends.

prefix	Character string, giving the prefix for the file names in the archive. Needed for newer LCC tar archives if they are already decompressed (autodetected if LCC.path points to the tar archive directly).
bigrams	Logical, whether infomration on bigrams should be imported. This is FALSE by default, because it might make the objects quite large. Note that this will only work in n = -1 because otherwise the tokens cannot be looked up.
cooccurence	Logical, like bigrams, but for information on co-occurences of tokens in a sentence.
caseSens	Logical, if FALSE forces all frequency statistics to be calculated regardless of the tokens' case. Otherwise, if the imported database supports it, you will get different frequencies for the same tokens in different cases (e.\,g., "one" and "One").

Details

The LCC database can either be unpacked or still a .tar/.tar.gz/.zip archive. If the latter is the case, then all necessary files will be extracted to a temporal location automatically, and by default removed again when the function has finished reading from it.

Newer LCC archives no longer feature the *-meta.txt file, resulting in less meta informtion in the object. In these cases, the total number of tokens is calculated as the sum of types' frequencies.

Value

An object of class kRp.corp.freq.

Note

Please note that MySQL support is not implemented yet.

References

Quasthoff, U., Richter, M. & Biemann, C. (2006). Corpus Portal for Search in Monolingual Corpora, In *Proceedings of the Fifth International Conference on Language Resources and Evaluation*, Genoa, 1799–1802.

[1] https://wortschatz.uni-leipzig.de/en/download/

See Also

kRp.corp.freq

Examples

```
## Not run:
# old format .zip archive
my.LCC.data <- read.corp.LCC(
  file.path("~","mydata","corpora","de05_3M.zip")
)
# new format tar archive
my.LCC.data <- read.corp.LCC(
```

```
  file.path("~","mydata","corpora","rus_web_2002_300K-text.tar")
)
# in case the tar archive was already unpacked
my.LCC.data <- read.corp.LCC(
  file.path("~","mydata","corpora","rus_web_2002_300K-text"),
  prefix="rus_web_2002_300K-"
)
freq.analysis(
  tokenized.obj,
  corp.freq=my.LCC.data
)

## End(Not run)
```

`readability` *Measure readability*

Description

These methods calculate several readability indices.

Usage

```
readability(txt.file, ...)

## S4 method for signature 'kRp.text'
readability(
  txt.file,
  hyphen = NULL,
  index = c("ARI", "Bormuth", "Coleman", "Coleman.Liau", "Dale.Chall",
   "Danielson.Bryan", "Dickes.Steiwer", "DRP", "ELF", "Farr.Jenkins.Paterson", "Flesch",
   "Flesch.Kincaid", "FOG", "FORCAST", "Fucks", "Harris.Jacobson", "Linsear.Write",
   "LIX", "nWS", "RIX", "SMOG", "Spache", "Strain", "Traenkle.Bailer", "TRI", "Tuldava",
    "Wheeler.Smith"),
  parameters = list(),
 word.lists = list(Bormuth = NULL, Dale.Chall = NULL, Harris.Jacobson = NULL, Spache =
    NULL),
  fileEncoding = "UTF-8",
  sentc.tag = "sentc",
  nonword.class = "nonpunct",
  nonword.tag = c(),
  quiet = FALSE,
  keep.input = NULL,
  as.feature = FALSE
)

## S4 method for signature 'missing'
readability(txt.file, index)
```

```
## S4 method for signature 'kRp.readability,ANY,ANY,ANY'
x[i]

## S4 method for signature 'kRp.readability'
x[[i]]
```

Arguments

txt.file	An object of class kRp.text.
...	Additional arguments for the generics.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically. All syllable handling will be skipped automatically if it's not needed for the selected indices.
index	A character vector, indicating which indices should actually be computed. If set to "all", then all available indices will be tried (meaning all variations of all measures). If set to "fast", a subset of the default values is used that is known to compute fast (currently, this only excludes "FOG"). You can also set it to "validation" to get information on the current status of validation.
parameters	A list with named magic numbers, defining the relevant parameters for each index. If none are given, the default values are used.
word.lists	A named list providing the word lists for indices which need one. If NULL or missing, the indices will be skipped and a warning is giving. Actual word lists can be provided as either a vector (or matrix or data.frame with only one column), or as a file name, where this file must contain one word per line. Alternatively, you can provide the number of words which are not on the list, directly.
fileEncoding	A character string defining the character encoding of the word.lists in case they are provided as files, like "Latin1" or "UTF-8".
sentc.tag	A character vector with POS tags which indicate a sentence ending. The default value "sentc" has special meaning and will cause the result of kRp.POS.tags(lang,tags="sentc",lis to be used.
nonword.class	A character vector with word classes which should be ignored for readability analysis. The default value "nonpunct" has special meaning and will cause the result of kRp.POS.tags(lang,tags=c("punct","sentc"),list.classes=TRUE) to be used. Will only be of consequence if hyphen is not set!
nonword.tag	A character vector with POS tags which should be ignored for readability analysis. Will only be of consequence if hyphen is not set!
quiet	Logical. If FALSE, short status messages will be shown. TRUE will also suppress all potential warnings regarding the validation status of measures.
keep.input	Logical. If FALSE, neither the object provided by (or generated from) txt.file nor hyphen will be kept in the output object. By default (NULL) they are kept if the input was not already of the needed object class (e.g., kRp.text) or missing, to allow for re-use without the need to tag or hyphenate the text again. If TRUE, they are always kept. In cases where you want smaller object sizes, set this to FALSE to always drop these slots.

as.feature	Logical, whether the output should be just the analysis results or the input object with the results added as a feature. Use corpusReadability to get the results from such an aggregated object.
x	An object of class kRp.readability.
i	Defines the row selector ([) or the name to match ([[).

Details

In the following formulae, W stands for the number of words, St for the number of sentences, C for the number of characters (usually meaning letters), Sy for the number of syllables, W_{3Sy} for the number of words with at least three syllables, $W_{<3Sy}$ for the number of words with less than three syllables, W^{1Sy} for words with exactly one syllable, W_{6C} for the number of words with at least six letters, and W_{-WL} for the number of words which are not on a certain word list (explained where needed).

"ARI": *Automated Readability Index*:

$$ARI = 0.5 \times \frac{W}{St} + 4.71 \times \frac{C}{W} - 21.43$$

If parameters is set to ARI="NRI", the revised parameters from the Navy Readability Indexes are used:

$$ARI_{NRI} = 0.4 \times \frac{W}{St} + 6 \times \frac{C}{W} - 27.4$$

If parameters is set to ARI="simple", the simplified formula is calculated:

$$ARI_{simple} = \frac{W}{St} + 9 \times \frac{C}{W}$$

Wrapper function: ARI

"Bormuth": *Bormuth Mean Cloze* & Grade Placement:

$$B_{MC} = 0.886593 - \left(0.08364 \times \frac{C}{W}\right) + 0.161911 \times \left(\frac{W_{-WL}}{W}\right)^3$$
$$-0.21401 \times \left(\frac{W}{St}\right) + 0.000577 \times \left(\frac{W}{St}\right)^2$$
$$-0.000005 \times \left(\frac{W}{St}\right)^3$$

Note: This index needs the long Dale-Chall list of 3000 familiar (english) words to compute W_{-WL}. That is, you must have a copy of this word list and provide it via the word.lists=list(Bormuth=<your.list> parameter!

$$B_{GP} = 4.275 + 12.881 \times B_{MC} - (34.934 \times B_{MC}^2) + (20.388 \times B_{MC}^3)$$
$$+(26.194C - 2.046C_{CS}^2) - (11.767C_{CS}^3) - (44.285 \times B_{MC} \times C_{CS})$$
$$+(97.620 \times (B_{MC} \times C_{CS})^2) - (59.538 \times (B_{MC} \times C_{CS})^3)$$

Where C_{CS} represents the cloze criterion score (35% by default).
Wrapper function: bormuth

`"Coleman"`: *Coleman's Readability Formulas*:

$$C_1 = 1.29 \times \left(\frac{100 \times W^{1Sy}}{W}\right) - 38.45$$

$$C_2 = 1.16 \times \left(\frac{100 \times W^{1Sy}}{W}\right) + 1.48 \times \left(\frac{100 \times St}{W}\right) - 37.95$$

$$C_3 = 1.07 \times \left(\frac{100 \times W^{1Sy}}{W}\right) + 1.18 \times \left(\frac{100 \times St}{W}\right) + 0.76 \times \left(\frac{100 \times W_{pron}}{W}\right) - 34.02$$

$$C_4 = 1.04 \times \left(\frac{100 \times W^{1Sy}}{W}\right) + 1.06 \times \left(\frac{100 \times St}{W}\right) + 0.56 \times \left(\frac{100 \times W_{pron}}{W}\right) - 0.36 \times \left(\frac{100 \times W_{prep}}{W}\right) - 26.01$$

Where W_{pron} is the number of pronouns, and W_{prep} the number of prepositions.

Wrapper function: `coleman`

`"Coleman.Liau"`: First estimates cloze percentage, then calculates grade equivalent:

$$CL_{ECP} = 141.8401 - 0.214590 \times \frac{100 \times C}{W} + 1.079812 \times \frac{100 \times St}{W}$$

$$CL_{grade} = -27.4004 \times \frac{CL_{ECP}}{100} + 23.06395$$

The short form is also calculated:

$$CL_{short} = 5.88 \times \frac{C}{W} - 29.6 \times \frac{St}{W} - 15.8$$

Wrapper function: `coleman.liau`

`"Dale.Chall"`: *New Dale-Chall Readability Formula*. By default the revised formula (1995) is calculated:

$$DC_{new} = 64 - 0.95 \times \frac{100 \times W_{-WL}}{W} - 0.69 \times \frac{W}{St}$$

This will result in a cloze score which is then looked up in a grading table. If `parameters` is set to `Dale.Chall="old"`, the original formula (1948) is used:

$$DC_{old} = 0.1579 \times \frac{100 \times W_{-WL}}{W} + 0.0496 \times \frac{W}{St} + 3.6365$$

If `parameters` is set to `Dale.Chall="PSK"`, the revised parameters by Powers-Sumner-Kearl (1958) are used:

$$DC_{PSK} = 0.1155 \times \frac{100 \times W_{-WL}}{W} + 0.0596 \times \frac{W}{St} + 3.2672$$

$$DC_PSK = 0.1155 * 100 * W\_WL/W + 0.0596 * W/St + 3.2672$$

Note: This index needs the long Dale-Chall list of 3000 familiar (english) words to compute W_{-WL}. That is, you must have a copy of this word list and provide it via the `word.lists=list(Dale.Chall=<your.li` parameter!

Wrapper function: `dale.chall`

`"Danielson.Bryan"`:

$$DB_1 = \left(1.0364 \times \frac{C}{Bl}\right) + \left(0.0194 \times \frac{C}{St}\right) - 0.6059$$

$$DB_2 = 131.059 - \left(10.364 \times \frac{C}{Bl}\right) - \left(0.194 \times \frac{C}{St}\right)$$

Where Bl means blanks between words, which is not really counted in this implementation, but estimated by $words - 1$. C is interpreted as literally all characters.

Wrapper function: `danielson.bryan`

`"Dickes.Steiwer"`: *Dickes-Steiwer Handformel*:

$$DS = 235.95993 - \left(73.021 \times \frac{C}{W}\right) - \left(12.56438 \times \frac{W}{St}\right) - (50.03293 \times TTR)$$

Where TTR refers to the type-token ratio, which will be calculated case-insensitive by default.

Wrapper function: `dickes.steiwer`

`"DRP"`: *Degrees of Reading Power*. Uses the Bormuth Mean Cloze Score:

$$DRP = (1 - B_{MC}) \times 100$$

This formula itself has no parameters. **Note:** The Bormuth index needs the long Dale-Chall list of 3000 familiar (english) words to compute W_{-WL}. That is, you must have a copy of this word list and provide it via the `word.lists=list(Bormuth=<your.list>)` parameter!
Wrapper function: `DRP`

`"ELF"`: Fang's *Easy Listening Formula*:

$$ELF = \frac{W_{2Sy}}{St}$$

Wrapper function: `ELF`

`"Farr.Jenkins.Paterson"`: A simplified version of Flesch Reading Ease:

$$FJP = -31.517 - 1.015 \times \frac{W}{St} + 1.599 \times \frac{W^{1Sy}}{W}$$

If `parameters` is set to `Farr.Jenkins.Paterson="PSK"`, the revised parameters by Powers-Sumner-Kearl (1958) are used:

$$FJP_{PSK} = 8.4335 + 0.0923 \times \frac{W}{St} - 0.0648 \times \frac{W^{1Sy}}{W}$$

Wrapper function: `farr.jenkins.paterson`

`"Flesch"`: *Flesch Reading Ease*:

$$F_{EN} = 206.835 - 1.015 \times \frac{W}{St} - 84.6 \times \frac{Sy}{W}$$

Certain internationalisations of the parameters are also implemented. They can be used by setting the `Flesch` parameter to one of the following language abbreviations.

"de" (Amstad's Verständlichkeitsindex):

$$F_{DE} = 180 - \frac{W}{St} - 58.5 \times \frac{Sy}{W}$$

"es" (Fernandez-Huerta):

$$F_{ES} = 206.835 - 1.02 \times \frac{W}{St} - 60 \times \frac{Sy}{W}$$

"es-s" (Szigriszt):

$$F_{ESS} = 206.835 - \frac{W}{St} - 62.3 \times \frac{Sy}{W}$$

"nl" (Douma):

$$F_{NL} = 206.835 - 0.93 \times \frac{W}{St} - 77 \times \frac{Sy}{W}$$

"nl-b" (Brouwer Leesindex):

$$F_{NLB} = 195 - 2 \times \frac{W}{St} - 67 \times \frac{Sy}{W}$$

"fr" (Kandel-Moles):

$$F_{FR} = 209 - 1.15 \times \frac{W}{St} - 68 \times \frac{Sy}{W}$$

If parameters is set to Flesch="PSK", the revised parameters by Powers-Sumner-Kearl (1958) are used to calculate a grade level:

$$F_{PSK} = 0.0778 \times \frac{W}{St} + 4.55 \times \frac{Sy}{W} - 2.2029$$

Wrapper function: flesch

"Flesch.Kincaid": *Flesch-Kincaid Grade Level*:

$$FK = 0.39 \times \frac{W}{St} + 11.8 \times \frac{Sy}{W} - 15.59$$

Wrapper function: flesch.kincaid

"FOG": Gunning *Frequency of Gobbledygook*:

$$FOG = 0.4 \times \left(\frac{W}{St} + \frac{100 \times W_{3Sy}}{W} \right)$$

If parameters is set to FOG="PSK", the revised parameters by Powers-Sumner-Kearl (1958) are used:

$$FOG_{PSK} = 3.0680 + \left(0.0877 \times \frac{W}{St} \right) + \left(0.0984 \times \frac{100 \times W_{3Sy}}{W} \right)$$

If parameters is set to FOG="NRI", the new FOG count from the Navy Readability Indexes is used:

$$FOG_{new} = \frac{\frac{W_{<3Sy} + (3 * W_{3Sy})}{\frac{100 \times St}{W}} - 3}{2}$$

If the text was POS-tagged accordingly, proper nouns and combinations of only easy words will not be counted as hard words, and the syllables of verbs ending in "-ed", "-es" or "-ing" will be counted without these suffixes.

Due to the need to re-hyphenate combined words after splitting them up, this formula takes considerably longer to compute than most others. If will be omitted if you set `index="fast"` instead of the default.

Wrapper function: `FOG`

`"FORCAST"`:

$$FORCAST = 20 - \frac{W^{1Sy} \times \frac{150}{W}}{10}$$

If `parameters` is set to `FORCAST="RGL"`, the parameters for the precise reading grade level are used (see Klare, 1975, pp. 84–85):

$$FORCAST_{RGL} = 20.43 - 0.11 \times W^{1Sy} \times \frac{150}{W}$$

Wrapper function: `FORCAST`

`"Fucks"`: Fucks' *Stilcharakteristik* (Fucks, 1955, as cited in Briest, 1974):

$$Fucks = \frac{Sy}{W} \times \frac{W}{St}$$

This simple formula has no parameters.

Wrapper function: `fucks`

`"Harris.Jacobson"`: *Revised Harris-Jacobson Readability Formulas* (Harris & Jacobson, 1974):

For primary-grade material:

$$HJ_1 = 0.094 \times \frac{100 \times W_{-WL}}{W} + 0.168 \times \frac{W}{St} + 0.502$$

For material above third grade:

$$HJ_2 = 0.140 \times \frac{100 \times W_{-WL}}{W} + 0.153 \times \frac{W}{St} + 0.560$$

For material below forth grade:

$$HJ_3 = 0.158 \times \frac{W}{St} + 0.055 \times \frac{100 \times W_{6C}}{W} + 0.355$$

For material below forth grade:

$$HJ_4 = 0.070 \times \frac{100 \times W_{-WL}}{W} + 0.125 \times \frac{W}{St} + 0.037 \times \frac{100 \times W_{6C}}{W} + 0.497$$

For material above third grade:

$$HJ_5 = 0.118 \times \frac{100 \times W_{-WL}}{W} + 0.134 \times \frac{W}{St} + 0.032 \times \frac{100 \times W_{6C}}{W} + 0.424$$

Note: This index needs the short Harris-Jacobson word list for grades 1 and 2 (english) to compute W_{-WL}. That is, you must have a copy of this word list and provide it via the `word.lists=list(Harris.Jacobson=<your.list>)` parameter!

Wrapper function: `harris.jacobson`

`"Linsear.Write"` **(O'Hayre, undated, see Klare, 1975, p. 85):**

$$LW_{raw} = \frac{100 - \frac{100 \times W_{<3Sy}}{W} + \left(3 \times \frac{100 \times W_{3Sy}}{W}\right)}{\frac{100 \times St}{W}}$$

$$LW(LW_{raw} \leq 20) = \frac{LW_{raw} - 2}{2}$$

$$LW(LW_{raw} > 20) = \frac{LW_{raw}}{2}$$

Wrapper function: `linsear.write`

`"LIX"` Björnsson's *Läsbarhetsindex*. Originally proposed for Swedish texts, calculated by:

$$LIX = \frac{W}{St} + \frac{100 \times W_{7C}}{W}$$

Texts with a LIX < 25 are considered very easy, around 40 normal, and > 55 very difficult to read.

Wrapper function: `LIX`

`"nWS"`**:** *Neue Wiener Sachtextformeln* (Bamberger & Vanecek, 1984):

$$nWS_1 = 19.35 \times \frac{W_{3Sy}}{W} + 0.1672 \times \frac{W}{St} + 12.97 \times \frac{W_{6C}}{W} - 3.27 \times \frac{W^{1Sy}}{W} - 0.875$$

$$nWS_2 = 20.07 \times \frac{W_{3Sy}}{W} + 0.1682 \times \frac{W}{St} + 13.73 \times \frac{W_{6C}}{W} - 2.779$$

$$nWS_3 = 29.63 \times \frac{W_{3Sy}}{W} + 0.1905 \times \frac{W}{St} - 1.1144$$

$$nWS_4 = 27.44 \times \frac{W_{3Sy}}{W} + 0.2656 \times \frac{W}{St} - 1.693$$

Wrapper function: `nWS`

`"RIX"` Anderson's *Readability Index*. A simplified version of LIX:

$$RIX = \frac{W_{7C}}{St}$$

Texts with a RIX < 1.8 are considered very easy, around 3.7 normal, and > 7.2 very difficult to read.

Wrapper function: `RIX`

`"SMOG"`**:** *Simple Measure of Gobbledygook*. By default calculates formula D by McLaughlin (1969):

$$SMOG = 1.043 \times \sqrt{W_{3Sy} \times \frac{30}{St}} + 3.1291$$

If `parameters` is set to `SMOG="C"`, formula C will be calculated:

$$SMOG_C = 0.9986 \times \sqrt{W_{3Sy} \times \frac{30}{St} + 5} + 2.8795$$

If `parameters` is set to `SMOG="simple"`, the simplified formula is used:

$$SMOG_{simple} = \sqrt{W_{3Sy} \times \frac{30}{St}} + 3$$

If `parameters` is set to `SMOG="de"`, the formula adapted to German texts ("Qu", Bamberger & Vanecek, 1984, p. 78) is used:

$$SMOG_{de} = \sqrt{W_{3Sy} \times \frac{30}{St}} - 2$$

Wrapper function: `SMOG`

`"Spache"`: *Spache Revised Formula (1974)*:

$$Spache = 0.121 \times \frac{W}{St} + 0.082 \times \frac{100 \times W_{-WL}}{W} + 0.659$$

If `parameters` is set to `Spache="old"`, the original parameters (Spache, 1953) are used:

$$Spache_{old} = 0.141 \times \frac{W}{St} + 0.086 \times \frac{100 \times W_{-WL}}{W} + 0.839$$

Note: The revised index needs the revised Spache word list (see Klare, 1975, p. 73), and the old index the short Dale-Chall list of 769 familiar (english) words to compute W_{-WL}. That is, you must have a copy of this word list and provide it via the `word.lists=list(Spache=<your.list>)` parameter!

Wrapper function: `spache`

`"Strain"`: *Strain Index*. This index was proposed in [1]:

$$S = Sy \times \frac{1}{St/3} \times \frac{1}{10}$$

Wrapper function: `strain`

`"Traenkle.Bailer"`: *Tränkle-Bailer Formeln*. These two formulas were the result of a re-examination of the ones proposed by Dickes-Steiwer. They try to avoid the usage of the type-token ratio, which is dependent on text length (Tränkle & Bailer, 1984):

$$TB1 = 224.6814 - \left(79.8304 \times \frac{C}{W}\right) - \left(12.24032 \times \frac{W}{St}\right) - \left(1.292857 \times \frac{100 \times W_{prep}}{W}\right)$$

$$TB2 = 234.1063 - \left(96.11069 \times \frac{C}{W}\right) - \left(2.05444 \times \frac{100 \times W_{prep}}{W}\right) - \left(1.02805 \times \frac{100 \times W_{conj}}{W}\right)$$

Where W_{prep} refers to the number of prepositions, and W_{conj} to the number of conjunctions.

Wrapper function: `traenkle.bailer`

`"TRI"`: Kuntzsch's *Text-Redundanz-Index*. Intended mainly for German newspaper comments.

$$TRI = \left(0.449 \times W^{1Sy}\right) - (2.467 \times Ptn) - (0.937 \times Frg) - 14.417$$

Where Ptn is the number of punctuation marks and Frg the number of foreign words.

Wrapper function: `TRI`

`"Tuldava"`: Tuldava's *Text Difficulty Formula*. Supposed to be rather independent of specific languages (Grzybek, 2010).

$$TD = \frac{Sy}{W} \times ln\left(\frac{W}{St}\right)$$

Wrapper function: `tuldava`

`"Wheeler.Smith"`: Intended for english texts in primary grades 1–4 (Wheeler & Smith, 1954):

$$WS = \frac{W}{St} \times \frac{10 \times W_{2Sy}}{W}$$

If `parameters` is set to `Wheeler.Smith="de"`, the calculation stays the same, but grade placement is done according to Bamberger & Vanecek (1984), that is for german texts.

Wrapper function: `wheeler.smith`

By default, if the text has to be tagged yet, the language definition is queried by calling `get.kRp.env(lang=TRUE)` internally. Or, if `txt` has already been tagged, by default the language definition of that tagged object is read and used. Set `force.lang=get.kRp.env(lang=TRUE)` or to any other valid value, if you want to forcibly overwrite this default behaviour, and only then. See `kRp.POS.tags` for all supported languages.

Value

Depending on `as.feature`, either an object of class `kRp.readability`, or an object of class `kRp.text` with the added feature `readability` containing it.

Note

To get a printout of the default parameters like they're set if no other parameters are specified, call `readability(parameters="dput")`. In case you want to provide different parameters, you must provide a complete set for an index, or special parameters that are mentioned in the index descriptions above (e.g., "PSK", if appropriate).

References

Anderson, J. (1981). Analysing the readability of english and non-english texts in the classroom with Lix. In *Annual Meeting of the Australian Reading Association*, Darwin, Australia.

Anderson, J. (1983). Lix and Rix: Variations on a little-known readability index. *Journal of Reading*, 26(6), 490–496.

Bamberger, R. & Vanecek, E. (1984). *Lesen–Verstehen–Lernen–Schreiben*. Wien: Jugend und Volk.

Briest, W. (1974). Kann man Verständlichkeit messen? *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung*, 27, 543–563.

Coleman, M. & Liau, T.L. (1975). A computer readability formula designed for machine scoring, *Journal of Applied Psychology*, 60(2), 283–284.

Dickes, P. & Steiwer, L. (1977). Ausarbeitung von Lesbarkeitsformeln für die deutsche Sprache. *Zeitschrift für Entwicklungspsychologie und Pädagogische Psychologie*, 9(1), 20–28.

DuBay, W.H. (2004). *The Principles of Readability*. Costa Mesa: Impact Information. WWW: http://www.impact-information.com/impactinfo/readability02.pdf; 22.03.2011.

Farr, J.N., Jenkins, J.J. & Paterson, D.G. (1951). Simplification of Flesch Reading Ease formula. *Journal of Applied Psychology*, 35(5), 333–337.

Flesch, R. (1948). A new readability yardstick. *Journal of Applied Psychology*, 32(3), 221–233.

Grzybek, P. (2010). Text difficulty and the Arens-Altmann law. In Peter Grzybek, Emmerich Kelih, Ján Mačutek (Eds.), *Text and Language. Structures – Functions – Interrelations. Quantitative Perspectives*. Wien: Praesens, 57–70.

Harris, A.J. & Jacobson, M.D. (1974). Revised Harris-Jacobson readability formulas. In *18th Annual Meeting of the College Reading Association*, Bethesda.

Klare, G.R. (1975). Assessing readability. *Reading Research Quarterly*, 10(1), 62–102.

McLaughlin, G.H. (1969). SMOG grading – A new readability formula. *Journal of Reading*, 12(8), 639–646.

Powers, R.D, Sumner, W.A, & Kearl, B.E. (1958). A recalculation of four adult readability formulas, *Journal of Educational Psychology*, 49(2), 99–105.

Smith, E.A. & Senter, R.J. (1967). *Automated readability index*. AMRL-TR-66-22. Wright-Paterson AFB, Ohio: Aerospace Medical Division.

Spache, G. (1953). A new readability formula for primary-grade reading materials. *The Elementary School Journal*, 53, 410–413.

Tränkle, U. & Bailer, H. (1984). Kreuzvalidierung und Neuberechnung von Lesbarkeitsformeln für die deutsche Sprache. *Zeitschrift für Entwicklungspsychologie und Pädagogische Psychologie*, 16(3), 231–244.

Wheeler, L.R. & Smith, E.H. (1954). A practical readability formula for the classroom teacher in the primary grades. *Elementary English*, 31, 397–399.

[1] https://strainindex.wordpress.com/2007/09/25/hello-world/

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  # call readability() on a tokenized text
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  # if you call readability() without arguments,
  # you will get its results directly
  rdb.results <- readability(tokenized.obj)

  # there are [ and [[ methods for these objects
  rdb.results[["ARI"]]

  # alternatively, you can also store those results as a
  # feature in the object itself
  tokenized.obj <- readability(
    tokenized.obj,
```

```
    as.feature=TRUE
  )
  # results are now part of the object
  hasFeature(tokenized.obj)
  corpusReadability(tokenized.obj)
} else {}
```

`readability.num` *Calculate readability*

Description

This function is a stripped down version of `readability`. It does not analyze text, but directly takes the values used by the formulae to calculate the readability measures.

Usage

```
readability.num(
 txt.features = list(sentences = NULL, words = NULL, letters = c(all = 0, l5 = 0, l6 =
    0), syllables = c(all = 0, s1 = 0, s2 = 0), punct = NULL, all.chars = NULL,
  prepositions = NULL, conjunctions = NULL, pronouns = NULL, foreign = NULL, TTR =
    NULL, FOG.hard.words = NULL, Bormuth.NOL = NULL, Dale.Chall.NOL = NULL,
    Harris.Jacobson.NOL = NULL, Spache.NOL = NULL, lang = character()),
 index = c("ARI", "Bormuth", "Coleman", "Coleman.Liau", "Dale.Chall",
  "Danielson.Bryan", "Dickes.Steiwer", "DRP", "ELF", "Farr.Jenkins.Paterson", "Flesch",
  "Flesch.Kincaid", "FOG", "FORCAST", "Fucks", "Harris.Jacobson", "Linsear.Write",
  "LIX", "nWS", "RIX", "SMOG", "Spache", "Strain", "Traenkle.Bailer", "TRI", "Tuldava",
    "Wheeler.Smith"),
 parameters = list()
)
```

Arguments

`txt.features` A named list with statistical information on the text, or an object of class `kRp.readability` (only its `desc` slot will then be used). Valid values are:

`sentences`**:** The number of sentences.

`words`**:** The number of words.

`letters`**:** A named vector providing the number of letters. Must contain a value called `"all"`, the total number of letters, and several values called `"l<digit>"`, giving the number of words with `<digit>` letters. To calculate all implemented measures with default parameters, you need at least the values `"l5"` (words with five *or less* letters) and `"l6"` (words with six letters).

`syllables`**:** Similar to `letters`, but providing the number of syllables. Must contain a value called `"all"`, the total number of syllables, and several values called `"s<digit>"`, giving the number of words with `<digit>` syllables. To calculate all implemented measures with default parameters, you

need at least the values "s1" and "s2". Only needed to calculate measures which need syllable count (see readability).

punct: The number of punctuation characters. Only needed to calculate "TRI".

all.chars: The number of all characters (including spaces). Only needed to calculate Danielson.Bryan.

prepositions: The number of prepositions. Only needed to calculate "Coleman" and "Traenkle.Bailer".

conjunctions: The number of conjunctions. Only needed to calculate "Traenkle.Bailer".

pronouns: The number of pronouns. Only needed to calculate "Coleman".

foreign: The number of foreign words. Only needed to calculate "TRI".

TTR: The type-token ratio. Only needed to calculate "Dickes.Steiwer".

FOG.hard.words: The number of hard words, counted according to FOG. Only needed to calculate "FOG".

Bormuth.NOL: Number of words not on the Bormuth word list. Only needed to calculate "Bormuth".

Dale.Chall.NOL: Number of words not on the Dale-Chall word list. Only needed to calculate "Dale.Chall".

Harris.Jacobson.NOL: Number of words not on the Harris-Jacobson word list. Only needed to calculate "Harris.Jacobson".

Spache.NOL: Number of words not on the Spache word list. Only needed to calculate "Spache".

lang: A character string defining the language, if needed.

index — A character vector, indicating which indices should actually be computed.

parameters — A named list with magic numbers, defining the relevant parameters for each index. If none are given, the default values are used.

Examples

```
## Not run:
test.features <- list(
  sentences=18,
  words=556,
  letters=c(
    all=2918,
    l1=19,
    l2=92,
    l3=74,
    l4=80,
    l5=51,
    l6=49
  ),
  syllables=c(
    all=974,
    s1=316,
    s2=116
  ),
  punct=78,
  all.chars=3553,
```

```
  prepositions=74,
  conjunctions=18,
  pronouns=9,
  foreign=0,
  TTR=0.5269784,
  Bormuth.NOL=192,
  Dale.Chall.NOL=192,
  Harris.Jacobson.NOL=240,
  Spache.NOL=240,
  lang="en"
)

# should not calculate FOG, because FOG.hard.words is missing:
readability.num(test.features, index="all")

## End(Not run)
```

---

readTagged *Import already tagged texts*

Description

This method can be used on text files or matrices containing already tagged text material, e.g. the results of TreeTagger[1].

Usage

```
readTagged(file, ...)

## S4 method for signature 'matrix'
readTagged(
  file,
  lang = "kRp.env",
  tagger = "TreeTagger",
  apply.sentc.end = TRUE,
  sentc.end = c(".", "!", "?", ";", ":"),
  stopwords = NULL,
  stemmer = NULL,
  rm.sgml = TRUE,
  doc_id = NA,
  add.desc = "kRp.env",
  mtx_cols = c(token = "token", tag = "tag", lemma = "lemma")
)

## S4 method for signature 'data.frame'
readTagged(
  file,
  lang = "kRp.env",
```

```
  tagger = "TreeTagger",
  apply.sentc.end = TRUE,
  sentc.end = c(".", "!", "?", ";", ":"),
  stopwords = NULL,
  stemmer = NULL,
  rm.sgml = TRUE,
  doc_id = NA,
  add.desc = "kRp.env",
  mtx_cols = c(token = "token", tag = "tag", lemma = "lemma")
)

## S4 method for signature 'kRp.connection'
readTagged(
  file,
  lang = "kRp.env",
  encoding = "unknown",
  tagger = "TreeTagger",
  apply.sentc.end = TRUE,
  sentc.end = c(".", "!", "?", ";", ":"),
  stopwords = NULL,
  stemmer = NULL,
  rm.sgml = TRUE,
  doc_id = NA,
  add.desc = "kRp.env"
)

## S4 method for signature 'character'
readTagged(
  file,
  lang = "kRp.env",
  encoding = getOption("encoding"),
  tagger = "TreeTagger",
  apply.sentc.end = TRUE,
  sentc.end = c(".", "!", "?", ";", ":"),
  stopwords = NULL,
  stemmer = NULL,
  rm.sgml = TRUE,
  doc_id = NA,
  add.desc = "kRp.env"
)
```

Arguments

file	Either a matrix, a connection or a character vector. If the latter, that must be a valid path to a file, containing the previously analyzed text. If it is a matrix, it must contain three columns named "token", "tag", and "lemma", and except for these three columns all others are ignored.
...	Additional options, currently unused.

lang	A character string naming the language of the analyzed corpus. See kRp.POS.tags for all supported languages. If set to "kRp.env" this is got from get.kRp.env.
tagger	The software which was used to tokenize and tag the text. Currently, "TreeTagger" and "manual" are the only supported values. If "manual", you must also adjust the values of mtx_cols to define the columns to be imported.
apply.sentc.end	Logical, whethter the tokens defined in sentc.end should be searched and set to a sentence ending tag. You could call this a compatibility mode to make sure you get the results you would get if you called treetag on the original file. If set to FALSE, the tags will be imported as they are.
sentc.end	A character vector with tokens indicating a sentence ending. This adds to given results, it doesn't replace them.
stopwords	A character vector to be used for stopword detection. Comparison is done in lower case. You can also simply set stopwords=tm::stopwords("en") to use the english stopwords provided by the tm package.
stemmer	A function or method to perform stemming. For instance, you can set stemmer=Snowball::SnowballSte if you have the Snowball package installed (or SnowballC::wordStem). As of now, you cannot provide further arguments to this function.
rm.sgml	Logical, whether SGML tags should be ignored and removed from output.
doc_id	Character string, optional identifier of the particular document. Will be added to the desc slot.
add.desc	Logical. If TRUE, the tag description (column "desc" of the data.frame) will be added directly to the resulting object. If set to "kRp.env" this is fetched from get.kRp.env. Only needed if tag=TRUE.
mtx_cols	Character vector with exactly three elements named "token", "tag", and "lemma", the values of which must match the respective column names of the matrix provided via file. It is possible to set lemma=NA if the tagged results only provide token and tag. This argument is ignored unless tagger="manual" and data is provided as either a matrix or data frame.
encoding	A character string defining the character encoding of the input file, like "Latin1" or "UTF-8".

Details

Note that the value of lang must match a valid language supported by kRp.POS.tags. It will also get stored in the resulting object and might be used by other functions at a later point.

Value

An object of class kRp.text. If debug=TRUE, prints internal variable settings and attempts to return the original output if the TreeTagger system call in a matrix.

References

Schmid, H. (1994). Probabilistic part-of-speec tagging using decision trees. In *International Conference on New Methods in Language Processing*, Manchester, UK, 44–49.

[1] https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

See Also

treetag, freq.analysis, get.kRp.env, kRp.text

Examples

```
## Not run:
  # call method on a connection
  text_con <- file("~/my.data/tagged_speech.txt", "r")
  tagged_results <- readTagged(text_con, lang="en")
  close(text_con)

  # call it on the file directly
  tagged_results <- readTagged("~/my.data/tagged_speech.txt", lang="en")

  # import the results of RDRPOSTagger, using the "manual" tagger feature
  sample_text <- c("Dies ist ein kurzes Beispiel. Es ergibt wenig Sinn.")
  tagger <- RDRPOSTagger::rdr_model(language="German", annotation="POS")
  tagged_rdr <- RDRPOSTagger::rdr_pos(tagger, x=sample_text)
  tagged_results <- readTagged(
    tagged_rdr,
    lang="de",
    tagger="manual",
    mtx_cols=c(token="token", tag="pos", lemma=NA)
  )

## End(Not run)
```

RIX *Readability: Anderson's Readability Index (RIX)*

Description

This is just a convenient wrapper function for readability.

Usage

```
RIX(txt.file, parameters = c(char = 6), ...)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for the index.
...	Further valid options for the main function, see readability for details.

Details

This function calculates the Readability Index (RIX) by Anderson, which is a simplified version of the l\"asbarhetsindex (LIX) by Bj\"ornsson. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

References

Anderson, J. (1981). Analysing the readability of english and non-english texts in the classroom with Lix. In *Annual Meeting of the Australian Reading Association*, Darwin, Australia.

Anderson, J. (1983). Lix and Rix: Variations on a little-known readability index. *Journal of Reading*, 26(6), 490–496.

Examples

```
## Not run:
  RIX(tagged.text)

## End(Not run)
```

S.ld *Lexical diversity: Summer's S*

Description

This is just a convenient wrapper function for `lex.div`.

Usage

```
S.ld(txt, char = FALSE, ...)
```

Arguments

`txt`	An object of class `kRp.text` containing the tagged text to be analyzed.
`char`	Logical, defining whether data for plotting characteristic curves should be calculated.
`...`	Further valid options for the main function, see `lex.div` for details.

Details

This function calculates Summer's S. In contrast to `lex.div`, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the S value, and characteristics are off by default.

Value

An object of class kRp.TTR.

See Also

kRp.POS.tags, kRp.text, kRp.TTR

Examples

```
## Not run:
S.ld(tagged.text)

## End(Not run)
```

---

segment.optimizer *A function to optimize MSTTR segment sizes*

Description

This function calculates an optimized segment size for MSTTR.

Usage

```
segment.optimizer(txtlgth, segment = 100, range = 20, favour.min = TRUE)
```

Arguments

txtlgth	Integer value, size of text in tokens.
segment	Integer value, start value of the segment size.
range	Integer value, range around segment to search for better fitting sizes.
favour.min	Logical, whether as a last ressort smaller or larger segment sizes should be preferred, if in doubt.

Details

When calculating the mean segmental type-token ratio (MSTTR), tokens are divided into segments of a given size and analyzed. If at the end text is left over which won't fill another full segment, it is discarded, i.e. information is lost. For interpretation it is debatable which is worse: Dropping more or less actual token material, or variance in segment size between analyzed texts. If you'd prefer the latter, this function might prove helpful.

Starting with a given text length, segment size and range to investigate, segment.optimizer iterates through possible segment values. It returns the segment size which would drop the fewest tokens (zero, if you're lucky). Should more than one value fulfill this demand, the one nearest to the segment start value is taken. In cases, where still two values are equally far away from the start value, it depends on the setting of favour.min if the smaller or larger segment size is returned.

Value

A numeric vector with two elements:

seg	The optimized segment size
drop	The number of tokens that would be dropped using this segment size

See Also

lex.div, MSTTR

Examples

```
segment.optimizer(2014, favour.min=FALSE)
```

---

set.kRp.env *A function to set information on your koRpus environment*

Description

The function set.kRp.env can be called before any of the analysing functions. It writes information on your session environment regarding the koRpus package, e.g. path to a local TreeTagger installation, to your global .Options.

Usage

```
set.kRp.env(..., validate = TRUE)
```

Arguments

...	Named parameters to set in the koRpus environment. Valid arguments are: **TT.cmd** A character string pointing to the tagger command you want to use for basic text analysis, or "manual" if you want to set TT.options as well. Set to "tokenize" to use tokenize. **lang** A character string specifying a valid language. **TT.options** A list with arguments to be used as TT.options by treetag. **hyph.cache.file** A character string specifying a path to a file to use for storing already hyphenated data, used by hyphen. **add.desc** A logical value, whether tag descriptions should be added directly to tagged text objects. To explicitly unset a value again, set it to an empty character string (e.g., lang="").
validate	Logical, if TRUE given paths will be checked for actual availablity, and the function will fail if files can't be found.

Details

To get the current settings, the function `get.kRp.env` should be used. For the most part, `set.kRp.env` is a convenient wrapper for `options`. To permanently set some defaults, you could also add respective `options` calls to an `.Rprofile` file.

Note that you can also suppress the startup message informing about `available.koRpus.lang` and `install.koRpus.lang` by adding `noStartupMessage=TRUE` to the options in `.Rprofile`.

Value

Returns an invisible `NULL`.

See Also

`get.kRp.env`

Examples

```
set.kRp.env(lang="en")
get.kRp.env(lang=TRUE)
## Not run:
set.kRp.env(
  TT.cmd=file.path("~","bin","treetagger","cmd","tree-tagger-german"),
  lang="de"
)

# example for setting permanent default values in an .Rprofile file
options(
  koRpus=list(
    TT.cmd="manual",
    TT.options=list(
      path=file.path("~","bin","treetagger"),
      preset="de"),
    lang="de",
    noStartupMessage=TRUE
  )
)
# be aware that setting a permamnent default language without loading
# the respective language support package might trigger errors

## End(Not run)
```

---

set.lang.support *Add support for new languages*

Description

You can use this function to add new languages to be used with `koRpus`.

Usage

```
set.lang.support(target, value, merge = TRUE)
```

Arguments

`target`	One of "kRp.POS.tags", "treetag", or "hyphen", depending on what support is to be added.
`value`	A named list that upholds exactly the structure defined here for its respective `target`.
`merge`	Logical, only relevant for the "kRp.POS.tags" target. This argument controls whether `value` will completely replace an already present tagset definition, or merge all given tags (i.e., replace single tags with an updated definition or add new tags).

Details

Language support in this package is designed to be extended easily. You could call it modular, although it's actually more "environemntal", but nevermind.

To add full new language support, say for Xyzedish, you basically have to call this function three times (or at least twice, see hyphen section below) with different targets. If you would like to re-use this language support, you should consider making it a package.

Be it a package or a script, it should contain all three calls to this function. If it succeeds, it will fill an internal environment with the information you have defined.

The function `set.language.support()` gets called three times because there's three functions of koRpus that need language support:

- treetag() needs the preset information from its own start scripts
- kRp.POS.tags() needs to learn all possible POS tags that TreeTagger uses for the given language
- hyphen() needs to know which language pattern tests are available as data files (which you must provide also)

All the calls follow the same pattern – first, you name one of the three targets explained above, and second, you provide a named list as the `value` for the respective `target` function.

"treetag"

The presets for the treetag() function are basically what the shell (GNU/Linux, MacOS) and batch (Win) scripts define that come with TreeTagger. Look for scripts called "$TREETAGGER/cmd/tree-tagger-xyzedish" and "$TREETAGGER\cmd\tree-tagger-xyzedish.bat", figure out which call resembles which call and then define them in set.lang.support("treetag") accordingly.

Have a look at the commented template in your `koRpus` installation directory for an elaborate example.

"kRp.POS.tags"

If Xyzedish is supported by TreeTagger, you should find a tagset definition for the language on its homepage. treetag() needs to know *all* POS tags that TreeTagger might return, otherwise you will get a self-explaining error message as soon as an unknown tag appears. Notice that this can still happen after you implemented the full documented tag set: sometimes the contributed TreeTagger parameter files added their own tags, e.g., for special punctuation. So please test your tag set well.

As you can see in the template file, you will also have to add a global word class and an explaination for each tag. The former is especially important for further steps like frequency analysis.

Again, please have a look at the commented template and/or existing language support files in the package sources, most of it should be almost self-explaining.

"hyphen"

Using the target "hyphen" will cause a call to the equivalent of this function in the `sylly` package. See the documentation of its `set.hyph.support` function for details.

Packaging

If you would like to create a proper language support package, you should only include the "treetag" and "kRp.POS.tags" calls, and the hyphenation patterns should be loaded as a dependency to a package called `sylly.xx`. You can generate such a sylly package rather quickly by using the private function `sylly:::sylly_langpack()`.

Examples

```
hyph_pat_yxz <- sylly::kRp_hyph_pat(
  lang = "xy",
  pattern = matrix(
    c(
      ".im5b", ".in1", ".in3d",
      ".imb", ".in", ".ind",
      "0050", "001", "0030"
    ),
    nrow=3,
    dimnames= list(
      NULL,
      c("orig", "char", "nums")
    )
  )
)
set.lang.support(
  target="hyphen",
  value=list("xyz"=hyph_pat_yxz)
)
```

show,kRp.lang-method *Show methods for koRpus objects*

Description

Show methods for S4 objects of classes kRp.lang, kRp.readability, kRp.corp.freq or kRp.TTR.

Usage

```
## S4 method for signature 'kRp.lang'
show(object)

## S4 method for signature 'kRp.TTR'
show(object)

## S4 method for signature 'kRp.corp.freq'
show(object)

## S4 method for signature 'kRp.readability'
show(object)

## S4 method for signature 'kRp.text'
show(object)
```

Arguments

object	An object of class kRp.lang, kRp.readability, kRp.corp.freq, or kRp.TTR.

See Also

kRp.lang, kRp.readability, kRp.corp.freq, kRp.TTR

Examples

```
## Not run:
  guess.lang("/home/user/data/some.txt", udhr.path="/home/user/data/udhr_txt/")

## End(Not run)
## Not run:
MTLD(tagged.txt)

## End(Not run)
## Not run:
flesch(tagged.txt)

## End(Not run)
```

SMOG *Readability: Simple Measure of Gobbledygook (SMOG)*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
SMOG(
  txt.file,
  hyphen = NULL,
 parameters = c(syll = 3, sqrt = 1.043, fact = 30, const = 3.1291, sqrt.const = 0),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`hyphen`	An object of class kRp.hyphen. If `NULL`, the text will be hyphenated automatically.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates the Simple Measure of Gobbledygook (SMOG). In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

By default calculates formula D by McLaughlin (1969). If `parameters` is set to `SMOG="C"`, formula C will be calculated. If `parameters` is set to `SMOG="simple"`, the simplified formula is used, and if `parameters="de"`, the formula adapted to German texts ("Qu", Bamberger & Vanecek, 1984, p. 78).

Value

An object of class `kRp.readability`.

References

Bamberger, R. & Vanecek, E. (1984). *Lesen–Verstehen–Lernen–Schreiben*. Wien: Jugend und Volk.

McLaughlin, G.H. (1969). SMOG grading – A new readability formula. *Journal of Reading*, 12(8), 639–646.

Examples

```
## Not run:
SMOG(tagged.text)

## End(Not run)
```

---

spache *Readability: Spache Formula*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
spache(
  txt.file,
  word.list,
  parameters = c(asl = 0.121, dword = 0.082, const = 0.659),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`word.list`	A vector or matrix (with exactly one column) which defines familiar words. For valid results the short Dale-Chall list with 769 easy words should be used.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`...`	Further valid options for the main function, see `readability` for details.

Details

Calculates the Spache Formula. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

By default the revised Spache formula is calculated. If `parameters="old"`, the original parameters are used.

This formula doesn't need syllable count.

Value

An object of class `kRp.readability`.

Examples

```
## Not run:
spache(tagged.text, word.list=spache.revised.wl)

## End(Not run)
```

`split_by_doc_id` *Turn a multi-document kRp.text object into a list of kRp.text objects*

Description

For some analysis steps it might be important to have individual tagged texts instead of one large corpus object. This method produces just that.

Usage

```
split_by_doc_id(obj, keepFeatures = TRUE)

## S4 method for signature 'kRp.text'
split_by_doc_id(obj, keepFeatures = TRUE)
```

Arguments

`obj`	An object of class `kRp.text`.
`keepFeatures`	Either logical, whether to keep all features or drop them, or a character vector of names of features to keep if present.

Value

A named list of objects of class `kRp.text`. Elements are named by their `doc_id`.

Examples

```
## Not run:
myCorpusList <- split_by_doc_id(myCorpus)

## End(Not run)
```

strain *Readability: Strain Index*

Description

This is just a convenient wrapper function for readability.

Usage

```
strain(txt.file, hyphen = NULL, parameters = c(sent = 3, const = 10), ...)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for the index.
...	Further valid options for the main function, see readability for details.

Details

This function calculates the Strain index. In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class kRp.readability.

Examples

```
## Not run:
strain(tagged.text)

## End(Not run)
```

summary *Summary methods for koRpus objects*

Description

Summary method for S4 objects of classes kRp.lang, kRp.readability, kRp.text, or kRp.TTR.

Usage

```
summary(object, ...)

## S4 method for signature 'kRp.lang'
summary(object)

## S4 method for signature 'kRp.TTR'
summary(object, flat = FALSE)

## S4 method for signature 'kRp.readability'
summary(object, flat = FALSE)

## S4 method for signature 'kRp.text'
summary(object, index = NA, feature = NULL)
```

Arguments

object	An object of class, kRp.lang, kRp.readability, kRp.text, or kRp.TTR.
...	Further options, depending on the object class.
flat	Logical, if TRUE only a named vector of main results is returned
index	Either a vector indicating which rows should be considered as transformed for the statistics, or the name of a particular transformation that was previously done to the object, if more than one transformation was applied. If NA, all rows where "equal" is FALSE are used. Only valid for objects providing a diff feature.
feature	A character string naming a feature present in the object, to trigger a summary regarding that feature. Currently only "freq" is implemented.

See Also

kRp.lang, kRp.readability, kRp.text, kRp.TTR

Examples

```
## Not run:
summary(guess.lang("/home/user/data/some.txt", udhr.path="/home/user/data/udhr_txt/"))

## End(Not run)
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
```

```
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  ld.results <- lex.div(tokenized.obj, char=c())
  summary(ld.results)
  summary(ld.results, flat=TRUE)
} else {}
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  rdb.results <- readability(tokenized.obj, index="fast")
  summary(rdb.results)
  summary(rdb.results, flat=TRUE)
} else {}
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  # this will look more useful when you
  # can use treetag() instead of tokenize()
  summary(tokenized.obj)
} else {}
```

`taggedText` *Getter/setter methods for koRpus objects*

Description

These methods should be used to get or set values of tagged text objects generated by koRpus functions like `treetag` or `tokenize`.

Usage

```
taggedText(obj, add.desc = FALSE, doc_id = FALSE)
```

```
## S4 method for signature 'kRp.text'
taggedText(obj, add.desc = FALSE, doc_id = FALSE)

taggedText(obj) <- value

## S4 replacement method for signature 'kRp.text'
taggedText(obj) <- value

doc_id(obj, ...)

## S4 method for signature 'kRp.text'
doc_id(obj, has_id = NULL)

hasFeature(obj, feature = NULL, ...)

## S4 method for signature 'kRp.text'
hasFeature(obj, feature = NULL)

hasFeature(obj, feature) <- value

## S4 replacement method for signature 'kRp.text'
hasFeature(obj, feature) <- value

feature(obj, feature, ...)

## S4 method for signature 'kRp.text'
feature(obj, feature, doc_id = NULL)

feature(obj, feature) <- value

## S4 replacement method for signature 'kRp.text'
feature(obj, feature) <- value

corpusReadability(obj, ...)

## S4 method for signature 'kRp.text'
corpusReadability(obj, doc_id = NULL)

corpusReadability(obj) <- value

## S4 replacement method for signature 'kRp.text'
corpusReadability(obj) <- value

corpusHyphen(obj, ...)

## S4 method for signature 'kRp.text'
corpusHyphen(obj, doc_id = NULL)
```

```
corpusHyphen(obj) <- value

## S4 replacement method for signature 'kRp.text'
corpusHyphen(obj) <- value

corpusLexDiv(obj, ...)

## S4 method for signature 'kRp.text'
corpusLexDiv(obj, doc_id = NULL)

corpusLexDiv(obj) <- value

## S4 replacement method for signature 'kRp.text'
corpusLexDiv(obj) <- value

corpusFreq(obj, ...)

## S4 method for signature 'kRp.text'
corpusFreq(obj)

corpusFreq(obj) <- value

## S4 replacement method for signature 'kRp.text'
corpusFreq(obj) <- value

corpusCorpFreq(obj, ...)

## S4 method for signature 'kRp.text'
corpusCorpFreq(obj)

corpusCorpFreq(obj) <- value

## S4 replacement method for signature 'kRp.text'
corpusCorpFreq(obj) <- value

corpusStopwords(obj, ...)

## S4 method for signature 'kRp.text'
corpusStopwords(obj)

corpusStopwords(obj) <- value

## S4 replacement method for signature 'kRp.text'
corpusStopwords(obj) <- value

## S4 method for signature 'kRp.text,ANY,ANY,ANY'
x[i, j, ..., drop = TRUE]
```

```
## S4 replacement method for signature 'kRp.text,ANY,ANY,ANY'
x[i, j, ...] <- value

## S4 method for signature 'kRp.text'
x[[i, doc_id = NULL, ...]]

## S4 replacement method for signature 'kRp.text'
x[[i, doc_id = NULL, ...]] <- value

## S4 method for signature 'kRp.text'
describe(obj, doc_id = NULL, simplify = TRUE, ...)

## S4 replacement method for signature 'kRp.text'
describe(obj, doc_id = NULL, ...) <- value

## S4 method for signature 'kRp.text'
language(obj)

## S4 replacement method for signature 'kRp.text'
language(obj) <- value

diffText(obj, doc_id = NULL)

## S4 method for signature 'kRp.text'
diffText(obj, doc_id = NULL)

diffText(obj) <- value

## S4 replacement method for signature 'kRp.text'
diffText(obj) <- value

originalText(obj)

## S4 method for signature 'kRp.text'
originalText(obj)

is.taggedText(obj)

is.kRp.text(obj)

fixObject(obj, doc_id = NA)

## S4 method for signature 'kRp.text'
fixObject(obj, doc_id = NA)

tif_as_tokens_df(tokens)

## S4 method for signature 'kRp.text'
```

```
tif_as_tokens_df(tokens)

## S4 method for signature 'kRp.tagged'
fixObject(obj, doc_id = NA)

## S4 method for signature 'kRp.txt.freq'
fixObject(obj, doc_id = NA)

## S4 method for signature 'kRp.txt.trans'
fixObject(obj, doc_id = NA)

## S4 method for signature 'kRp.analysis'
fixObject(obj, doc_id = NA)
```

Arguments

obj	An arbitrary R object.
add.desc	Logical, determines whether the desc column should be re-written with descriptions for all POS tags.
doc_id	Logical (except for fixObject, feature, and [[/[[<-), if TRUE the doc_id column will be a factor with the respective value of the desc slot, i.\,e., the document ID will be preserved in the data.frame. If used with fixObject, can be a character string to set the document ID manually (the default NA will preserve existing values and not overwrite them). If used with feature or [[/[[<-, a character vector to limit the scope to one or more particular document IDs.
value	The new value to replace the current with.
...	Additional arguments for the generics.
has_id	A character vector with doc_ids to look for in the object. The return value is then a logical vector of the same length, indicating if the respective id was found or not.
feature	Character string naming the feature to look for. The return value is logical if a single feature name is given. If feature=NULL, a character vector is returned, naming all features found in the object.
x	An object of class kRp.text or kRp.hyphen.
i	Defines the row selector ([) or the name to match ([[).
j	Defines the column selector.
drop	Logical, whether the result should be coerced to the lowest possible dimension. See [for more details.
simplify	Logical, if TRUE and the result is a list oft length one (i.e., just a single doc_id), returns the contents of the single list entry.
tokens	An object of class kRp.text.

Details

- taggedText() returns the tokens slot.

- doc_id() Returns a character vector of all doc_id values in the object.
- describe() returns the desc slot.
- language() returns the lang slot.
- [/[[Can be used as a shortcut to index the results of taggedText().
- fixObject returns the same object upgraded to the object structure of this package version (e.g., new columns, changed names, etc.).
- hasFeature() returns TRUE or codeFALSE, depending on whether the requested feature is present or not.
- feature() returns the list entry of the feat_list slot for the requested feature.
- corpusReadability() returns the list of kRp.readability objects, see readability.
- corpusHyphen() returns the list of kRp.hyphen objects, see hyphen.
- corpusLexDiv() returns the list of kRp.TTR objects, see lex.div.
- corpusFreq() returns the frequency analysis data from the feat_list slot, see freq.analysis.
- corpusCorpFreq() returns the kRp.corp.freq object of the feat_list slot, see for example read.corp.custom.
- corpusStopwords() returns the number of stopwords found in each text (if analyzed) from the feat_list slot.
- tif_as_tokens_df returns the tokens slot in a TIF[1] compliant format, i.e., doc_id is not a factor but a character vector.
- originalText() similar to taggedText(), but reverts any transformations back to the original text before returning the tokens slot. Only works if the object has the feature diff, see examples.
- diffText() returns the diff slot, if present.

References

[1] Text Interchange Formats (https://github.com/ropensci/tif)

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )

  doc_id(tokenized.obj)

  describe(tokenized.obj)

  language(tokenized.obj)
```

```
  taggedText(tokenized.obj)
  tokenized.obj[["token"]]
  tokenized.obj[1:3, "token"]

  tif_as_tokens_df(tokenized.obj)

  # example for originalText()
  tokenized.obj <- jumbleWords(tokenized.obj)
  # now compare the jumbled words to the original
  tokenized.obj[["token"]]
  originalText(tokenized.obj)[["token"]]
} else {}
```

`textFeatures` *Extract text features for authorship analysis*

Description

This function combines several of koRpus' methods to extract the 9-Feature Set for authorship detection (Brannon, Afroz & Greenstadt, 2011; Brannon & Greenstadt, 2009).

Usage

```
textFeatures(text, hyphen = NULL)
```

Arguments

`text`	An object of class `kRp.text`. Can also be a list of these objects, if you want to analyze more than one text at once.
`hyphen`	An object of class `kRp.hyphen`, if `text` has already been hyphenated. If `text` is a list and `hyphen` is not `NULL`, it must also be a list with one object for each text, in the same order.

Value

A data.frame:

uniqWd Number of unique words (tokens)

cmplx Complexity (TTR)

sntCt Sentence count

sntLen Average sentence length

syllCt Average syllable count

charCt Character count (all characters, including spaces)

lttrCt Letter count (without spaces, punctuation and digits)

FOG Gunning FOG index

flesch Flesch Reading Ease index

References

Brennan, M., Afroz, S., & Greenstadt, R. (2011). Deceiving authorship detection. Presentation at *28th Chaos Communication Congress (28C3)*, Berlin, Germany. Brennan, M. & Greenstadt, R. (2009). Practical Attacks Against Authorship Recognition Techniques. In *Proceedings of the Twenty-First Conference on Innovative Applications of Artificial Intelligence (IAAI)*, Pasadena, CA. Tweedie, F.J., Singh, S., & Holmes, D.I. (1996). Neural Network Applications in Stylometry: The Federalist Papers. *Computers and the Humanities*, 30, 1–10.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  textFeatures(tokenized.obj)
} else {}
```

---

textTransform *Letter case transformation*

Description

Transforms text in koRpus objects token by token.

Usage

```
textTransform(txt, ...)

## S4 method for signature 'kRp.text'
textTransform(
  txt,
  scheme,
  p = 0.5,
  paste = FALSE,
  var = "wclass",
  query = "fullstop",
  method = "replace",
  replacement = ".",
  f = NA,
  ...
)
```

Arguments

txt	An object of class kRp.text.
...	Parameters passed to query to find matching tokens. Relevant only if scheme="normalize".
scheme	One of the following character strings: • "minor" Start each word with a lowercase letter. • "all.minor" Forces all letters into lowercase. • "major" Start each word with a uppercase letter. • "all.major" Forces all letters into uppercase. • "random" Randomly start words with uppercase or lowercase letters. • "de.norm" German norm: All names, nouns and sentence beginnings start with an uppercase letter, anything else with a lowercase letter. • "de.inv" Inversion of "de.norm". • "eu.norm" Usual European cases: Only names and sentence beginnings start with an uppercase letter, anything else with a lowercase letter. • "eu.inv" Inversion of "eu.norm". • "normalize" Replace all tokens matching query in column var according to method (see below).
p	Numeric value between 0 and 1. Defines the probability for upper case letters (relevant only if scheme="random").
paste	Logical, see value section.
var	A character string naming a variable in the object (i.e., colname). See query for details. Relevant only if scheme="normalize".
query	A character vector (for words), regular expression, or single number naming values to be matched in the variable. See query for details. Relevant only if scheme="normalize".
method	One of the following character strings: • "shortest" Replace all matches with the shortest value found. • "longest" Replace all matches with the longest value found. • "replace" Replace all matches with the token given via replacement. • "function" Replace all matches with the result of the function provided by f (see section Function for details). In case of "shortest" and "longest", if multiple values of the same length are found, the (first) most prevalent one is being used. The actual replacement value is documented in the diff slot of the object, as a list called transfmt.normalize. Relevant only if scheme="normalize".
replacement	Character string defining the exact token to replace all query matches with. Relevant only if scheme="normalize" and method="replace".
f	A function to calculate the replacement for all query matches. Relevant only if scheme="normalize" and method="function".

Details

This method is mainly intended to produce text material for experiments.

Value

By default an object of class kRp.text with the added feature diff is returned. It provides a list with mostly atomic vectors, describing the amount of diffences between both text variants (percentage):

all.tokens: Percentage of all tokens, including punctuation, that were altered.

words: Percentage of altered words only.

all.chars: Percentage of all characters, including punctuation, that were altered.

letters: Percentage of altered letters in words only.

transfmt: Character vector documenting the transformation(s) done to the tokens.

transfmt.equal: Data frame documenting which token was changed in which transformational step. Only available if more than one transformation was done.

transfmt.normalize: A list documenting steps of normalization that were done to the object, one element per transformation. Each entry holds the name of the method, the query parameters, and the effective replacement value.

If paste=TRUE, returns an atomic character vector (via pasteText).

Function

You can dynamically calculate the replacement value for the "normalize" scheme by setting method="function" and providing a function object as f. The function you provide must support the following arguments:

- tokens The original tokens slot of the txt object (see taggedText).
- match A logical vector, indicating for each row of tokens whether it's a query match or not.

You can then use these arguments in your function body to calculate the replacement, e.g. tokens[match,"token"] to get all relevant tokens. The return value of the function will be used as the replacement for all matched tokens. You probably want to make sure it's a character vecor of length one or of the same length as all matches.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )
  tokenized.obj <- textTransform(
    tokenized.obj,
    scheme="random"
  )
  pasteText(tokenized.obj)

  # diff stats are now part of the object
```

```
  hasFeature(tokenized.obj)
  diffText(tokenized.obj)
} else {}
```

---

tokenize *A simple tokenizer*

Description

This tokenizer can be used to try replace TreeTagger. Its results are not as detailed when it comes to word classes, and no lemmatization is done. However, for most cases this should suffice.

Usage

```
tokenize(
  txt,
  format = "file",
  fileEncoding = NULL,
  split = "[[:space:]]",
  ign.comp = "-",
  heuristics = "abbr",
  heur.fix = list(pre = c("\u2019", "'"), suf = c("\u2019", "'")),
  abbrev = NULL,
  tag = TRUE,
  lang = "kRp.env",
  sentc.end = c(".", "!", "?", ";", ":"),
  detect = c(parag = FALSE, hline = FALSE),
  clean.raw = NULL,
  perl = FALSE,
  stopwords = NULL,
  stemmer = NULL,
  doc_id = NA,
  add.desc = "kRp.env",
  ...
)

## S4 method for signature 'character'
tokenize(
  txt,
  format = "file",
  fileEncoding = NULL,
  split = "[[:space:]]",
  ign.comp = "-",
  heuristics = "abbr",
  heur.fix = list(pre = c("\u2019", "'"), suf = c("\u2019", "'")),
  abbrev = NULL,
  tag = TRUE,
```

```
  lang = "kRp.env",
  sentc.end = c(".", "!", "?", ";", ":"),
  detect = c(parag = FALSE, hline = FALSE),
  clean.raw = NULL,
  perl = FALSE,
  stopwords = NULL,
  stemmer = NULL,
  doc_id = NA,
  add.desc = "kRp.env"
)

## S4 method for signature 'kRp.connection'
tokenize(
  txt,
  format = NA,
  fileEncoding = NULL,
  split = "[[:space:]]",
  ign.comp = "-",
  heuristics = "abbr",
  heur.fix = list(pre = c("\u2019", "'"), suf = c("\u2019", "'")),
  abbrev = NULL,
  tag = TRUE,
  lang = "kRp.env",
  sentc.end = c(".", "!", "?", ";", ":"),
  detect = c(parag = FALSE, hline = FALSE),
  clean.raw = NULL,
  perl = FALSE,
  stopwords = NULL,
  stemmer = NULL,
  doc_id = NA,
  add.desc = "kRp.env"
)
```

Arguments

txt	Either an open connection, the path to directory with txt files to read and tokenize, or a vector object already holding the text corpus.
format	Either "file" or "obj", depending on whether you want to scan files or analyze the given object. Ignored if txt is a connection.
fileEncoding	A character string naming the encoding of all files.
split	A regular expression to define the basic split method. Should only need refinement for languages that don't separate words by space.
ign.comp	A character vector defining punctuation which might be used in composita that should not be split.
heuristics	A vector to indicate if the tokenizer should use some heuristics. Can be none, one or several of the following:

- "abbr"Assume that "letter-dot-letter-dot" combinations are abbreviations and leave them intact.

- "suf"Try to detect possesive suffixes like "'s", or shorting suffixes like "'ll" and treat them as one token
- "pre"Try to detect prefixes like "s'" or "l'" and treat them as one token

Earlier releases used the names "en" and "fr" instead of "suf" and "pre". They are still working, that is "en" is equivalent to "suf", whereas "fr" is now equivalent to both "suf" and "pre" (and not only "pre" as in the past, which was missing the use of suffixes in French).

heur.fix — A list with the named vectors pre and suf. These will be used if heuristics were set to use one of the presets that try to detect pre- and/or suffixes. Change them if you document uses other characters than the ones defined by default.

abbrev — Path to a text file with abbreviations to take care of, one per line. Note that this file must have the same encoding as defined by fileEncoding.

tag — Logical. If TRUE, the text will be rudimentarily tagged and returned as an object of class kRp.text.

lang — A character string naming the language of the analyzed text. If set to "kRp.env" this is fetched from get.kRp.env. Only needed if tag=TRUE.

sentc.end — A character vector with tokens indicating a sentence ending. Only needed if tag=TRUE.

detect — A named logical vector, indicating by the setting of parag and hline whether tokenize should try to detect paragraphs and headlines.

clean.raw — A named list of character values, indicating replacements that should globally be made to the text prior to tokenizing it. This is applied after the text was converted into UTF-8 internally. In the list, the name of each element represents a pattern which is replaced by its value if met in the text. Since this is done by calling gsub, regular expressions are basically supported. See the perl attribute, too.

perl — Logical, only relevant if clean.raw is not NULL. If perl=TRUE, this is forwarded to gsub to allow for perl-like regular expressions in clean.raw.

stopwords — A character vector to be used for stopword detection. Comparison is done in lower case. You can also simply set stopwords=tm::stopwords("en") to use the english stopwords provided by the tm package.

stemmer — A function or method to perform stemming. For instance, you can set SnowballC::wordStem if you have the SnowballC package installed. As of now, you cannot provide further arguments to this function.

doc_id — Character string, optional identifier of the particular document. Will be added to the desc slot, and as a factor to the "doc_id" column of the tokens slot. If NA, the document name will be used (for format="obj" a random name).

add.desc — Logical. If TRUE, the tag description (column "desc" of the data.frame) will be added directly to the resulting object. If set to "kRp.env" this is fetched from get.kRp.env. Only needed if tag=TRUE.

... — Only used for the method generic.

Details

tokenize can try to guess what's a headline and where a paragraph was inserted (via the detect parameter). A headline is assumed if a line of text without sentence ending punctuation is found, a paragraph if two blocks of text are separated by space. This will add extra tags into the text: "<kRp.h>" (headline starts), "</kRp.h>" (headline ends) and "<kRp.p/>" (paragraph), respectively. This can be useful in two cases: "</kRp.h>" will be treated like a sentence ending, which gives you more control for automatic analyses. And adding to that, pasteText can replace these tags, which probably preserves more of the original layout.

Value

If tag=FALSE, a character vector with the tokenized text. If tag=TRUE, returns an object of class kRp.text.

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )

  ## character manipulation
  # this is useful if you know of problematic characters in your
  # raw text files, but don't want to touch them directly. you
  # don't have to, as you can substitute them, even using regular
  # expressions. a simple example: replace all single quotes by
  # double quotes througout the text:
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en",
    clean.raw=list("'"="'\"')
  )

  # now replace all occurrances of the letter A followed
  # by two digits with the letter B, followed by the same
  # two digits:
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en",
    clean.raw=list("(A)([[:digit:]]{2})"="B\\2"),
    perl=TRUE
  )

  ## enabling stopword detection and stemming
  if(all(
    requireNamespace("tm", quietly=TRUE),
```

```
    requireNamespace("SnowballC", quietly=TRUE)
  )){
    # if you also installed the packages tm and Snowball,
    # you can use some of their features with koRpus:
    tokenized.obj <- tokenize(
      txt=sample_file,
      lang="en",
      stopwords=tm::stopwords("en"),
      stemmer=SnowballC::wordStem
    )

    # removing all stopwords now is simple:
    tokenized.noStopWords <- filterByClass(tokenized.obj, "stopword")
  } else {}
} else {}
```

`traenkle.bailer` *Readability: Traenkle-Bailer Formeln*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
traenkle.bailer(
  txt.file,
  TB1 = c(const = 224.6814, awl = 79.8304, asl = 12.24032, prep = 1.292857),
  TB2 = c(const = 234.1063, awl = 96.11069, prep = 2.05444, conj = 1.02805),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`TB1`	A numeric vector with named magic numbers for the first of the formulas.
`TB2`	A numeric vector with named magic numbers for the second of the formulas.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates the two formulae by Tr\"ankle-Bailer, which are based on the Dickes-Steiwer formulae. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index values.

This formula doesn't need syllable count.

Value

An object of class kRp.readability.

Examples

```
## Not run:
  traenkle.bailer(tagged.text)

## End(Not run)
```

---

treetag *A method to call TreeTagger*

Description

This method calls a local installation of TreeTagger[1] to tokenize and POS tag the given text.

Usage

```
treetag(
  file,
  treetagger = "kRp.env",
  rm.sgml = TRUE,
  lang = "kRp.env",
  apply.sentc.end = TRUE,
  sentc.end = c(".", "!", "?", ";", ":"),
  encoding = NULL,
  TT.options = NULL,
  debug = FALSE,
  TT.tknz = TRUE,
  format = "file",
  stopwords = NULL,
  stemmer = NULL,
  doc_id = NA,
  add.desc = "kRp.env",
  ...
)

## S4 method for signature 'character'
treetag(
  file,
  treetagger = "kRp.env",
  rm.sgml = TRUE,
  lang = "kRp.env",
  apply.sentc.end = TRUE,
  sentc.end = c(".", "!", "?", ";", ":"),
  encoding = NULL,
```

```
  TT.options = NULL,
  debug = FALSE,
  TT.tknz = TRUE,
  format = "file",
  stopwords = NULL,
  stemmer = NULL,
  doc_id = NA,
  add.desc = "kRp.env"
)

## S4 method for signature 'kRp.connection'
treetag(
  file,
  treetagger = "kRp.env",
  rm.sgml = TRUE,
  lang = "kRp.env",
  apply.sentc.end = TRUE,
  sentc.end = c(".", "!", "?", ";", ":"),
  encoding = NULL,
  TT.options = NULL,
  debug = FALSE,
  TT.tknz = TRUE,
  format = NA,
  stopwords = NULL,
  stemmer = NULL,
  doc_id = NA,
  add.desc = "kRp.env"
)
```

Arguments

`file`	Either a connection or a character vector, valid path to a file, containing the text to be analyzed. If `file` is a connection, its contents will be written to a temporary file, since TreeTagger can't read from R connection objects.
`treetagger`	A character vector giving the TreeTagger script to be called. If set to `"kRp.env"` this is got from `get.kRp.env`. Only if set to `"manual"`, it is assumend not to be a wrapper script that can work the given text file, but that you would like to manually tweak options for tokenizing and POS tagging yourself. In that case, you need to provide a full set of options with the `TT.options` parameter.
`rm.sgml`	Logical, whether SGML tags should be ignored and removed from output
`lang`	A character string naming the language of the analyzed corpus. See `kRp.POS.tags` and `available.koRpus.lang`for all supported languages. If set to `"kRp.env"` this is fetched from `get.kRp.env`.
`apply.sentc.end`	Logical, whethter the tokens defined in `sentc.end` should be searched and set to a sentence ending tag.
`sentc.end`	A character vector with tokens indicating a sentence ending. This adds to TreeTaggers results, it doesn't really replace them.

encoding — A character string defining the character encoding of the input file, like "Latin1" or "UTF-8". If NULL, the encoding will either be taken from a preset (if defined in TT.options), or fall back to "". Hence you can overwrite the preset encoding with this parameter.

TT.options — A list of options to configure how TreeTagger is called. You have two basic choices: Either you choose one of the pre-defined presets or you give a full set of valid options:

- path Mandatory: The absolute path to the TreeTagger root directory. That is where its subfolders bin, cmd and lib are located.
- preset Optional: If you choose one of the pre-defined presets of one of the available language packages (like "de" for German, see available.koRpus.lang for details), you can omit all the following elements, because they will be filled with defaults. Of course this only makes sense if you have a working default installation. Note that since koRpus 0.07-1, UTF-8 is the global default encoding.
- tokenizer Mandatory: A character string, naming the tokenizer to be called. Interpreted relative to path/cmd/.
- tknz.opts Optional: A character string with the options to hand over to the tokenizer. You don't need to specify "-a" if abbrev is given. If TT.tknz=FALSE, you can pass configurational options to tokenize by providing them as a named list (instead of a character string) here.
- pre.tagger Optional: A character string with code to be run before the tagger. This code is used as-is, so you need make sure it includes the needed pipe symbols.
- tagger Mandatory: A character string, naming the tagger-command to be called. Interpreted relative to path/bin/.
- abbrev Optional: A character string, naming the abbreviation list to be used. Interpreted relative to path/lib/.
- params Mandatory: A character string, naming the parameter file to be used. Interpreted relative to path/lib/.
- lexicon Optional: A character string, naming the lexicon file to be used. Interpreted relative to path/lib/.
- lookup Optional: A character string, naming the lexicon lookup command. Interpreted relative to path/cmd/.
- filter Optional: A character string, naming the output filter to be used. Interpreted relative to path/cmd/.
- no.unknown Optional: Logical, can be used to toggle the "-no-unknown" option of TreeTagger (defaults to FALSE).
- splitter Optional: A character string, naming the splitter to be called (before the tokenizer). Interpreted relative to path/cmd/.
- splitter.opts Optional: A character string with the options to hand over to the splitter.

You can also set these options globally using set.kRp.env, and then force treetag to use them by setting TT.options="kRp.env" here. Note: If you use the treetagger setting from kRp.env and it's set to TT.cmd="manual", treetag will treat TT.options=NULL like TT.options="kRp.env" automatically.

debug	Logical. Especially in cases where the presets wouldn't work as expected, this switch can be used to examine the values treetag is assuming.
TT.tknz	Logical, if FALSE TreeTagger's tokenzier script will be replaced by koRpus' function tokenize. To accomplish this, its results will be written to a temporal file which is automatically deleted afterwards (if debug=FALSE). Note that this option only has an effect if treetagger="manual".
format	Either "file" or "obj", depending on whether you want to scan files or analyze the text in a given object, like a character vector. If the latter, it will be written to a temporary file (see file).
stopwords	A character vector to be used for stopword detection. Comparison is done in lower case. You can also simply set stopwords=tm::stopwords("en") to use the english stopwords provided by the tm package.
stemmer	A function or method to perform stemming. For instance, you can set SnowballC::wordStem if you have the SnowballC package installed. As of now, you cannot provide further arguments to this function.
doc_id	Character string, optional identifier of the particular document. Will be added to the desc slot, and as a factor to the "doc_id" column of the tokens slot. If NA, the document name will be used (for format="obj" a random name).
add.desc	Logical. If TRUE, the tag description (column "desc" of the data.frame) will be added directly to the resulting object. If set to "kRp.env" this is fetched from get.kRp.env.
...	Only used for the method generic.

Details

Note that the value of lang must match a valid language supported by kRp.POS.tags. It will also get stored in the resulting object and might be used by other functions at a later point. E.g., treetag is being called by freq.analysis, which will by default query this language definition, unless explicitly told otherwise. The rationale behind this is to comfortably make it possible to have tokenized and POS tagged objects of various languages around in your workspace, and not worry about that too much.

Value

An object of class kRp.text. If debug=TRUE, prints internal variable settings and attempts to return the original output if the TreeTagger system call in a matrix.

Author(s)

m.eik michalke <meik.michalke@hhu.de>, support for various laguages was contributed by Earl Brown (Spanish), Alberto Mirisola (Italian) and Alexandre Brulet (French).

References

Schmid, H. (1994). Probabilistic part-of-speec tagging using decision trees. In *International Conference on New Methods in Language Processing*, Manchester, UK, 44–49.

[1] http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

See Also

freq.analysis, get.kRp.env, kRp.text

Examples

```
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
## Not run:
# first way to invoke POS tagging, using a built-in preset:
tagged.results <- treetag(
  sample_file,
  treetagger="manual",
  lang="en",
  TT.options=list(
    path=file.path("~","bin","treetagger"),
    preset="en"
  )
)
# second way, use one of the batch scripts that come with TreeTagger:
tagged.results <- treetag(
  sample_file,
  treetagger=file.path("~","bin","treetagger","cmd","tree-tagger-english"),
  lang="en"
)
# third option, set the above batch script in an environment object first:
set.kRp.env(
  TT.cmd=file.path("~","bin","treetagger","cmd","tree-tagger-english"),
  lang="en"
)
tagged.results <- treetag(
  sample_file
)

# after tagging, use the resulting object with other functions in this package:
readability(tagged.results)
lex.div(tagged.results)

## enabling stopword detection and stemming
# if you also installed the packages tm and SnowballC,
# you can use some of their features with koRpus:
set.kRp.env(
  TT.cmd="manual",
  lang="en",
  TT.options=list(
    path=file.path("~","bin","treetagger"),
    preset="en"
  )
)
tagged.results <- treetag(
  sample_file,
  stopwords=tm::stopwords("en"),
```

```
  stemmer=SnowballC::wordStem
)

# removing all stopwords now is simple:
tagged.noStopWords <- filterByClass(
  tagged.results,
  "stopword"
)

## End(Not run)
```

TRI *Readability: Kuntzsch's Text-Redundanz-Index*

Description

This is just a convenient wrapper function for readability.

Usage

```
TRI(
  txt.file,
  hyphen = NULL,
 parameters = c(syll = 1, word = 0.449, pnct = 2.467, frgn = 0.937, const = 14.417),
  ...
)
```

Arguments

txt.file	Either an object of class kRp.text, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by readability.num.
hyphen	An object of class kRp.hyphen. If NULL, the text will be hyphenated automatically.
parameters	A numeric vector with named magic numbers, defining the relevant parameters for the index.
...	Further valid options for the main function, see readability for details.

Details

This function calculates Kuntzsch's Text-Redundanz-Index (text redundancy index). In contrast to readability, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class kRp.readability.

Examples

```
## Not run:
  TRI(tagged.text)

## End(Not run)
```

TTR *Lexical diversity: Type-Token Ratio*

Description

This is just a convenient wrapper function for lex.div.

Usage

```
TTR(txt, char = FALSE, ...)
```

Arguments

txt	An object of class kRp.text containing the tagged text to be analyzed.
char	Logical, defining whether data for plotting characteristic curves should be calculated.
...	Further valid options for the main function, see lex.div for details.

Details

This function calculates the classic type-token ratio (TTR). In contrast to lex.div, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the TTR value, and characteristics are off by default.

Value

An object of class kRp.TTR.

See Also

kRp.POS.tags, kRp.text, kRp.TTR

Examples

```
## Not run:
TTR(tagged.text)

## End(Not run)
```

`tuldava` *Readability: Tuldava's Text Difficulty Formula*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
tuldava(
  txt.file,
  hyphen = NULL,
  parameters = c(syll = 1, word1 = 1, word2 = 1, sent = 1),
  ...
)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`hyphen`	An object of class kRp.hyphen. If `NULL`, the text will be hyphenated automatically.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates Tuldava's Text Difficulty Formula. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

Value

An object of class `kRp.readability`.

Note

This index originally has no parameter weights. To be able the use weights anyway, each parameter of the formula is available and its weight set to 1 by default.

Examples

```
## Not run:
  tuldava(tagged.text)

## End(Not run)
```

types *Get types and tokens of a given text*

Description

These methods return character vectors that return all types or tokens of a given text, where text can either be a character vector itself, a previosly tokenized/tagged koRpus object, or an object of class kRp.TTR.

Usage

```
types(txt, ...)

tokens(txt, ...)

## S4 method for signature 'kRp.TTR'
types(txt, stats = FALSE)

## S4 method for signature 'kRp.TTR'
tokens(txt)

## S4 method for signature 'kRp.text'
types(
  txt,
  case.sens = FALSE,
  lemmatize = FALSE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c(),
  stats = FALSE
)

## S4 method for signature 'kRp.text'
tokens(
  txt,
  case.sens = FALSE,
  lemmatize = FALSE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c()
)

## S4 method for signature 'character'
types(
  txt,
  case.sens = FALSE,
  lemmatize = FALSE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c(),
```

```
  stats = FALSE,
  lang = NULL
)

## S4 method for signature 'character'
tokens(
  txt,
  case.sens = FALSE,
  lemmatize = FALSE,
  corp.rm.class = "nonpunct",
  corp.rm.tag = c(),
  lang = NULL
)
```

Arguments

txt	An object of either class kRp.text or kRp.TTR, or a character vector.
...	Only used for the method generic.
stats	Logical, whether statistics on the length in characters and frequency of types in the text should also be returned.
case.sens	Logical, whether types should be counted case sensitive. This option is available for tagged text and character input only.
lemmatize	Logical, whether analysis should be carried out on the lemmatized tokens rather than all running word forms. This option is available for tagged text and character input only.
corp.rm.class	A character vector with word classes which should be dropped. The default value "nonpunct" has special meaning and will cause the result of kRp.POS.tags(lang,tags=c("punct to be used. This option is available for tagged text and character input only.
corp.rm.tag	A character vector with POS tags which should be dropped. This option is available for tagged text and character input only.
lang	Set the language of a text, see the force.lang option of lex.div. This option is available for character input only.

Value

A character vector. Fortypes and stats=TRUE a data.frame containing all types, their length (characters) and frequency. The types result is always sorted by frequency, with more frequent types coming first.

Note

If the input is of class kRp.TTR, the result will only be useful if lex.div or the respective wrapper function was called with keep.tokens=TRUE. Similarily, lemmatize can only work properly if the input is a tagged text object with lemmata or you've properly set up the enviroment via set.kRp.env. Calling these methods on kRp.TTR objects is just returning the respective part of its tt slot.

See Also

kRp.POS.tags, kRp.text, kRp.TTR, lex.div

Examples

```
# code is only run when the english language package can be loaded
if(require("koRpus.lang.en", quietly = TRUE)){
  sample_file <- file.path(
    path.package("koRpus"), "examples", "corpus", "Reality_Winner.txt"
  )
  tokenized.obj <- tokenize(
    txt=sample_file,
    lang="en"
  )

  types(tokenized.obj)
  tokens(tokenized.obj)
} else {}
```

U.ld *Lexical diversity: Uber Index (U)*

Description

This is just a convenient wrapper function for lex.div.

Usage

```
U.ld(txt, char = FALSE, ...)
```

Arguments

txt	An object of class kRp.text containing the tagged text to be analyzed.
char	Logical, defining whether data for plotting characteristic curves should be calculated.
...	Further valid options for the main function, see lex.div for details.

Details

This function calculates the Uber Index (U). In contrast to lex.div, which by default calculates all possible measures and their progressing characteristics, this function will only calculate the U value, and characteristics are off by default.

Value

An object of class kRp.TTR.

See Also

kRp.POS.tags, kRp.text, kRp.TTR

Examples

```
## Not run:
U.ld(tagged.text)

## End(Not run)
```

`wheeler.smith` *Readability: Wheeler-Smith Score*

Description

This is just a convenient wrapper function for `readability`.

Usage

```
wheeler.smith(txt.file, hyphen = NULL, parameters = c(syll = 2), ...)
```

Arguments

`txt.file`	Either an object of class `kRp.text`, a character vector which must be be a valid path to a file containing the text to be analyzed, or a list of text features. If the latter, calculation is done by `readability.num`.
`hyphen`	An object of class kRp.hyphen. If `NULL`, the text will be hyphenated automatically.
`parameters`	A numeric vector with named magic numbers, defining the relevant parameters for the index.
`...`	Further valid options for the main function, see `readability` for details.

Details

This function calculates the Wheeler-Smith Score. In contrast to `readability`, which by default calculates all possible indices, this function will only calculate the index value.

If `parameters="de"`, the calculation stays the same, but grade placement is done according to Bamberger & Vanecek (1984), that is for german texts.

Value

An object of class `kRp.readability`.

References

Bamberger, R. & Vanecek, E. (1984). *Lesen–Verstehen–Lernen–Schreiben*. Wien: Jugend und Volk.

Wheeler, L.R. & Smith, E.H. (1954). A practical readability formula for the classroom teacher in the primary grades. *Elementary English*, 31, 397–399.

Examples

```
## Not run:
  wheeler.smith(tagged.text)

## End(Not run)
```

Index