Package 'koRpus'

January 27, 2015

Type Package

Title An R Package for Text Analysis

Depends R (>= 2.10.0),methods

Enhances rkward

Suggests testthat,tm,SnowballC,shiny

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. koRpus also includes a plugin for the R GUI and IDE RKWard, providing dialogs for its basic features. To use them, install RKWard from http://rkward.sf.net (plugins are detected automatically). Due to some restrictions on CRAN, the full package sources are only available from the project homepage.

License GPL (>= 3)

Encoding UTF-8

LazyLoad yes

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

Version 0.05-5

Date 2014-03-19

Collate '00_class_01_kRp.tagged.R' '00_class_02_kRp.TTR.R' '00_class_03_kRp.txt.freq.R' '00_class_04_kRp.txt.trans.R' '00_class_05_kRp.analysis.R' '00_class_06_kRp.corp.freq.R' '00_class_07_kRp.hyph.pat.R' '00_class_08_kRp.hyphen.R' '00_class_09_kRp.lang.R' '00_class_10_kRp.readability.R' '01_method_cTest.R' 'kRp.filter.wclass.R' 'koRpus-internal.R' '01_method_clozeDelete.R' '01_method_correct.R' '01_method_kRp.taggedText.R' '01_method_plot.kRp.tagged.R' '01_method_query.R' '01_method_show.kRp.lang.R'

'01_method_show.kRp.TTR.R' '01_method_show.kRp.corp.freq.R'
'01_method_show.kRp.readability.R'
'01_method_summary.kRp.lang.R' '01_method_summary.kRp.TTR.R'
'01_method_summary.kRp.readability.R'
'01_method_summary.kRp.tagged.R'
'01_method_summary.kRp.txt.freq.R' 'ARI.R' 'C.ld.R' 'CTTR.R'
'DRP.R' 'ELF.R' 'FOG.R' 'FORCAST.R' 'HDD.R' 'K.ld.R' 'LIX.R'
'MATTR.R' 'MSTTR.R' 'MTLD.R' 'R.ld.R' 'RIX.R' 'S.ld.R' 'SMOG.R'
'TRI.R' 'TTR.R' 'U.ld.R' 'bormuth.R' 'coleman.R'
'coleman.liau.R' 'dale.chall.R' 'danielson.bryan.R'
'dickes.steiwer.R' 'farr.jenkins.paterson.R' 'flesch.R'
'flesch.kincaid.R' 'freq.analysis.R' 'fucks.R' 'get.kRp.env.R'
'guess.lang.R' 'harris.jacobson.R' 'hyph.XX-data.R' 'hyphen.R'
'jumbleWords.R' 'kRp.POS.tags.R' 'kRp.cluster.R'
'kRp.text.analysis.R' 'kRp.text.paste.R' 'kRp.text.transform.R'
'koRpus-internal.import.R' 'koRpus-internal.lexdiv.formulae.R'
'koRpus-internal.rdb.formulae.R'
'koRpus-internal.rdb.params.grades.R'
'koRpus-internal.roxy.all.R' 'koRpus-package.R'
'lang.support-de.R' 'lang.support-en.R' 'lang.support-es.R'
'lang.support-fr.R' 'lang.support-it.R' 'lang.support-ru.R'
'lex.div.R' 'lex.div.num.R' 'linsear.write.R' 'maas.R'
'manage.hyph.pat.R' 'nWS.R' 'read.BAWL.R' 'read.corp.LCC.R'
'read.corp.celex.R' 'read.corp.custom.R' 'read.hyph.pat.R'
'read.tagged.R' 'readability.R' 'readability.num.R'
'segment.optimizer.R' 'set.kRp.env.R' 'spache.R' 'strain.R'
'textFeatures.R' 'tokenize.R' 'traenkle.bailer.R' 'treetag.R'
'wheeler.smith.R'

Author m.eik michalke [aut, cre],
Earl Brown [ctb],
Alberto Mirisola [ctb],
Alexandre Brulet [ctb],
Laura Hauser [ctb]

Maintainer m.eik michalke <meik.michalke@hhu.de>

NeedsCompilation no

Repository CRAN

Date/Publication 2014-03-20 03:17:04

R topics documented:

`koRpus-package` *The koRpus Package*

Description

An R Package for Text Analysis.

Details

Package:	koRpus
Type:	Package
Version:	0.05-5
Date:	2014-03-19

Depends:	R (>= 2.10.0),methods
Enhances:	rkward
Encoding:	UTF-8
License:	GPL (>= 3)
LazyLoad:	yes
URL:	http://reaktanz.de/?c=hacking&s=koRpus

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. koRpus also includes a plugin for the R GUI and IDE RKWard, providing dialogs for its basic features. To use them, install RKWard from http://rkward.sf.net (plugins are detected automatically). Due to some restrictions on CRAN, the full package sources are only available from the project homepage.

Author(s)

m.eik michalke, with contributions from Earl Brown, Alberto Mirisola, Alexandre Brulet, Laura Hauser

---

`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.tagged-class`, 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-class`.

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)
```

`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.tagged-class, 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.
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.
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 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-class.

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 either class kRp.tagged-class or kRp.analysis-class, 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-class.

See Also

kRp.POS.tags, kRp.tagged-class, kRp.TTR-class

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.taggedText'
clozeDelete(obj, every = 5, offset = 0,
  replace.by = "_", fixed = 10)
```

Arguments

`...`	Additional arguments to the method (as described in this document).
`obj`	An object of class "kRp.tagged"
`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

And object of class kRp.tagged, with an additional list `cloze` in its `desc` slot, listing the words which were changed.

`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.tagged-class`, 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-class.

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.tagged-class, 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-class.

Examples

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

## End(Not run)
```

---

correct.tag *Methods to correct koRpus objects*

Description

The methods correct.tag and correct.hyph can be used to alter objects of class kRp.tagged-class, or of class kRp.hyphen-class respectively.

Usage

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

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

correct.hyph(obj, word = NULL, hyphen = NULL, cache = TRUE)

## S4 method for signature 'kRp.hyphen'
correct.hyph(obj, word = NULL, hyphen = NULL,
  cache = TRUE)
```

Arguments

obj	An object of class kRp.tagged-class or kRp.hyphen-class.
row	Integer, the row number of the entry to be changed. Can be an integer vector to change several rows in one go.
word	A character string, the (possibly incorrectly hyphenated) word entry to be replaced with hyphen.
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.
hyphen	A character string, the new manually hyphenated version of word. Mustn't contain anything other than characters of word plus the hyphenation mark "-".
cache	Logical, if TRUE, the given hyphenation will be added to the sessions' hyphenation cache. Existing entries for the same word will be replaced.

Details

Although automatic POS tagging, lemmatization and hyphenation are remarkably accurate, the algorithms do ususally produce some errors. If you want to correct for these flaws, these methods can be of help, because they might prevent you from introducing new errors. That is, the 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.

correct.hyph will check whether word and hyphen are actually hyphenations of the same token before proceeding. If so, it will also recalculate the number of syllables and update the syll field.

If both word and hyphen are NULL, correct.hyph will try to simply recalculate the syllable count for each word, by counting the hyphenation marks (and adding 1 to the number). This can be usefull if you changed hyphenation some other way, e.g. in a spreadsheet GUI, but don't want to have to correct the syllable count yourself as well.

Value

An object of the same class as obj.

See Also

kRp.tagged-class, treetag, kRp.POS.tags.

Examples

```
## Not run:
tagged.txt <- correct.tag(tagged.txt, row=21, tag="NN")

hyphenated.txt <- correct.hyph(hyphenated.txt, "Hilfe", "Hil-fe")

## End(Not run)
```

---

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.tagged'
cTest(obj, every = 2, min.length = 3,
  intact = c(start = 1, end = 1), replace.by = "_")
```

Arguments

`...`	Additional arguments to the method (as described in this document).
`obj`	An object of class "kRp.tagged"
`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

And object of class kRp.tagged, with an additional list `cTest` in its `desc` slot, listing the words which were changed.

`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 either class `kRp.tagged-class` or `kRp.analysis-class`, 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-class`.

See Also

`kRp.POS.tags`, `kRp.tagged-class`, `kRp.TTR-class`

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.tagged-class, 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-class.

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.tagged-class`, 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-class`.

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.tagged-class`, 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-class`.

Examples

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

## End(Not run)
```

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.tagged-class`, 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-class`.

Examples

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

## End(Not run)
```

ELF *Readability: Farr'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.tagged-class`, 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 Farr'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-class`.

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.tagged-class`, 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-class`.

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)
```

`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.tagged-class`, 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"`, `"nl"`, `"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), `"de"` 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-class`.

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.tagged-class`, 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-class`.

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.tagged-class`, 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-class`.

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.tagged-class, 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-class.

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, corp.freq = NULL, desc.stat = TRUE,
  force.lang = NULL, tagger = "kRp.env", corp.rm.class = "nonpunct",
  corp.rm.tag = c(), tfidf = TRUE, ...)

kRp.freq.analysis(txt.file, corp.freq = NULL, desc.stat = TRUE,
  force.lang = NULL, tagger = "kRp.env", corp.rm.class = "nonpunct",
  corp.rm.tag = c(), ...)
```

Arguments

`txt.file`	Either an object of class `kRp.tagged-class`, `kRp.txt.freq-class`, `kRp.analysis-class` or `kRp.txt.trans-class`, or a character vector which must be a valid path to a file containing the text to be analyzed.
`corp.freq`	An object of class `kRp.corp.freq-class`.
`desc.stat`	Logical, whether a descriptive statistical analysis should be performed.
`force.lang`	A character string defining the language to be assumed for the text, by force.
`tagger`	A character string defining the tokenizer/tagger command you want to use for basic text analysis. Can be omitted if `txt.file` is already of class `kRp.tagged-class`. Defaults to `"kRp.env"` to get the settings by `get.kRp.env`. Set to `"tokenize"` to use `tokenize`.
`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, 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.
`tfidf`	Logical, whether the term frequency–inverse document frequency statistic (tf-idf) should be computed. Requires `corp.freq` to provide appropriate idf values for the types in `txt.file`. Missing idf values will result in `NA`.
`...`	Additional options to be passed through to the function defined with `tagger`.

Details

The easiest way to see what kinds of analyses are done is probably to look at the slot description of kRp.txt.freq-class.

By default, if the text has yet to be tagged, the language definition is queried by calling get.kRp.env(lang=TRUE) internally. Or, if txt.file 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

An object of class kRp.txt.freq-class.

Note

Prior to koRpus 0.04-29, this function was named kRp.freq.analysis(). For backwards compatibility there is a wrapper function, but it should be considered deprecated.

See Also

get.kRp.env, kRp.tagged-class, kRp.corp.freq-class

Examples

```
## Not run:
freq.analysis("~/some/text.txt", corp.freq=my.LCC.data)

## End(Not run)
```

---

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.tagged-class, 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"). 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-class`.

References

Fucks, W. (1955). Der Unterschied des Prosastils von Dichtern und anderen Schriftstellern. *Sprachforum*, 1, 233–244.

Examples

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

## End(Not run)
```

---

get.kRp.env *Get koRpus session environment*

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.
`errorIfUnset`	Logical, if `TRUE` and the desired property is not set at all, the function will fail with an error message.

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

```
## Not run:
set.kRp.env(TT.cmd="~/bin/treetagger/cmd/tree-tagger-german", lang="de")
get.kRp.env(TT.cmd=TRUE)

## End(Not run)
```

---

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-class.

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 http://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/Issues/Pages/UDHRIndex.aspx

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

Examples

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

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 either class `kRp.tagged-class` or `kRp.analysis-class`, 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-class`.

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.tagged-class, kRp.TTR-class

Examples

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

## End(Not run)
```

---

hyph.XX *Hyphenation patterns*

Description

Hyphenation patterns for hyphen() in several languages. Replace "XX" with "de", "de.old", "en", "es", "fr", "it", "en.us", or "ru", respectively. These data objects are not really intended to be used directly, but rather to be consulted by the hyphen() function without further user interaction.

Usage

```
hyph.XX
```

Format

The pattern slot of each hyphenation pattern object has three colums:

orig The original pattern in patgen style format.

char Only the character elements of the pattern which can be matched to parts of an actual word.

nums A code of digits defining the possibility to split syllables at respective places in this pattern.

Source

The patterns (as they are present in the "orig" column described above) were originally provided by the LaTeX developers[1], under the terms of the LaTeX Project Public License[2]. Refer to Liang (1983) for a detailed explaination – it's fascinating. From these original patterns the values in the remaining columns were created (using read.hyph.pat).

Note: Some of the original patterns might have been altered before, too. This is because the use cases slightly differ between LaTeX and this package. For accurate hyphenation in text processing it's important to not split words in wrong ways, rather than identifying all possible split points. I therefore understand that in general these patterns might lead to slightly more conservative results. This package, however, uses the same algorithm to count syllables, that is, it's more important to just get the right sum of syllables in a word, and actually quite irrelevant (for the time being) which actual syllables were identified. That is, for this purpose even completely wrong suggestions wouldn't do any harm, as long as the overall number is correct, whereas perfectly valid hyphenations would be misleading if they are missing one or more possibilities. If any such changes to the patterns have been made, they are fully documented in the file "ChangeLog_hyph_patterns.txt" in the sources for this package. The unchanged original patterns can be found under [1].

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

---

hyphen *Automatic hyphenation*

Description

This function implements word hyphenation, based on Liang's algorithm.

Usage

```
hyphen(words, hyph.pattern = NULL, min.length = 3, rm.hyph = TRUE,
  corp.rm.class = "nonpunct", corp.rm.tag = c(), quiet = FALSE,
  cache = TRUE)
```

Arguments

words	Either an object of class kRp.tagged-class, kRp.txt.freq-class or kRp.analysis-class, or a character vector with words to be hyphenated.
hyph.pattern	Either an object of class kRp.hyph.pat-class, 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.
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, c("punct","sentc" 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 hyphen.cache.file in set.kRp.env.

Details

For this to work the function must be told which pattern set it should use to find the right hyphenation spots. If `words` is already a tagged object, its language definition might be used. Otherwise, in addition to the words to be processed you must specify `hyph.pattern`. You have two options: If you want to use one of the built-in language patterns, just set it to the according language abbrevation. As of this version valid choices are:

- `"de"` — German (new spelling, since 1996)
- `"de.old"` — German (old spelling, 1901–1996)
- `"en"` — English (UK)
- `"en.us"` — English (US)
- `"es"` — Spanish
- `"fr"` — French
- `"it"` — Italian
- `"ru"` — Russian

In case you'd rather use your own pattern set, `hyph.pattern` can be an object of class `kRp.hyph.pat`, alternatively.

The built-in hyphenation patterns were derived from the patterns available on CTAN[1] under the terms of the LaTeX Project Public License[2], see `hyph.XX` for detailed information.

Value

An object of class `kRp.hyphen-class`

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`

jumbleWords *Produce jumbled words*

Description

This function takes either a character vector or objects inheriting class kRp.tagged (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, min.length = 3, intact = c(start = 1, end = 1))
```

Arguments

words	Either a character vector or an object inheriting from class kRp.tagged.
min.length	An integer value, defining the minimum word length. Words with less characters will not be changed.
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 tagged text object.

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 either class kRp.tagged-class or kRp.analysis-class, 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-class`.

See Also

`kRp.POS.tags`, `kRp.tagged-class`, `kRp.TTR-class`

Examples

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

## End(Not run)
```

---

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

Description

This class is used for objects that are returned by `kRp.text.analysis`.

Slots

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

`TT.res` A commented verion of the fully tagged text. Depending on input data, this is identical to the slot `TT.res` of function `treetag` or `freq.analysis`.

`desc` Descriptive statistics

`lex.div` Information on lexical diversity

`freq.analysis` Information on the word frequencies of the analyzed text.

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 (dee 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.

References

[1] http://corpora.informatik.uni-leipzig.de/download.html [2] http://celex.mpi.nl

---

`kRp.filter.wclass` *Remove word classes*

Description

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

Usage

```
kRp.filter.wclass(txt, corp.rm.class = "nonpunct", corp.rm.tag = c(),
  as.vector = FALSE)
```

Arguments

txt	An object of class kRp.tagged-class.
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, c("punct","s 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.

Value

An object of class kRp.tagged-class. If as.vector=TRUE, returns only a character vector.

See Also

kRp.POS.tags

Examples

```
## Not run:
   kRp.filter.wclass(tagged.text)

## End(Not run)
```

---

kRp.hyph.pat,-class *S4 Class kRp.hyph.pat*

Description

This class is used for objects that are returned by read.hyph.pat.

Slots

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

pattern A matrix with three colums:

- orig: The unchanged patgen patterns.
- char: Only the characters used for matching.
- nums: The hyphenation number code for the pattern.

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

Description

This class is used for objects that are returned by hyphen.

Slots

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

`desc` Descriptive statistics of the analyzed text.

`hyphen` A data.frame with two columns:

`syll`: Number of recognized syllables

`word`: The hyphenated word

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 a short 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.

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

Currently supported languages are:

- "de" — German, according to the STTS guidelines (Schiller, Teufel, Stockert, & Thielen, 1995)
- "en" — English, according to the Penn Treebank guidelines (Santorini, 1991)
- "es" — Spanish, according to http://www.ims.uni-stuttgart.de/ftp/pub/corpora/spanish-tagset.txt
- "fr" — French, according to http://www.ims.uni-stuttgart.de/~schmid/french-tagset.html
- "it" — Italian, according to ftp://ftp.ims.uni-stuttgart.de/pub/corpora/italian-tagset.txt and http://sslmit.unibo.it/~baroni/collocazioni/itwac.tagset.txt, respectively
- "ru" — Russian, according to the MSD tagset by Sharoff, Kopotev, Erjavec, Feldman & Divjak (2008)

For the internal tokenizer a small subset of tags is also defined, available through lang="kRp". 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.

Author(s)

m.eik michalke <meik.michalke@hhu.de>, support for Spanish contributed by Earl Brown <eabrown@csumb.edu>, support for Italian contributed by Alberto Mirisola.

References

Santorini, B. (1991). *Part-of-Speech Tagging Guidelines for the Penn Treebank Project*. URL: http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/Penn-Treebank-Tagset.pdf

Schiller, A., Teufel, S., Stockert, C. & Thielen, C. (1995). *Vorl\"aufge Guidelines f\"ur das Tagging deutscher Textcorpora mit STTS*. URL: http://www.ims.uni-stuttgart.de/ftp/pub/corpora/stts_guide.pdf

Sharoff, S., Kopotev, M., Erjavec, T., Feldman, A. & Divjak, D. (2008). *Designing and evaluating Russian tagsets*. In: Proc. LREC 2008, Marrakech. URL: http://corpus.leeds.ac.uk/mocky/

See Also

`get.kRp.env`

Examples

```
tags.de <- kRp.POS.tags("de")
```

---

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.

`TT.res` The tokenized and POS-tagged text. See `kRp.tagged-class` 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.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.

`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.

`kRp.tagged,-class` *S4 Class kRp.tagged*

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.

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

- `token`: The tokenized text.
- `tag`: POS tags for each token.
- `lemma`: Lemma for each token.
- `lttr`: Number of letters.
- `wclass`: Word class.
- `desc`: A short description of the POS tag.
- `stop`: Logical, `TRUE` if token is a stopword.
- `stem`: Stemmed token.

Note

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

---

kRp.text.analysis *Analyze texts using TreeTagger and word frequencies*

Description

The function `kRp.text.analysis` analyzes texts in various ways.

Usage

```
kRp.text.analysis(txt.file, tagger = "kRp.env", force.lang = NULL,
  desc.stat = TRUE, lex.div = TRUE, corp.freq = NULL,
  corp.rm.class = "nonpunct", corp.rm.tag = c(), ...)
```

Arguments

`txt.file`	Either an object of class `kRp.tagged-class`, `kRp.txt.freq-class`, `kRp.analysis-class` or `kRp.txt.trans-class`, or a character vector which must be be a valid path to a file containing the text to be analyzed.
`tagger`	A character string defining the tokenizer/tagger command you want to use for basic text analysis. Can be omitted if `txt.file` is already of class `kRp.tagged-class`. Defaults to `"kRp.env"` to get the settings by `get.kRp.env`. Set to `"tokenize"` to use `tokenize`.
`force.lang`	A character string defining the language to be assumed for the text, by force.
`desc.stat`	Logical, whether a descriptive statistical analysis should be performed.
`lex.div`	Logical, whether some lexical diversity analysis should be performed, using `lex.div`.

corp.freq	An object of class kRp.corp.freq-class. If present, a frequency index for the analyzed text is computed (see details).
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, 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.
...	Additional options to be passed through to the function defined with tagger.

Details

The function is basically a wrapper for treetag(),freq.analysis() and lex.div().

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.file 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

An object of class kRp.analysis-class.

References

[1] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html

See Also

set.kRp.env, get.kRp.env, kRp.POS.tags, lex.div

Examples

```
## Not run:
kRp.text.analysis("/some/text.txt")

## End(Not run)
```

---

kRp.text.paste *Paste koRpus objects*

Description

Paste the text in koRpus objects.

Usage

```
kRp.text.paste(txt, replace = c(hon.kRp = "", hoff.kRp = "\n\n", p.kRp =
  "\n\n"))
```

Arguments

txt	An object of class kRp.txt.trans-class, kRp.tagged-class, kRp.txt.freq-class or kRp.analysis-class.
replace	A named character vector to define replacements for koRpus' internal headline and paragraph tags.

Details

This function takes objects of either class kRp.tagged, kRp.txt.freq or kRp.analysis and pastes only the actual text as is.

Value

An atomic character vector.

Examples

```
## Not run:
tagged.text.obj <- freq.analysis("/some/text.txt", corp.freq=my.LCC.data)
kRp.text.paste(tagged.text.obj)

## End(Not run)
```

---

kRp.text.transform *Letter case transformation*

Description

Transforms text in koRpus objects token by token.

Usage

```
kRp.text.transform(txt, scheme, p = 0.5, paste = FALSE)
```

Arguments

txt	An object of class kRp.txt.trans-class, kRp.tagged-class, kRp.txt.freq-class or kRp.analysis-class.
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".

p Numeric value between 0 and 1. Defines the probability for upper case letters (relevant only if scheme="random").

paste Logical, see value section.

Details

This function is mainly intended to produce text material for experiments.

Value

By default an object of class kRp.txt.trans-class is returned. If paste=TRUE, returns an atomic character vector (via kRp.text.paste).

Examples

```
## Not run:
tagged.text.obj <- freq.analysis("/some/text.txt", corp.freq=my.LCC.data)
kRp.text.transform(tagged.text.obj, scheme="random", paste=TRUE)

## End(Not run)
```

---

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 six elements ("tokens", "types", "lemmas", "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 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.

`kRp.txt.freq,-class` *S4 Class kRp.txt.freq*

Description

This class is used for objects that are returned by `freq.analysis`.

Slots

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

`TT.res` A data.frame with a version of the fully tagged text (like `TT.res` in class `koRpus.tagged`, plus frequency data).

`desc` A list with detailed descriptive statistics on the analyzed text.

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

`kRp.txt.trans,-class` *S4 Class kRp.txt.trans*

Description

This class is used for objects that are returned by `kRp.text.transform`.

Slots

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

`desc` Descriptive statistics of the tagged text.

`TT.res` A data.frame with the fully tagged and transformed text (like `TT.res` in class `koRpus.tagged`, plus the new columns `token.old` and `equal`).

`diff` A list with 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.

lex.div *Analyze lexical diversity*

Description

This function analyzes the lexical diversity/complexity of a text corpus.

Usage

```
lex.div(txt, segment = 100, factor.size = 0.72, min.tokens = 9,
  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,
  corp.rm.class = "nonpunct", corp.rm.tag = c(), quiet = FALSE)
```

Arguments

txt	An object of either class kRp.tagged-class, kRp.txt.freq-class, kRp.analysis-class or kRp.txt.trans-class, containing the tagged text to be analyzed.
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.
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".
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 stepwidth 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.
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, c("punct","s to be used.
corp.rm.tag	A character vector with POS tags which should be dropped.
quiet	Logical. If FALSE, short status messages will be shown. TRUE will also suppress all potential warnings regarding the validation status of measures.

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

An object of class `kRp.TTR-class`.

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.tagged-class`, `kRp.TTR-class`

Examples

```
## Not run:
lex.div(tagged.text)

## End(Not run)
```

`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-class`.

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(104, 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.tagged-class`, 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-class`.

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.tagged-class`, 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-class`.

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 either class kRp.tagged-class or kRp.analysis-class, 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-class.

See Also

kRp.POS.tags, kRp.tagged-class, kRp.TTR-class

Examples

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

## End(Not run)
```

manage.hyph.pat *Handling hyphenation pattern objects*

Description

This function can be used to examine and change hyphenation pattern objects be used with hyphen.

Usage

```
manage.hyph.pat(hyph.pattern, get = NULL, set = NULL, rm = NULL,
  word = NULL, min.length = 3, rm.hyph = TRUE)
```

Arguments

hyph.pattern	Either an object of class kRp.hyph.pat, or a valid language abbreviation for patterns included in this package.
get	A character string, part of a word to look up in the pattern set, i.e., without the numbers indicating split probability.
set	A character string, a full pattern to be added to the pattern set, i.e., including the numbers indicating split probability.
rm	A character string, part of a word to remove from the pattern set, i.e., without the numbers indicating split probability.
word	A character string, a full word to hyphenate using the given pattern set.
min.length	Integer, number of letters a word must have for considering a hyphenation.
rm.hyph	Logical, whether appearing hyphens in words should be removed before pattern matching.

Details

You can only run one of the possible actions at a time. If any of these arguments is not NULL, the corresponding action is done in the following order, and every additional discarded:

- getSearches the pattern set for a given word part
- setAdds or replaces a pattern in the set (duplicates are removed)
- rmRemoves a word part and its pattern from the set
- wordHyphenates a word and returns all parts examined as well as all matching patterns

If all action arguments are NULL, manage.hyph.pat returns the full pattern object.

Value

If all action arguments are NULL, returns an object of class kRp.hyph.pat-class. The same is true if set or rm are set and hyph.pattern is itself an object of that class; if you refer to a language instead, pattern changes will be done internally for the running session and take effect immediately. The get argument will return a caracter vector, and word a data frame.

References

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

See Also

kRp.hyph.pat-class, hyphen

Examples

```
## Not run:
manage.hyph.pat("en", set="r3ticl")
manage.hyph.pat("en", get="rticl")
manage.hyph.pat("en", word="article")
manage.hyph.pat("en", rm="rticl")

## 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 either class kRp.tagged-class or kRp.analysis-class, 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-class.

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.tagged-class, kRp.TTR-class

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 either class kRp.tagged-class or kRp.analysis-class, 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-class.

See Also

kRp.POS.tags, kRp.tagged-class, kRp.TTR-class

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, ...)
```

Arguments

`txt`	An object of either class `kRp.tagged-class` or `kRp.analysis-class`, 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.
`...`	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-class`.

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.tagged-class, kRp.TTR-class

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.tagged-class, 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-class`.

References

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

Examples

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

## End(Not run)
```

---

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

Description

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

Usage

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

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

Arguments

`x`	An object of class `kRp.tagged`
`what`	Character string, valid options are: `"wclass"`**:** Barplot of distribution of word classes `"letters"`**:** Line plot of distribution of word length in letters
`y`	the y coordinates of points in the plot, *optional* if `x` is an appropriate structure.
`...`	Arguments to be passed to methods, such as graphical parameters (see `par`). Many methods will accept the following arguments: `type` what type of plot should be drawn. Possible types are • `"p"` for **p**oints,

- "l" for **l**ines,
- "b" for **b**oth,
- "c" for the lines part alone of "b",
- "o" for both '**o**verplotted',
- "h" for '**h**istogram' like (or 'high-density') vertical lines,
- "s" for stair **s**teps,
- "S" for other **s**teps, see 'Details' below,
- "n" for no plotting.

All other types give a warning or an error; using, e.g., type = "punkte" being equivalent to type = "p" for S compatibility. Note that some methods, e.g. plot.factor, do not accept this.

main an overall title for the plot: see title.

sub a sub title for the plot: see title.

xlab a title for the x axis: see title.

ylab a title for the y axis: see title.

asp the y/x aspect ratio, see plot.window.

See Also

kRp.tagged-class

Examples

```
## Not run:
tagged.results <- treetag("~/my.data/sample_text.txt", treetagger="manual", lang="en",
   TT.options=list(path="~/bin/treetagger", preset="en"))
plot(tagged.results)

## End(Not run)
```

---

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

Description

The method query returns query information from objects of classes kRp.corp.freq-class and kRp.tagged-class.

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)
```

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

Arguments

obj	An object of class kRp.corp.freq-class.
var	A character string naming a variable in the object (i.e., colname). If set to "regexp", grepl is called on the word column of corpus frequency objects.
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.
ignore.case	Logical, passed through to grepl if var="regexp".
perl	Logical, passed through to grepl if var="regexp".
...	Optional arguments, see above.

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.tagged: var can be any of the variables in slot TT.res. For rel currently only "eq" and "num" are implemented. The latter isn't a relation, but will return a vector with the row numbers in which the query was found.

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-class, subset

Examples

```
## Not run:
# look up frequencies for the word "aber"
query(LCC.data, var="word", query="aber")

# show all entries with a frequency of exactly 3000 in the corpus
query(LCC.data, "freq", 3000)

# now, which words appear more than 40000 times in a million?
query(LCC.data, "pmio", 40000, "gt")

# example for a range request: words with a log10 between 2 and 2.1
# (including these two values)
query(LCC.data, "log10", c(2, 2.1))
# (and without them)
query(LCC.data, "log10", c(2, 2.1), "gt")

# example for a list of queries: get words with a frequency between
# 700 and 750 per million and at least five letters
query(LCC.data, query=list(
  list(pmio=c(700,750)),
  list(lttr=5, rel="ge"))
)

# get all "he" lemmata in a previously tagged text object
query(tagged.txt, "lemma", "he")

## End(Not run)
```

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 either class kRp.tagged-class or kRp.analysis-class, 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-class.

See Also

kRp.POS.tags, kRp.tagged-class, kRp.TTR-class

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-class.

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 neccessary.

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-class.

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.

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

See Also

kRp.corp.freq-class, query, kRp.text.analysis

Examples

```
## Not run:
bawl.corp <- read.BAWL("~/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)
```

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.

Value

An object of class kRp.corp.freq-class.

References

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

See Also

kRp.corp.freq-class

Examples

```
## Not run:
my.Celex.data <- read.corp.celex("~/mydata/Celex/GERMAN/GFW/GFW.CD",
      running.words=5952000)
freq.analysis("/some/text.txt", 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-class.

Usage

```
read.corp.custom(corpus, format = "file", fileEncoding = "UTF-8",
  quiet = TRUE, caseSens = TRUE, log.base = 10, ...)
```

Arguments

corpus	Either the path to directory with txt files to read and analyze, or a vector object already holding the text corpus. Can also be an already tokenized and tagged text object which inherits class kRp.tagged (then the column "token" of the "TT.res" slot is used).
format	Either "file" or "obj", depending on whether you want to scan files or analyze the given object.
fileEncoding	A character string naming the encoding of the corpus files.
quiet	Logical. If FALSE, short status messages will be shown.
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 to be passed through to the tokenize function.

Details

The function 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-class, so it can be used for frequency analysis by other functions of this package.

Value

An object of class kRp.corp.freq-class.

See Also

kRp.corp.freq-class

Examples

```
## Not run:
ru.corp <- read.corp.custom("~/mydata/corpora/russian_corpus/")

## End(Not run)
```

---

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)
```

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).

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.

Value

An object of class kRp.corp.freq-class.

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] http://corpora.informatik.uni-leipzig.de/download.html

See Also

kRp.corp.freq-class

Examples

```
## Not run:
# old format .zip archive
my.LCC.data <- read.corp.LCC("~/mydata/corpora/de05_3M.zip")
# new format tar archive
my.LCC.data <- read.corp.LCC("~/mydata/corpora/rus_web_2002_300K-text.tar")
```

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

tagged.results <- treetag("/some/text.txt")
freq.analysis(tagged.results, corp.freq=my.LCC.data)

## End(Not run)
```

`read.hyph.pat` *Reading patgen-compatible hyphenation pattern files*

Description

This function reads hyphenation pattern files, to be used with hyphen.

Usage

```
read.hyph.pat(file, lang, fileEncoding = "UTF-8")
```

Arguments

`file`	A character string with a valid path to a file with hyphenation patterns (one pattern per line).
`lang`	A character string, usually two letters short, naming the language the patterns are meant to be used with (e.g. "es" for Spanish).
`fileEncoding`	A character string defining the character encoding of the file to be read. Unless you have a really good reason to do otherwise, your pattern files should all be UTF-8 encoded.

Details

Hyphenation patterns that can be used are available from CTAN[1]. But actually any file with only the patterns themselves, one per line, should work.

The language designation is of no direct consequence here, but if the resulting pattern object is to be used by other functions in this package, it should resamble the designation that's used for the same language there.

Value

An object of class `kRp.hyph.pat-class`.

References

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

See Also

hyphen, manage.hyph.pat

Examples

```
## Not run:
read.hyph.pat("~/patterns/hyph-en-us.pat.txt", lang="en_us")

## End(Not run)
```

---

read.tagged *Import already tagged texts*

Description

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

Usage

```
read.tagged(file, lang = "kRp.env", encoding = NULL,
  tagger = "TreeTagger", apply.sentc.end = TRUE, sentc.end = c(".", "!",
  "?", ";", ":"), stopwords = NULL, stemmer = NULL, rm.sgml = TRUE)
```

Arguments

file	Either a connection or a character vector, valid path to a file, containing the previously analyzed text.
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.
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.
tagger	The software which was used to tokenize and tag the text. Currently, TreeTagger is the only supported tagger.
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

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.tagged-class. 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] http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html

See Also

treetag, freq.analysis, get.kRp.env, kRp.tagged-class

Examples

```
## Not run:
tagged.results <- read.tagged("~/my.data/tagged_speech.txt", lang="en")

## End(Not run)
```

---

readability *Measure readability*

Description

This function calculates several readability indices.

Usage

```
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",
```

```
  "Wheeler.Smith"),
parameters=list(),
word.lists=list(Bormuth=NULL, Dale.Chall=NULL, Harris.Jacobson=NULL,
  Spache=NULL),
fileEncoding="UTF-8", tagger="kRp.env", force.lang=NULL,
sentc.tag="sentc", nonword.class="nonpunct", nonword.tag=c(),
quiet=FALSE, ...)
```

Arguments

txt.file	Either an object of class kRp.tagged-class, kRp.txt.freq-class, kRp.analysis-class or kRp.txt.trans-class, or a character vector which must be be a valid path to a file containing the text to be analyzed. If the latter, force.lang must be set as well, and the language specified must be supported by both treetag and hyphen
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.
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 naming the encoding of the word list files (if they are files). "ISO_8859-1" or "UTF-8" should work in most cases.
tagger	A character string pointing to the tokenizer/tagger command you want to use for basic text analysis. Can be omitted if txt.file is already of class kRp.tagged-class. Defaults to tagger="kRp.env" to get the settings by get.kRp.env. Set to "tokenize" to use tokenize.
force.lang	A character string defining the language to be assumed for the text, by force.
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", l 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, 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.
...	Additional options for the specified tagger function

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:

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

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:

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

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

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

Wrapper function: `farr.jenkins.paterson`

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

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

Certain internationalisations of the parameters are also implemented. They can be used by setting the `Flesch` parameter to `"es"` (Fernandez-Huerta), `"es-s"` (Szigriszt), `"nl"` (Douma), `"de"` (Amstad's Verst\"andlichkeitsindex), or `"fr"` (Kandel-Moles). If `parameters` is set to `Flesch="PSK"`, the revised parameters by Powers-Sumner-Kearl (1958) are used to calculate a grade level:

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

Wrapper function: `flesch`

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

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

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.

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*:

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

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"` Bj\"ornsson's *L\"asbarhetsindex*. Originally proposed for Swedish texts, calculated by:

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

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:

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

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]:

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

Wrapper function: `strain`

`"Traenkle.Bailer"`: *Tr\"anke-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 (Tr\"ankle, & 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`

`"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

An object of class `kRp.readability-class`.

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.

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 f\"ur die deutsche Sprache. *Zeitschrift f\"ur Entwicklungspsychologie und P\"adagogische 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.

Fucks, W. (1955). Der Unterschied des Prosastils von Dichtern und anderen Schriftstellern. *Sprachforum*, 1, 233–244.

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.

Tr\"ankle, U. & Bailer, H. (1984). Kreuzvalidierung und Neuberechnung von Lesbarkeitsformeln f\"ur die deutsche Sprache. *Zeitschrift f\"ur Entwicklungspsychologie und P\"adagogische 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] http://strainindex.wordpress.com/2007/09/25/hello-world/

`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),
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", "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".

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.

... — Additional options, see readability.

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),
 syll=c(all=974, s1=316, s2=116),
 punct=78,
 all.chars=3553,
 prepositions=74,
 conjunctions=18,
 pronouns=9,
 foreign=0,
 TTR=0.5269784,
 Bormuth=192,
 Dale.Chall=192,
 Harris.Jacobson=240,
 Spache=240)

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

## 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.tagged-class, 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-class.

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 either class kRp.tagged-class or kRp.analysis-class, 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-class.

See Also

kRp.POS.tags, kRp.tagged-class, kRp.TTR-class

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 preffered, 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 environmenton*

Description

The function `set.kRp.env` can be called once 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 a hidden environment.

Usage

```
set.kRp.env(...)
```

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`.

To explicitly unset a value again, set it to an empty character string (e.g., `lang=""`).

Details

To get the contents of the hitten environment, the function `get.kRp.env` can be used.

Value

Returns an invisible `NULL`.

See Also

`get.kRp.env`

Examples

```
## Not run:
set.kRp.env(TT.cmd="~/bin/treetagger/cmd/tree-tagger-german", lang="de")
get.kRp.env(TT.cmd=TRUE)

## End(Not run)
```

---

show,kRp.corp.freq-method

Show methods for koRpus objects

Description

Show methods for S4 objects of classes `kRp.lang-class`, `kRp.readability-class`, `kRp.corp.freq-class` or `kRp.TTR-class`.

Usage

```
## S4 method for signature 'kRp.corp.freq'
show(object)

show(object)

## S4 method for signature 'kRp.lang'
show(object)

## S4 method for signature 'kRp.readability'
show(object)

## S4 method for signature 'kRp.TTR'
show(object)
```

Arguments

object	An object of class kRp.lang, kRp.readability, kRp.corp.freq or kRp.TTR.

See Also

kRp.lang-class, kRp.readability-class, kRp.corp.freq-class, kRp.TTR-class

Examples

```
## Not run:
  guess.lang("/home/user/data/some.txt", udhr.path="/home/user/data/udhr_txt/")

## End(Not run)
## Not run:
flesch(tagged.txt)

## End(Not run)
## Not run:
MTLD(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.tagged-class, 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-class`.

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.tagged-class`, 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-class.

Examples

```
## Not run:
spache(tagged.text, word.list=spache.revised.wl)

## 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.tagged-class, 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-class.

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-class, kRp.readability-class, kRp.tagged-class, kRp.TTR-class or kRp.txt.freq-class.

Usage

```
summary(object, ...)

## S4 method for signature 'kRp.readability'
summary(object, flat = FALSE)

## S4 method for signature 'kRp.tagged'
summary(object)

## S4 method for signature 'kRp.TTR'
summary(object)

## S4 method for signature 'kRp.txt.freq'
summary(object)
```

Arguments

object	An object of class kRp.lang, kRp.readability, kRp.tagged, kRp.TTR or kRp.txt.freq.
flat	Logical, if TRUE only a named vector of main results is returned
...	additional arguments affecting the summary produced.

See Also

kRp.lang-class, kRp.readability-class, kRp.tagged-class, kRp.TTR-class, kRp.txt.freq-class

Examples

```
## Not run:
summary(guess.lang("/home/user/data/some.txt", udhr.path="/home/user/data/udhr_txt/"))

## End(Not run)
## Not run:
summary(flesch(tagged.txt))

## End(Not run)
## Not run:
tagged.results <- treetag("~/my.data/sample_text.txt", treetagger="manual", lang="en",
   TT.options=list(path="~/bin/treetagger", preset="en"))
summary(tagged.results)

## End(Not run)
## Not run:
summary(lex.div(tagged.txt))

## End(Not run)
## Not run:
summary(freq.analysis(tagged.txt))

## End(Not run)
```

---

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)

## S4 method for signature 'kRp.taggedText'
taggedText(obj)

taggedText(obj) <- value

## S4 replacement method for signature 'kRp.taggedText'
taggedText(obj) <- value

describe(obj)

## S4 method for signature 'kRp.taggedText'
describe(obj)
```

```
describe(obj) <- value

## S4 replacement method for signature 'kRp.taggedText'
describe(obj) <- value

## S4 method for signature 'kRp.hyphen'
describe(obj)

## S4 replacement method for signature 'kRp.hyphen'
describe(obj) <- value

language(obj)

## S4 method for signature 'kRp.taggedText'
language(obj)

language(obj) <- value

## S4 replacement method for signature 'kRp.taggedText'
language(obj) <- value

## S4 method for signature 'kRp.hyphen'
language(obj)

## S4 replacement method for signature 'kRp.hyphen'
language(obj) <- value

is.taggedText(obj)

hyphenText(obj)

## S4 method for signature 'kRp.hyphen'
hyphenText(obj)

hyphenText(obj) <- value

## S4 replacement method for signature 'kRp.hyphen'
hyphenText(obj) <- value
```

Arguments

value	The new value to replace the current with.
obj	An arbitrary R object.

Details

- taggedText() returns the TT.res slot.

- describe() returns the desc slot.
- language() returns the lang slot.
- hyphenText() returns the hyphen slot from objects of class kRp.hyphen.

Examples

```
## Not run:
taggedText(tagged.txt)

## End(Not run)
```

---

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.tagged-class, kRp.txt.freq-class or kRp.analysis-class. 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-class, 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

```
## Not run:
set.kRp.env(TT.cmd="manual", lang="en", TT.options=list(path="~/bin/treetagger",
      preset="en"))
tagged.txt <- treetag("example_text.txt")
tagged.txt.features <- textFeatures(tagged.txt)

## End(Not run)
```

---

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("'",
  "'"), suf = c("'", "'")), 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)
```

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.
`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.tagged.

lang — A character string naming the language of the analyzed text. If set to "kRp.env" this is got 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.

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, kRp.text.paste 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.tagged-class.

Examples

```
## Not run:
tokenized.obj <- tokenize("~/mydata/corpora/russian_corpus/")

## 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("~/my.data/speech.txt",
   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("~/my.data/speech.txt",
   clean.raw=list("(A)([[:digit:]]{2})"="B\\2"),
   perl=TRUE)

## enabling stopword detection and stemming
# if you also installed the packages tm and Snowball,
# you can use some of their features with koRpus:
tokenized.obj <- tokenize("~/my.data/speech.txt",
   stopwords=tm::stopwords("en"),
   stemmer=SnowballC::wordStem)

# removing all stopwords now is simple:
tokenized.noStopWords <- kRp.filter.wclass(tokenized.obj, "stopword")

## End(Not run)
```

---

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.tagged-class, 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-class.

Examples

```
## Not run:
  traenkle.bailer(tagged.text)

## End(Not run)
```

---

treetag *A function to call TreeTagger*

Description

This function 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)
```

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 for all supported languages. If set to "kRp.env" this is got 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 here:
  - "de-utf8" German, UTF-8
  - "de" German
  - "en" English
  - "es-utf8" Spanish, UTF-8
  - "es" Spanish
  - "fr-utf8" French, UTF-8
  - "fr" French
  - "it-utf8" Italian, UTF-8
  - "it" Italian
  - "ru" Russian, UTF-8

  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.
- 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.
- `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/`.

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.

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.tagged-class. 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.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html

See Also

freq.analysis, get.kRp.env, kRp.tagged-class

Examples

```
## Not run:
# first way to invoke POS tagging, using a built-in preset:
tagged.results <- treetag("~/my.data/speech.txt", treetagger="manual", lang="en",
   TT.options=list(path="~/bin/treetagger", preset="en"))
# second way, use one of the batch scripts that come with TreeTagger:
tagged.results <- treetag("~/my.data/speech.txt",
   treetagger="~/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="~/bin/treetagger/cmd/tree-tagger-english", lang="en")
tagged.results <- treetag("~/my.data/speech.txt")

# 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="~/bin/treetagger",
   preset="en"))
tagged.results <- treetag("~/my.data/speech.txt",
   stopwords=tm::stopwords("en"),
   stemmer=SnowballC::wordStem)

# removing all stopwords now is simple:
tagged.noStopWords <- kRp.filter.wclass(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.tagged-class, 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-class.

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 either class `kRp.tagged-class` or `kRp.analysis-class`, 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-class`.

See Also

`kRp.POS.tags`, `kRp.tagged-class`, `kRp.TTR-class`

Examples

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

## End(Not run)
```

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 either class kRp.tagged-class or kRp.analysis-class, 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-class.

See Also

kRp.POS.tags, kRp.tagged-class, kRp.TTR-class

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.tagged-class`, 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-class`.

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