Package 'stylo'

October 19, 2014

Type Package

Title Functions for a Variety of Stylometric Analyses

Version 0.5.8-2

Date 2014-10-19

Author Maciej Eder, Jan Rybicki, Mike Kestemont

Maintainer Maciej Eder <maciejeder@gmail.com>

URL https://sites.google.com/site/computationalstylistics/

Imports tcltk, tcltk2, ape, pamr, e1071, class, lattice, tsne

Description A number of functions, supplemented by GUI, to perform various analyses in the field of computational stylistics, authorship attribution, etc.

License GPL (>= 3)

NeedsCompilation no

Repository CRAN

Date/Publication 2014-10-19 11:53:15

R topics documented:

`assign.plot.colors` *Assign colors to samples*

Description

Function that assigns unique colors to each class represented in a corpus: used for graph auto-coloring.

Usage

```
assign.plot.colors(labels, col = "colors", opacity = 1)
```

Arguments

`labels`	a vector containing the names of the samples in a corpus; it is obligatory to use an underscore as a class delimiter. Consider the following examples: c("Sterne_Tristram", "Sterne_Sentimental", "Fielding_Tom", ...), where the classes are the authors' names, and c("M_Joyce_Dubliners", "F_Woolf_Night_and_day", "M_Conrad_Lord_Jim", ...), where the classes are M(ale) and F(emale) according to authors' gender. Note that only the part up to the first underscore in the sample's name will be included in the class label.
`col`	an optional argument specifying the color palette to be used: "colors" for full-color output (default), "greyscale" for greyscale (useful for preparing publishable pictures), and "black", if no colors should be used.

opacity — optional argument to set transparency/opacity of the colors. 0 means full transparency, 1 means full opacity (default).

Details

Function for graph auto-coloring; depending on the user's choice it assigns either colors or greyscale tones to matching strings of characters which stand for class identifiers. These metadata will typically be encoded in the texts' filenames. (As class delimiter, the underscore character should be used). Alternatively, all labels can be plotted in black.

Value

The function returns a vector of colors, using their conventional names (e.g. `red`, `maroon4`, `mediumturquoise`, `gold4`, `deepskyblue`, ...), or numeric values if the greyscale option was chosen (e.g. `#000000`, `#000000`, `#595959`, `#B2B2B2`, ...).

Author(s)

Maciej Eder

Examples

```
# in this example, three discrete classes are specified,
# for Tacitus, Caesar, and Livius
sample.names = c("Tacitus_Annales","Tacitus_Germania","Tacitus_Histories",
                 "Caesar_Civil_wars","Caesar_Gallic_wars",
                 "Livius_Ab_Urbe_Condita")
assign.plot.colors(sample.names)

# as above, but using greyscale:
assign.plot.colors(sample.names, col = "greyscale")
```

---

classify *Machine-learning classification*

Description

Function that performs a number of machine-learning methods for classification used in computational stylistics: Delta (Burrows, 2002), k-Nearest Neighbors, Support Vector Machines, Naive Bayes, and Nearest Shrunken Centroids (Jockers and Witten, 2010). Most of the options are derived from the `stylo` function.

Usage

```
classify(gui = TRUE, training.frequencies = NULL, test.frequencies = NULL,
         training.corpus = NULL, test.corpus = NULL, features = NULL,
         path = NULL, training.corpus.dir = "primary_set",
         test.corpus.dir = "secondary_set", ...)
```

Arguments

gui	an optional argument; if switched on, a simple yet effective graphical user interface (GUI) will appear. Default value is TRUE.
training.frequencies	using this optional argument, one can load a custom table containing frequencies/counts for several variables, e.g. most frequent words, across a number of text samples (for the training set). It can be either an R object (matrix or data frame), or a filename containing tab-delimited data. If you use an R object, make sure that the rows contain samples, and the columns – variables (words). If you use an external file, the variables should go vertically (i.e. in rows): this is because files containing vertically-oriented tables are far more flexible and easily editable using, say, Excel or any text editor. To flip your table horizontally/vertically use the generic function t().
test.frequencies	using this optional argument, one can load a custom table containing frequencies/counts for the test set. Further details: immediately above.
training.corpus	another option is to pass a pre-processed corpus as an argument (here: the training set). It is assumed that this object is a list, each element of which is a vector containing one tokenized sample. The example shown below will give you some hints how to prepare such a corpus. Also, refer to help(load.corpus.and.parse)
test.corpus	if training.corpus is used, then you should also prepare a similar R object containing the test set.
features	usually, a number of the most frequent features (words, word n-grams, character n-grams) are extracted automatically from the corpus, and they are used as variables for further analysis. However, in some cases it makes sense to use a set of tailored features, e.g. the words that are associated with emotions or, say, a specific subset of function words. This optional argument allows to pass either a filename containing your custom list of features, or a vector (R object) of features to be assessed.
path	if not specified, the current directory will be used for input/output procedures (reading files, outputting the results).
training.corpus.dir	the subdirectory (within the current working directory) that contains the training set, or the collection of texts used to exemplify the differences between particular classes (e.g. authors or genres). The discriminating features extracted from this training material will be used during the testing procedure (see below). If not specified, the default subdirectory primary_set will be used.
test.corpus.dir	the subdirectory (within the working directory) that contains the test set, or the collection of texts that are used to test the effectiveness of the discriminative features extracted from the training set. In the case of authorship attribution e.g., this set might contain works of non-disputed authorship, in order to check whether a classification procedure attribute the tets texts to their correct author. This set contains 'new' or 'unseen' data (e.g. anonymous samples or samples of disputed authorship in the case of authorship studies). If not specified, the default subdirectory secondary_set will be used.

... any variable as produced by `stylo.default.settings()` can be set here to overwrite the default values.

Details

There are numerous additional options that are passed to this function; so far, they are all loaded when `stylo.default.settings()` is executed (it will be invoked automatically from inside this function); the user can set/change them in the GUI.

Value

The function returns an object of the class `stylo.results`: a list of variables, including tables of word frequencies, vector of features used, a distance table and some more stuff. Additionally, depending on which options have been chosen, the function produces a number of files used to save the results, features assessed, generated tables of distances, etc.

Author(s)

Maciej Eder, Mike Kestemont, Jan Rybicki

References

Eder, M. Kestemont, M. and Rybicki, J. (2013). Stylometry with R: a suite of tools. In: "Digital Humanities 2013: Conference Abstracts". University of Nebraska-Lincoln, Lincoln, NE, pp. 487-89.

Burrows, J. F. (2002). "Delta": a measure of stylistic difference and a guide to likely authorship. "Literary and Linguistic Computing", 17(3): 267-87.

Jockers, M. L. and Witten, D. M. (2010). A comparative study of machine learning methods for authorship attribution. "Literary and Linguistic Computing", 25(2): 215-23.

Argamon, S. (2008). Interpreting Burrows's Delta: geometric and probabilistic foundations. "Literary and Linguistic Computing", 23(2): 131-47.

See Also

`stylo`, `rolling.delta`, `oppose`

Examples

```
## Not run:
# standard usage (it builds a corpus from a collection of text files):
classify()


# loading word frequencies from two tab-delimited files:
classify(training.frequencies = "table_with_training_frequencies.txt",
         test.frequencies = "table_with_test_frequencies.txt")


# using two existing sub-corpora (a list containing tokenized texts):
txt1 = c("now", "i", "am", "alone", "o", "what", "a", "slave", "am", "i")
```

```
txt2 = c("what", "do", "you", "read", "my", "lord")
  setTRAIN = list(txt1, txt2)
  names(setTRAIN) = c("hamlet_sample1","polonius_sample1")
txt4 = c("to", "be", "or", "not", "to", "be")
txt5 = c("though", "this", "be", "madness", "yet", "there", "is", "method")
txt6 = c("the", "rest", "is", "silence")
  setTEST = list(txt4, txt5, txt6)
  names(setTEST) = c("hamlet_sample2", "polonius_sample2", "uncertain_1")
classify(training.corpus = setTRAIN, test.corpus = setTEST)


# using a custom set of features (words, n-grams) to be analyzed:
my.selection.of.function.words = c("the", "and", "of", "in", "if", "into",
                                   "within", "on", "upon", "since")
classify(features = my.selection.of.function.words)


# loading a custom set of features (words, n-grams) from a file:
classify(features = "wordlist.txt")


# batch mode, custom name of corpus directories:
my.test = classify(gui = FALSE, training.corpus.dir = "TrainingSet",
       test.corpus.dir = "TestSet")
summary(my.test)


# batch mode, character 3-grams requested:
classify(gui = FALSE, analyzed.features = "c", ngram.size = 3)


## End(Not run)
```

`define.plot.area` *Define area for scatterplots*

Description

Function that determines the size of a scatterplot, taking into consideration additional margin to fit longer labels appearing on a graph (if applicable), optional margin defined by user, and some space to offset scatterplot labels from points (if applicable).

Usage

```
define.plot.area(x.coord, y.coord, xymargins = 2, v.offset = 0)
```

Arguments

`x.coord`	a vector of x coordinates, optionally with names.

y.coord	a vector of y coordinates.
xymargins	additional margins (expressed as a % of the actual plot area).
v.offset	label offset (expressed as a % of the actual plot area).

Details

Function that finds out the coordinates of scatterplots: it computes the extreme x and y values, adds margins, and optionally extends the top margin if a plot uses sample labels. Automatic margin extension will only take place if the x coordinates are supplemented by their names (i.e. labels of points to be shown on scatterplot).

Author(s)

Maciej Eder

See Also

assign.plot.colors, stylo

Examples

```
# to determine the plotting area for 4 points:
define.plot.area( c(1,2,3,4), c(-0.001,0.11,-0.023,0.09))

# to determine plot coordinates, taking into consideration
# the objects' names
my.points = cbind(c(1,2,3,4),c(-0.001,0.11,-0.023,0.09))
rownames(my.points) = c("first","second","third","very_long_fourth")
define.plot.area(my.points[,1], my.points[,2])
```

---

delete.markup *Delete HTML or XML tags*

Description

Function for removing markup tags (e.g. HTML, XML) from a string of characters. All XML markup is assumed to be compliant with the TEI guidelines (http://www.tei-c.org/).

Usage

```
delete.markup(input.text, markup.type = "plain")
```

Arguments

input.text	any string of characters (e.g. vector) containing markup tags that have to be deleted.
markup.type	any of the following values: plain (nothing will happen), html (all <tags> will be deleted as well as HTML header), xml (TEI header, all strings between <note> </note> tags, and all the tags will be deleted), xml.drama (as above; but, additionally, speaker's names will be deleted, or strings within each the <speaker> </speaker> tags), xml.notitles (as above; but, additionally, all the chapter/section (sub)titles will be deleted, or strings within each the <head> </head> tags).

Details

This function needs to be used carefully: while a document formatted in compliance with the TEI guidelines will be parsed flawlessly, the cleaning up of an HTML page harvested randomly on the web might cause some side effects, e.g. the footers, disclaimers, etc. will not be removed.

Author(s)

Maciej Eder, Mike Kestemont

See Also

load.corpus, txt.to.words, txt.to.words.ext, txt.to.features

Examples

```
delete.markup("Gallia est omnis <i>divisa</i> in partes tres",
         markup.type = "html")

delete.markup("Gallia<note>Gallia: Gaul.</note> est omnis
         <emph>divisa</emph> in partes tres", markup.type = "xml")

delete.markup("<speaker>Hamlet</speaker>Words, words, words...",
         markup.type = "xml.drama")
```

---

delete.stop.words *Exclude stop words (e.g. pronouns, particles, etc.) from a dataset*

Description

Function for removing custom words from a dataset: it can be the so-called stop words (frequent words without much meaning), or personal pronouns, or other custom elements of a dataset. It can be used to cull certain words from a vector containing tokenized text (particular words as elements of the vector), or to exclude unwanted columns (variables) from a table with frequencies. See examples below.

Usage

```
delete.stop.words(input.data, stop.words = NULL)
```

Arguments

input.data	either a vector containing words (actually, any countable features), or a data matrix/frame. The former in case of culling stop words from running text, the latter for culling them from tables of frequencies (then particular columns are excluded). The table should be oriented to contain samples in rows, variables in columns, and variables' names should be accessible via `colnames(input.table)`.
stop.words	a vector of words to be excluded.

Details

This function might be usefull to perform culling, or automatic deletion of the words that are too characteristic for particular texts. See `help(culling)` for further details.

Author(s)

Maciej Eder

See Also

`stylo.pronouns`, `perform.culling`

Examples

```
# (i) excluding stop words from a vector
my.text = c("omnis", "homines", "qui", "sese", "student", "praestare",
        "ceteris", "animalibus", "summa", "ope", "niti", "decet", "ne",
        "vitam", "silentio", "transeant", "veluti", "pecora", "quae",
        "natura", "prona", "atque", "ventri", "oboedientia", "finxit")
delete.stop.words(my.text, stop.words = c("qui", "quae", "ne", "atque"))


# (ii) excluding stop words from tabular data
#
# assume there is a matrix containing some frequencies
# (be aware that these counts are fictional):
t1 = c(2, 1, 0, 8, 9, 5, 6, 3, 4, 7)
t2 = c(7, 0, 5, 9, 1, 8, 6, 4, 2, 3)
t3 = c(5, 9, 2, 1, 6, 7, 8, 0, 3, 4)
t4 = c(2, 8, 6, 3, 0, 5, 9, 4, 7, 1)
my.data.table = rbind(t1, t2, t3, t4)

# names of the samples:
rownames(my.data.table) = c("text1", "text2", "text3", "text4")
# names of the variables (e.g. words):
colnames(my.data.table) = c("the", "of", "in", "she", "me", "you",
                                    "them", "if", "they", "he")
# the table looks as follows
```

```
print(my.data.table)

# now, one might want to get rid of the words "the", "of", "if":
delete.stop.words(my.data.table, stop.words = c("the", "of", "if"))

# also, some pre-defined lists of pronouns can be applied:
delete.stop.words(my.data.table,
                      stop.words = stylo.pronouns(language = "English"))
```

`gui.classify` *GUI for the function classify*

Description

Graphical user interface for `classify`. Via the GUI, this function can set most of the variables needed for `classify`.

Usage

```
gui.classify(...)
```

Arguments

`...`	any variable as produced by `stylo.default.settings` can be set here to overwrite the default values.

Details

The function calls `stylo.default.settings` to initialize a number of default variables. Then it reads the file `classify_config.txt` (if the file exists and can be found in the current directory) to overwrite any default values. Then a GUI box appears, allowing the variables' customization by the user. Refer to HOWTO available at https://sites.google.com/site/computationalstylistics/ for a detailed explanation what the particular variables are for and how to use them.

Value

The function returns a list containing ca. 100 variables.

Author(s)

Jan Rybicki, Maciej Eder

See Also

`classify`, `gui.stylo`

Examples

```
## Not run:
gui.classify()

my.variables = gui.classify()
summary(my.variables)

## End(Not run)
```

`gui.oppose` *GUI for the function oppose*

Description

Graphical user interface for `oppose`. This function sets most of the variables needed for `oppose`.

Usage

```
gui.oppose(...)
```

Arguments

`...`	any variable as produced by `stylo.default.settings` can be set here to overwrite the default values.

Details

The function calls `stylo.default.settings` to initialize a number of default variables. Then it reads the file `oppose_config.txt` (if the file exists and can be found in the current directory) to overwrite any default values. Then a GUI box appears, allowing the variables' customization by the user. Refer to HOWTO available at https://sites.google.com/site/computationalstylistics/ for a detailed explanation what the particular variables are for and how to use them.

Value

The function returns a list containing ca. 100 variables.

Author(s)

Jan Rybicki, Maciej Eder

See Also

`oppose`, `stylo.default.settings`

Examples

```
## Not run:
gui.oppose()

my.variables = gui.oppose()
summary(my.variables)

## End(Not run)
```

`gui.stylo` *GUI for stylo*

Description

Graphical user interface for the function `stylo`. This function sets most of the variables needed for `stylo`.

Usage

```
gui.stylo(...)
```

Arguments

`...`	any variable as produced by `stylo.default.settings` can be set here to overwrite the default values.

Details

The function calls `stylo.default.settings` to initialize a number of default variables. Then it reads the file `stylo_config.txt` (if the file exists and can be found in the current directory) to overwrite any default values. Then a GUI box appears, allowing the variables' customization by the user. Refer to HOWTO available at https://sites.google.com/site/computationalstylistics/ for a detailed explanation what the particular variables are for and how to use them.

Value

The function returns a list containing ca. 100 variables.

Author(s)

Jan Rybicki, Maciej Eder

See Also

`stylo`, `stylo.default.settings`

Examples

```
## Not run:
gui.stylo()

my.variables = gui.stylo()
summary(my.variables)

## End(Not run)
```

`load.corpus` *Load text files*

Description

Function for loading text files from a specified directory.

Usage

```
load.corpus(files, corpus.dir = "", encoding = "native.enc")
```

Arguments

`files`	a vector of file names.
`corpus.dir`	a directory containing the text files to be loaded; if not specified, the current working directory will be used.
`encoding`	useful if you use Windows and non-ASCII alphabets: French, Polish, Hebrew, etc. In such a situation, it is quite convenient to convert your text files into Unicode and to set this option to `encoding = "UTF-8"`. In Linux and Mac, you are always expected to use Unicode, thus you don't need to set anything.

Value

The function returns an object of the class `stylo.corpus`. It is a list containing as elements the texts loaded.

Author(s)

Maciej Eder

See Also

`stylo`, `classify`, `rolling.delta`, `oppose`, `txt.to.words`

Examples

```
## Not run:
# to load file1.txt and file2.txt, stored in the subdirectory my.files:
my.corpus = load.corpus(corpus.dir = "my.files",
                        files = c("file1.txt", "file2.txt") )

# to load all XML files from the current directory:
my.corpus = load.corpus(files = list.files(pattern="[.]xml$") )

## End(Not run)
```

`load.corpus.and.parse` *Load text files and perform pre-processing*

Description

A high-level function that controls a number of other functions responsible for loading texts from files, deleting markup, sampling from texts, converting samples to n-grams, etc. It is build on top of a number of functions and thus it requires a large number of arguments. The only obligatory argument, however, is a vector containing the names of the files to be loaded.

Usage

```
load.corpus.and.parse(files, corpus.dir = "", markup.type = "plain",
                      language = "English", splitting.rule = NULL,
                      sample.size = 10000, sampling = "no.sampling",
                      sample.overlap = 0, number.of.samples = 1,
                      sampling.with.replacement = FALSE, features = "w",
                      ngram.size = 1, preserve.case = FALSE,
                      encoding = "native.enc")
```

Arguments

`files`	a vector of file names.
`corpus.dir`	the directory containing the text files to be loaded; if not specified, the current directory will be used.
`markup.type`	choose one of the following values: `plain` (nothing will happen), `html` (all tags will be deleted as well as HTML header), `xml` (TEI header, any text between <note> </note> tags, and all the tags will be deleted), `xml.drama` (as above; additionally, speaker's names will be deleted, or strings within the <speaker> </speaker> tags), `xml.notitles` (as above; but, additionally, all the chapter/section (sub)titles will be deleted, or strings within each the <head> </head> tags); see `delete.markup` for further details.
`language`	an optional argument indicating the language of the texts analyzed; the values that will affect the function's behavior are: `English.contr`, `English.all`, `Latin.corr` (type `help(txt.to.words.ext)` for explanation). The default value is `English`.

`splitting.rule` if you are not satisfied with the default language settings (or your input string of characters is not a regular text, but a sequence of, say, dance movements represented using symbolic signs), you can indicate your custom splitting regular expression here. This option will overwrite the above language settings. For further details, refer to `help(txt.to.words)`.

`sample.size` desired size of samples, expressed in number of words; default value is 10,000.

`sampling` one of three values: `no.sampling` (default), `normal.sampling`, `random.sampling`. See `make.samples` for explanation.

`sample.overlap` if this opion is used, a reference text is segmented into consecutive, equal-sized samples that are allowed to partially overlap. If one specifies the `sample.size` parameter of 5,000 and the `sample.overlap` of 1,000, for example, the first sample of a text contains words 1–5,000, the second 4001–9,000, the third sample 8001–13,000, and so forth.

`number.of.samples` optional argument which will be used only if `random.sampling` was chosen; it is self-evident.

`sampling.with.replacement` optional argument which will be used only if `random.sampling` was chosen; it specifies the method used to randomly harvest words from texts.

`features` an option for specifying the desired type of features: `w` for words, `c` for characters (default: `w`). See `txt.to.features` for further details.

`ngram.size` an optional argument (integer) specifying the value of *n*, or the size of n-grams to be produced. If this argument is missing, the default value of 1 is used. See `txt.to.features` for further details.

`preserve.case` whether ot not to lowercase all characters in the corpus (default = F).

`encoding` useful if you use Windows and non-ASCII alphabets: French, Polish, Hebrew, etc. In such a situation, it is quite convenient to convert your text files into Unicode and to set this option to `encoding = "UTF-8"`. In Linux and Mac, you are always expected to use Unicode, thus you don't need to set anything.

Value

The function returns an object of the class `stylo.corpus`. It is a list containing as elements the samples (entire texts or sampled subsets) split into words/characters and combined into n-grams (if applicable).

Author(s)

Maciej Eder

See Also

`load.corpus`, `delete.markup`, `txt.to.words`, `txt.to.words.ext`, `txt.to.features`, `make.samples`

Examples

```
## Not run:
# to load file1.txt and file2.txt, stored in the subdirectory my.files:
my.corpus = load.corpus.and.parse(files = c("file1.txt", "file2.txt"),
                        corpus.dir = "my.files")

# to load all XML files from the current directory, while getting rid of
# all markup tags in the file, and split the texts into consecutive
# word pairs (2-grams):
my.corpus = load.corpus.and.parse(files = list.files(pattern = "[.]xml$"),
                        markup.type = "xml", ngram.size = 2)

## End(Not run)
```

`make.ngrams` *Make text n-grams*

Description

Function that combines a vector of text units (words, characters, POS-tags, other features) into pairs, triplets, or longer sequences, commonly referred to as n-grams.

Usage

```
make.ngrams(input.text, ngram.size = 1)
```

Arguments

`input.text`	a vector containing words or characters to be parsed into n-grams.
`ngram.size`	an optional argument (integer) indicating the value of *n*, or the size of n-grams to be produced. If this argument is missing, default value of 1 is used.

Details

Function for combining series of items (e.g. words or characters) into n-grams, or strings of *n* elements. E.g. character 2-grams of the sentence "This is a sentence" are as follows: "th", "hi", "is", "s ", " i", "is", "s ", " a", "a ", " s", "se", "en", "nt", "te", "en", "nc", "ce". Character 4-grams would be, of course: "this", "his ", "is a", "s a ", " a s", etc. Word 2-grams: "this is", "is a", "a sentence". The issue whether using n-grams of items increases the accuracy of stylometric procedures has been heavily debated in the secondary literature (see the reference section for further reading). Eder (2013) e.g. shows that character n-grams are suprisingly robust for dealing with noisy corpora (in terms of a high number of misspelled characters).

Author(s)

Maciej Eder

References

Alexis, A., Craig, H., and Elliot, J. (2014). Language chunking, data sparseness, and the value of a long marker list: explorations with word n-grams and authorial attribution. "Literary and Linguistic Computing", 29, advanced access (doi: 10.1093/llc/fqt028).

Eder, M. (2011). Style-markers in authorship attribution: a cross-language study of the authorial fingerprint. "Studies in Polish Linguistics", 6: 99-114. http://www.wuj.pl/page,art,artid,1923.html.

Eder, M. (2013). Mind your corpus: systematic errors in authorship attribution. "Literary and Linguistic Computing", 28(4): 603-14.

Hoover, D. L. (2002). Frequent word sequences and statistical stylistics. "Literary and Linguistic Computing", 17: 157-80.

Hoover, D. L. (2003). Frequent collocations and authorial style. "Literary and Linguistic Computing", 18: 261-86.

Hoover, D. L. (2012). The rarer they are, the more they are, the less they matter. In: Digital Humanities 2012: Conference Abstracts, Hamburg University, Hamburg, pp. 218-21.

Koppel, M., Schler, J. and Argamon, S. (2009). Computational methods in authorship attribution. "Journal of the American Society for Information Science and Technology", 60(1): 9-26.

Stamatatos, E. (2009). A survey of modern authorship attribution methods. "Journal of the American Society for Information Science and Technology", 60(3): 538-56.

See Also

txt.to.words, txt.to.words.ext, txt.to.features

Examples

```
# Consider the string my.text:
my.text = "Quousque tandem abutere, Catilina, patientia nostra?"
# which can be split into a vector of consecutive words:
my.vector.of.words = txt.to.words(my.text)
# now, we create a vector of word 2-grams:
make.ngrams(my.vector.of.words, ngram.size = 2)

# similarly, you can produce character n-grams:
my.vector.of.chars = txt.to.features(my.vector.of.words, features = "c")
make.ngrams(my.vector.of.chars, ngram.size = 4)
```

---

make.samples *Split text to samples*

Description

Function that either splits an input text (a vector of linguistic items, such as words, word n-grams, character n-grams, etc.) into equal-sized samples of a desired length (expressed in words), or excerpts randomly a number of words from the original text.

Usage

```
make.samples(tokenized.text, sample.size = 10000,
             sampling = "no.sampling", sample.overlap = 0,
             number.of.samples = 1, sampling.with.replacement = FALSE)
```

Arguments

`tokenized.text`	input textual data stored either in a form of vector (single text), or as a list of vectors (whole corpus); particular vectors should contain tokenized data, i.e. words, word n-grams, or other features, as elements.
`sample.size`	desired size of sample expressed in number of words; default value is 10,000.
`sampling`	one of three values: `no.sampling` (default), `normal.sampling`, `random.sampling`.
`sample.overlap`	if this opion is used, a reference text is segmented into consecutive, equal-sized samples that are allowed to partially overlap. If one specifies the `sample.size` parameter of 5,000 and the `sample.overlap` of 1,000, for example, the first sample of a text contains words 1–5,000, the second 4001–9,000, the third sample 8001–13,000, and so forth.
`number.of.samples`	optional argument which will be used only if `random.sampling` was chosen; it is self-evident.
`sampling.with.replacement`	optional argument which will be used only if `random.sampling` was chosen; it specifies the method to randomly harvest words from texts.

Details

Normal sampling is probably a good choice when the input texts are long: the advantage is that one gets a bigger number of samples which, in a way, validate the results (when several independent samples excerpted from one text are clustered together). When the analyzed texts are significantly unequal in length, it is not a bad idea to prepare samples as randomly chosen "bags of words". For this, set the `sampling` variable to `random.sampling`. The desired size of the sample should be specified via the `sample.size` variable. Sampling with and without replacement is also available. It has been shown by Eder (2010) that harvesting random samples from original texts improves the performance of authorship attribution methods.

Author(s)

Mike Kestemont, Maciej Eder

References

Eder, M. (2014). Does size matter? Authorship attribution, short samples, big problem. "Literary and Linguistic Computing", 29, advanced access (doi:10.1093/llc/fqt066).

See Also

`txt.to.words`, `txt.to.words.ext`, `txt.to.features`, `make.ngrams`

Examples

```
my.text = "Arma virumque cano, Troiae qui primus ab oris
           Italiam fato profugus Laviniaque venit
           litora, multum ille et terris iactatus et alto
           vi superum, saevae memorem Iunonis ob iram,
           multa quoque et bello passus, dum conderet urbem
           inferretque deos Latio; genus unde Latinum
           Albanique patres atque altae moenia Romae.
           Musa, mihi causas memora, quo numine laeso
           quidve dolens regina deum tot volvere casus
           insignem pietate virum, tot adire labores
           impulerit. tantaene animis caelestibus irae?"
my.words = txt.to.words(my.text)

# split the above text into samples of 20 words:
make.samples(my.words, sampling = "normal.sampling", sample.size = 20)

# excerpt randomly 50 words from the above text:
make.samples(my.words, sampling = "random.sampling", sample.size = 50)

# excerpt 5 random samples from the above text:
make.samples(my.words, sampling = "random.sampling", sample.size = 50,
             number.of.samples = 5)
```

---

make.table.of.frequencies

Prepare a table of (relative) word frequencies

Description

Function that collects several frequency lists and combines them into a single frequency table. To this end a number of rearrangements inside particular lists are carried out. The table is produced using a reference list of words/features (passed as an argument).

Usage

```
make.table.of.frequencies(corpus, features, absent.sensitive = TRUE,
                          relative = TRUE)
```

Arguments

`corpus`	textual data: either a corpus (represented as a list), or a single text (represented as a vector); the data have to be split into words (or other features, such as character n-grams or word pairs).
`features`	a vector containing a reference feature list that will be used to build the table of frequencies (it is assumed that the reference list contains the same type of features as the `corpus` list, e.g. words, character n-grams, word pairs, etc.; otherwise, an empty table will be build).

`absent.sensitive`
: this optional argument is used to prevent building tables of words/features that never occur in the corpus. When switched on (default), variables containing 0 values across all samples, will be excluded. However, in some cases this is important to keep all the variables regardless of their values. This is e.g. the case when comparing two corpora: even if a given word did not occur in corpus A, it might be present in corpus B. In short: whenever you perform any analysis involving two or multiple sets of texts, switch this option to `FALSE`.

`relative`
: when this argument is switched to `TRUE` (default), relative frequencies are computed instead of raw frequencies.

Author(s)

Maciej Eder

See Also

`load.corpus`, `load.corpus.and.parse`

Examples

```
# to get frequencies of the words "a", "the" and "of" from a text:

sample.txt = txt.to.words("My father had a small estate
             in Nottinghamshire: I was the third of five sons.")
make.table.of.frequencies(sample.txt, c("a", "the", "of"))



# to get a table of frequencies across several texts:

txt.1 = "Gallia est omnis divisa in partes tres, quarum unam incolunt
    Belgae, aliam Aquitani, tertiam qui ipsorum lingua Celtae, nostra
    Galli appellantur."
txt.2 = "Si quis antea, iudices, mirabatur quid esset quod, pro tantis
    opibus rei publicae tantaque dignitate imperi, nequaquam satis multi
    cives forti et magno animo invenirentur qui auderent se et salutem
    suam in discrimen offerre pro statu civitatis et pro communi
    libertate, ex hoc tempore miretur potius si quem bonum et fortem
    civem viderit, quam si quem aut timidum aut sibi potius quam rei
    publicae consulentem."
txt.3 = "Nam mores et instituta vitae resque domesticas ac familiaris
    nos profecto et melius tuemur et lautius, rem vero publicam nostri
    maiores certe melioribus temperaverunt et institutis et legibus."
my.corpus.raw = list(txt.1, txt.2, txt.3)
my.corpus.clean = lapply(my.corpus.raw, txt.to.words)
my.favorite.words = c("et", "in", "se", "rara", "avis")
make.table.of.frequencies(my.corpus.clean, my.favorite.words)


# to include all words in the reference list, no matter if they
# occurred in the corpus:
```

```
make.table.of.frequencies(my.corpus.clean, my.favorite.words,
    absent.sensitive=FALSE)


# to prepare a table of frequencies of all the words represented in
# a corpus, in descendent occurence order:
complete.word.list = names(sort(table(unlist(my.corpus.clean)),
    decreasing = TRUE))
make.table.of.frequencies(my.corpus.clean, complete.word.list)


# to create a table of frequencies of word pairs (word 2-grams):
my.word.pairs = lapply(my.corpus.clean, txt.to.features, ngram.size=2)
make.table.of.frequencies(my.word.pairs, c("et legibus", "hoc tempore"))
```

---

oppose *Contrastive analysis of texts*

Description

Function that performs a contrastive analysis between two given sets of texts. It generates a list of words significantly preferred by a tested author (or, a collection of authors), and another list containing the words significantly avoided by the former when compared to another set of texts. Some visualizations are available.

Usage

```
oppose(gui = TRUE, path = NULL,
         primary.corpus = NULL,
         secondary.corpus = NULL,
         test.corpus = NULL,
         primary.corpus.dir = "primary_set",
         secondary.corpus.dir = "secondary_set",
         test.corpus.dir = "test_set", ...)
```

Arguments

`gui`	an optional argument; if switched on, a simple yet effective graphical interface (GUI) will appear. Default value is `TRUE`.
`path`	if not specified, the current working directory will be used for input/output procedures (reading files, outputting the results, etc.).
`primary.corpus.dir`	the subdirectory (within the current working directory) that contains one or more texts to be compared to a comparison corpus. These texts can e.g. be the oeuvre by author A (to be compared to the oeuvre of another author B) or a collection of texts by female authors (to be contrasted with texts by male authors). If not specified, the default subdirectory `primary_set` will be used.

`secondary.corpus.dir`
: the subdirectory (within the current working directory) that contains a comparison corpus: a pool of texts to be contrasted with texts from the `primary.corpus`. If not specified, the default subdirectory `secondary_set` will be used.

`test.corpus.dir`
: the subdirectory (within the current working directory) that contains texts to verify the discriminatory strength of the features extracted from the `primary.set` and `secondary.sets`. Ideally, the `test.corpus.dir` should contain texts known to belong to both classes (e.g. texts written by female and male authors in the case of a gender-oriented study). If not specified, the default subdirectory `test_set` will be used. If the default subdirectory does not exist or does not contain any texts, the validation test will not be performed.

`primary.corpus`
: another option is to pass a pre-processed corpus as an argument (here: the primary set). It is assumed that this object is a list, each element of which is a vector containing one tokenized sample. Refer to `help(load.corpus.and.parse)` to get some hints how to prepare such a corpus.

`secondary.corpus`
: if `primary.corpus` is used, then you should also prepare a similar R object containing the secondary set.

`test.corpus`
: if you decide to use test corpus, you can pass it as a pre-processed R object using this argument.

`...`
: any variable produced by `stylo.default.settings` can be set here, in order to overwrite the default values.

Details

This function performs a contrastive analysis between two given sets of texts, using Burrows's Zeta (2007) in its different flavors, including Craig's extensions (Craig and Kinney, 2009). Also, the Whitney-Wilcoxon procedure as introduced by Kilgariff (2001) is available. The function generates a vector of words significantly preferred by a tested author, and another vector containing the words significantly avoided.

Value

The function returns an object of the class `stylo.results`: a list of variables, including a list of words significantly preferred in the primary set, words significantly avoided (or, preferred in the secondary set), and possibly some other results, if applicable.

Author(s)

Maciej Eder, Mike Kestemont, Jan Rybicki

References

Eder, M. Kestemont, M. and Rybicki, J. (2013). Stylometry with R: a suite of tools. In: "Digital Humanities 2013: Conference Abstracts". University of Nebraska-Lincoln, Lincoln, NE, pp. 487-89.

Burrows, J. F. (2007). All the way through: testing for authorship in different frequency strata. "Literary and Linguistic Computing", 22(1): 27-48.

Craig, H. and Kinney, A. F., eds. (2009). Shakespeare, Computers, and the Mystery of Authorship. Cambridge: Cambridge University Press.

Hoover, D. (2010). Teasing out authorship and style with t-tests and Zeta. In: "Digital Humanities 2010: Conference Abstracts". King's College London, pp. 168-170.

Kilgariff A. (2001). Comparing Corpora. "International Journal of Corpus Linguistics" 6(1): 1-37.

See Also

stylo, classify, rolling.delta

Examples

```
## Not run:
# standard usage:
oppose()

# batch mode, custom name of corpus directories:
oppose(gui = FALSE, primary.corpus.dir = "ShakespeareCanon",
       secondary.corpus.dir = "MarloweSamples")

## End(Not run)
```

---

perform.culling *Exclude variables (e.g. words, n-grams) from a frequency table that are too characteristic for some samples*

Description

Culling refers to the automatic manipulation of the wordlist (proposed by Hoover 2004a, 2004b). The culling values specify the degree to which words that do not appear in all the texts of a corpus will be removed. A culling value of 20 indicates that words that appear in at least 20% of the texts in the corpus will be considered in the analysis. A culling setting of 0 means that no words will be removed; a culling setting of 100 means that only those words will be used in the analysis that appear in all texts of the corpus at least once.

Usage

```
perform.culling(input.table, culling.level = 0)
```

Arguments

input.table	a matrix or data frame containing frequencies of words or any other countable features; the table should be oriented to contain samples in rows, variables in columns, and variables' names should be accessible via colnames(input.table).
culling.level	percentage of samples that need to have a given word in order to prevent this word from being culled (see the description above).

Author(s)

Maciej Eder

References

Hoover, D. (2004a). Testing Burrows's Delta. "Literary and Linguistic Computing", 19(4): 453-75.

Hoover, D. (2004b). Delta prime. "Literary and Linguistic Computing", 19(4): 477-95.

See Also

`delete.stop.words`, `stylo.pronouns`

Examples

```
# assume there is a matrix containing some frequencies
# (be aware that these counts are entirely fictional):
t1 = c(2, 1, 0, 2, 9, 1, 0, 0, 2, 0)
t2 = c(1, 0, 4, 2, 1, 0, 3, 0, 1, 3)
t3 = c(5, 2, 2, 0, 6, 0, 1, 0, 0, 0)
t4 = c(1, 4, 1, 0, 0, 0, 0, 3, 0, 1)
my.data.table = rbind(t1, t2, t3, t4)

# names of the samples:
rownames(my.data.table) = c("text1", "text2", "text3", "text4")
# names of the variables (e.g. words):
colnames(my.data.table) = c("the", "of", "in", "she", "me", "you",
                                    "them", "if", "they", "he")
# the table looks as follows
print(my.data.table)

# selecting the words that appeared in at laest 50% of samples:
perform.culling(my.data.table, 50)
```

---

perform.delta *Distance-based classifier*

Description

Delta: a simple yet effective machine-learning method of supervised classification, introduced by Burrows (2002). It computes a table of distances between samples, and compares each sample from the test set against training samples, in order to find its nearest neighbor. Apart from classic Delta, a number of alternative distance measures are supported by this function.

Usage

```
perform.delta(training.set, test.set,
              classes.training.set = NULL,
              classes.test.set = NULL,
              distance = "CD", no.of.candidates = 3,
              z.scores.both.sets = TRUE)
```

Arguments

`training.set` a table containing frequencies/counts for several variables – e.g. most frequent words – across a number of text samples (for the training set). Make sure that the rows contain samples, and the columns – variables (words, n-grams, or whatever needs to be analyzed).

`test.set` a table containing frequencies/counts for the training set. The variables used (i.e. columns) must match the columns of the training set.

`classes.training.set`
a vector containing class identifiers for the training set. When missing, the row names of the training set table will be used; the assumed classes are the strings of characters followed by the first underscore. Consider the following examples: c("Sterne_Tristram", "Sterne_Sentimental", "Fielding_Tom", ...), where the classes are the authors' names, and c("M_Joyce_Dubliners", "F_Woolf_Night_and_day", "M_Conrad_Lord_Jim", ...), where the classes are M(ale) and F(emale) according to authors' gender. Note that only the part up to the first underscore in the sample's name will be included in the class label.

`classes.test.set`
a vector containing class identifiers for the test set. When missing, the row names of the test set table will be used (see above).

`distance` a kernel (i.e. a distance measure) used for computing similarities between texts. Available options so far: "CD" (Burrow's Delta, default), "AD" (Argamon's Linear Delta), "ED" (Eder's Delta), "ES" (Eder's Simple distance), "CB" (Canberra Distance), "MH" (Manhattan Distance), "EU" (Euclidean Distance).

`no.of.candidates`
how many nearest neighbors will be computed for each test sample (default = 3).

`z.scores.both.sets`
many distance measures convert input variables into z-scores before computing any distances. Such a variable weighting is highly dependent on the number of input texts. One might choose either training set only to scale the variables, or the entire corpus (both sets). The latter is default.

Value

The function returns a vector of "guessed" classes: each test sample is linked with one of the classes represented in the training set. Additionally, final scores and final rankings of candidates are returned as attributes.

Author(s)

Maciej Eder

References

Argamon, S. (2008). Interpreting Burrows's Delta: geometric and probabilistic foundations. "Literary and Linguistic Computing", 23(2): 131-47.

Burrows, J. F. (2002). "Delta": a measure of stylistic difference and a guide to likely authorship. "Literary and Linguistic Computing", 17(3): 267-87.

Jockers, M. L. and Witten, D. M. (2010). A comparative study of machine learning methods for authorship attribution. "Literary and Linguistic Computing", 25(2): 215-23.

See Also

`perform.svm`, `perform.nsc`, `perform.knn`, `perform.naivebayes`

Examples

```
## Not run:
perform.delta(training.set, test.set)

## End(Not run)
```

`perform.knn` *k-Nearest Neighbor classifier*

Description

A machine-learning supervised classifier; this function is a wrapper for the k-NN procedure provided by the package `class`.

Usage

```
perform.knn(training.set, test.set, k.value = 1)
```

Arguments

`training.set`	a table containing frequencies/counts for several variables – e.g. most frequent words – across a number of text samples (for the training set). Make sure that the rows contain samples, and the columns – variables (words, n-grams, or whatever needs to be analyzed).
`test.set`	a table containing frequencies/counts for the training set. The variables used (i.e. columns) must match the columns of the training set.
`k.value`	number of nearest neighbors considered.

Value

The function returns a vector of "guessed" classes: each test sample is linked with one of the classes represented in the training set.

Author(s)

Maciej Eder

See Also

perform.svm, perform.nsc, perform.delta, perform.naivebayes

Examples

```
## Not run:
perform.knn(training.set, test.set)

## End(Not run)
```

---

perform.naivebayes *Naive Bayes classifier*

Description

A machine-learning supervised classifier; this function is a wrapper for the Naive Bayes procedure provided by the package e1071.

Usage

```
perform.naivebayes(training.set, test.set)
```

Arguments

training.set	a table containing frequencies/counts for several variables – e.g. most frequent words – across a number of text samples (for the training set). Make sure that the rows contain samples, and the columns – variables (words, n-grams, or whatever needs to be analyzed).
test.set	a table containing frequencies/counts for the training set. The variables used (i.e. columns) must match the columns of the training set.

Value

The function returns a vector of "guessed" classes: each test sample is linked with one of the classes represented in the training set.

Author(s)

Maciej Eder

See Also

perform.svm, perform.nsc, perform.delta, perform.knn

Examples

```
## Not run:
perform.naivebayes(training.set, test.set)

## End(Not run)
```

---

perform.nsc *Nearest Shrunken Centroids classifier*

Description

A machine-learning supervised classifier; this function is a wrapper for the Nearest Shrunken Centroids procedure provided by the package pamr.

Usage

```
perform.nsc(training.set,
            test.set,
            classes.training.set = NULL,
            classes.test.set = NULL,
            no.of.candidates = 3)
```

Arguments

training.set	a table containing frequencies/counts for several variables – e.g. most frequent words – across a number of text samples (for the training set). Make sure that the rows contain samples, and the columns – variables (words, n-grams, or whatever needs to be analyzed).
test.set	a table containing frequencies/counts for the training set. The variables used (i.e. columns) must match the columns of the training set.
classes.training.set	a vector containing class identifiers for the training set. When missing, the row names of the training set table will be used; the assumed classes are the strings of characters followed by the first underscore. Consider the following examples: c("Sterne_Tristram", "Sterne_Sentimental", "Fielding_Tom", ...), where the classes are the authors' names, and c("M_Joyce_Dubliners", "F_Woolf_Night_and_day", "M_Conrad_Lord_Jim", ...), where the classes are M(ale) and F(emale) according to authors' gender. Note that only the part up to the first underscore in the sample's name will be included in the class label.
classes.test.set	a vector containing class identifiers for the test set. When missing, the row names of the test set table will be used (see above).

no.of.candidates
: how many nearest neighbors will be computed for each test sample (default = 3).

Value

The function returns a vector of "guessed" classes: each test sample is linked with one of the classes represented in the training set. Additionally, final scores and final rankings of candidates are returned as attributes.

Author(s)

Maciej Eder

See Also

perform.delta, perform.svm, perform.knn, perform.naivebayes

Examples

```
## Not run:
perform.nsc(training.set, test.set)

## End(Not run)
```

---

perform.svm *Support Vector Machines classifier*

Description

A machine-learning supervised classifier; this function is a wrapper for the Support Vector Machines procedure provided by the package e1071.

Usage

```
perform.svm(training.set,
            test.set,
            classes.training.set = NULL,
            classes.test.set = NULL,
            no.of.candidates = 3,
            tune.parameters = FALSE,
            svm.kernel = "linear",
            svm.degree = 3,
            svm.coef0 = 0,
            svm.cost = 1)
```

Arguments

`training.set`	a table containing frequencies/counts for several variables – e.g. most frequent words – across a number of text samples (for the training set). Make sure that the rows contain samples, and the columns – variables (words, n-grams, or whatever needs to be analyzed).
`test.set`	a table containing frequencies/counts for the training set. The variables used (i.e. columns) must match the columns of the training set.
`classes.training.set`	a vector containing class identifiers for the training set. When missing, the row names of the training set table will be used; the assumed classes are the strings of characters followed by the first underscore. Consider the following examples: c("Sterne_Tristram", "Sterne_Sentimental", "Fielding_Tom", ...), where the classes are the authors' names, and c("M_Joyce_Dubliners", "F_Woolf_Night_and_day", "M_Conrad_Lord_Jim", ...), where the classes are M(ale) and F(emale) according to authors' gender. Note that only the part up to the first underscore in the sample's name will be included in the class label.
`classes.test.set`	a vector containing class identifiers for the test set. When missing, the row names of the test set table will be used (see above).
`no.of.candidates`	how many nearest neighbors will be computed for each test sample (default = 3).
`tune.parameters`	if this argument is used, two parameters, namely gamma and cost, are tuned using a bootstrap procedure, and then used to build a SVM model.
`svm.kernel`	SVM kernel. Available values: "linear", which is probably the best choice in stylometry, since the number of variables (e.g. MFWs) is many times bigger than the number of classes; "polynomial", and "radial".
`svm.degree`	parameter needed for kernel of type "polynomial" (default: 3).
`svm.coef0`	parameter needed for kernel of type "polynomial" (default: 0).
`svm.cost`	cost of constraints violation (default: 1); it is the C-constant of the regularization term in the Lagrange formulation.

Value

The function returns a vector of "guessed" classes: each test sample is linked with one of the classes represented in the training set. Additionally, final scores and final rankings of candidates are returned as attributes.

Author(s)

Maciej Eder

See Also

`perform.delta`, `perform.nsc`, `perform.knn`, `perform.naivebayes`

Examples

```
## Not run:
perform.svm(training.set, test.set)

## End(Not run)
```

`rolling.classify` *Sequential machine-learning classification*

Description

Function that splits a text into equal-sized consecutive blocks (slices) and performs a supervised classification of these blocks against a training set. A number of machine-learning methods for classification used in computational stylistics are available: Delta, k-Nearest Neighbors, Support Vector Machines, Naive Bayes, and Nearest Shrunken Centroids.

Usage

```
rolling.classify(gui = FALSE, training.corpus.dir = "reference_set",
         test.corpus.dir = "test_set", training.frequencies = NULL,
         test.frequencies = NULL, training.corpus = NULL,
         test.corpus = NULL,  features = NULL, path = NULL,
         slice.size = 5000, slice.overlap = 4500,
         training.set.sampling = "no.sampling", mfw = 100, culling = 0,
         milestone.points = NULL, add.ticks = FALSE, ...)
```

Arguments

`gui` an optional argument; if switched on, a simple yet effective graphical user interface (GUI) will appear. Default value is `FALSE` so far, since GUI is still under development.

`training.frequencies` using this optional argument, one can load a custom table containing frequencies/counts for several variables, e.g. most frequent words, across a number of text samples (for the training set). It can be either an R object (matrix or data frame), or a filename containing tab-delimited data. If you use an R object, make sure that the rows contain samples, and the columns – variables (words). If you use an external file, the variables should go vertically (i.e. in rows): this is because files containing vertically-oriented tables are far more flexible and easily editable using, say, Excel or any text editor. To flip your table horizontally/vertically use the generic function `t()`.

`test.frequencies` using this optional argument, one can load a custom table containing frequencies/counts for the test set. Further details: immediately above.

`training.corpus`
: another option is to pass a pre-processed corpus as an argument (here: the training set). It is assumed that this object is a list, each element of which is a vector containing one tokenized sample. The example shown below will give you some hints how to prepare such a corpus. Also, refer to `help(load.corpus.and.parse)`

`test.corpus`
: if `training.corpus` is used, then you should also prepare a similar R object containing the test set.

`features`
: usually, a number of the most frequent features (words, word n-grams, character n-grams) are extracted automatically from the corpus, and they are used as variables for further analysis. However, in some cases it makes sense to use a set of tailored features, e.g. the words that are associated with emotions or, say, a specific subset of function words. This optional argument allows to pass either a filename containing your custom list of features, or a vector (R object) of features to be assessed.

`path`
: if not specified, the current directory will be used for input/output procedures (reading files, outputting the results).

`training.corpus.dir`
: the subdirectory (within the current working directory) that contains the training set, or the collection of texts used to exemplify the differences between particular classes (e.g. authors or genres). The discriminating features extracted from this training material will be used during the testing procedure (see below). If not specified, the default subdirectory `reference_set` will be used.

`test.corpus.dir`
: the subdirectory (within the working directory) that contains a test to be assessed, long enough to be split automatically into equal-sized slices, or blocks. If not specified, the default subdirectory `test_set` will be used.

`slice.size`
: a text to be analyzed is segmented into consecutive, equal-sized samples (slices, windows, or blocks); the slice size is set using this parameter: default is 5,000 words. The samples are allowed to partially overlap (see the next parameter).

`slice.overlap`
: if we specifies a `slice.size` of 5,000 and a `slice.overlap` of 4,500 (which is default), then the first sample of a text contains words 1–5,000, the second 501–5,500, the third sample 1001–6,000, and so forth.

`training.set.sampling`
: sometimes, it makes sense to split training set texts into smaller samples. Available options: "no.sampling" (default), "normal.sampling", "random.sampling". See `help(make.samples)` for further details.

`mfw`
: number of the most frequent words (MFWs) to be analyzed.

`culling`
: culling level; see `help(perform.culling)` to get some help on the culling procedure principles.

`milestone.points`
: sometimes, there is a need to mark one or more passages in an analyzed text (e.g. when external evidence suggests an authorial takeover at a certain point) to compare if the a priori knowledge is confirmed by stylometric evidence. To this end, one should add into the test file a string "xmilestone" (when input texts are loaded directly from files), or specify the break points using this parameter. E.g., to add two lines at 10,000 words and 15,000 words, use `milestone.points = c(10000, 15000)`.

add.ticks	a graphical parameter: consider adding tiny ticks (short horizontal lines) to see the density of sampling. Default: FALSE.
...	any variable as produced by stylo.default.settings() can be set here to overwrite the default values.

Details

There are numerous additional options that are passed to this function; so far, they are all loaded when stylo.default.settings() is executed (it will be invoked automatically from inside this function); the user can set/change them in the GUI.

Value

The function returns an object of the class stylo.results: a list of variables, including tables of word frequencies, vector of features used, a distance table and some more stuff. Additionally, depending on which options have been chosen, the function produces a number of files used to save the results, features assessed, generated tables of distances, etc.

Author(s)

Maciej Eder

See Also

classify, rolling.delta

Examples

```
## Not run:
# standard usage (it builds a corpus from a collection of text files):
rolling.classify()

rolling.classify(training.frequencies = "freqs_train.txt",
    test.frequencies = "freqs_test.txt", write.png.file = TRUE,
    classification.method = "nsc")

## End(Not run)
```

---

rolling.delta *Sequential stylometric analysis*

Description

Function that analyses collaborative works and tries to determine the authorship of their fragments.

Usage

```
rolling.delta(gui = TRUE, path = NULL, primary.corpus.dir = "primary_set",
              secondary.corpus.dir = "secondary_set")
```

Arguments

gui	an optional argument; if switched on, a simple yet effective graphical user interface (GUI) will appear. Default value is TRUE.
path	if not specified, the current working directory will be used for input/output procedures (reading files, outputting the results).
primary.corpus.dir	the subdirectory (within the current working directory) that contains a collection of texts written by the authorial candidates, likely to have been involved in the collaborative work analyzed. If not specified, the default subdirectory primary_set will be used.
secondary.corpus.dir	the subdirectory (within the current working directory) that contains the collaborative work to be analyzed. If not specified, the default subdirectory secondary_set will be used.

Details

The procedure provided by this function analyses collaborative works and tries to determine the authorship of their fragments. The first step involves a "windowing" procedure (Dalen-Oskam and Zundert, 2007) in which each reference text is segmented into consecutive, equal-sized samples or windows. After "rolling" through the test text, we can plot the resulting series of Delta scores for each reference text in a graph.

Value

The function returns an object of the class stylo.results, and produces a final plot.

Author(s)

Mike Kestemont, Maciej Eder, Jan Rybicki

References

Eder, M. Kestemont, M. and Rybicki, J. (2013). Stylometry with R: a suite of tools. In: "Digital Humanities 2013: Conference Abstracts". University of Nebraska-Lincoln, Lincoln, NE, pp. 487-89.

van Dalen-Oskam, K. and van Zundert, J. (2007). Delta for Middle Dutch: author and copyist distinction in Walewein. "Literary and Linguistic Computing", 22(3): 345-62.

Hoover, D. (2011). The Tutor's Story: a case study of mixed authorship. In: "Digital Humanities 2011: Conference Abstracts". Stanford University, Stanford, CA, pp. 149-51.

Rybicki, J., Kestemont, M. and Hoover D. (2014). Collaborative authorship: Conrad, Ford and rolling delta. "Literary and Linguistic Computing", 29(3): 422-31.

See Also

stylo, classify, oppose

Examples

```
## Not run:
# standard usage:
rolling.delta()

# batch mode, custom name of corpus directories:
rolling.delta(gui = FALSE, primary.corpus.dir = "MySamples",
       secondary.corpus.dir = "ReferenceCorpus")

## End(Not run)
```

`stylo` *Stylometric multidimensional analyses*

Description

It is quite a long story what this function does. Basically, it is an all-in-one tool for a variety of experiments in computational stylistics. For a more detailed description, refer to HOWTO available at: https://sites.google.com/site/computationalstylistics/

Usage

```
stylo(gui = TRUE, frequencies = NULL, parsed.corpus = NULL,
      features = NULL, path = NULL, corpus.dir = "corpus", ...)
```

Arguments

`gui`	an optional argument; if switched on, a simple yet effective graphical interface (GUI) will appear. Default value is `TRUE`.
`frequencies`	using this optional argument, one can load a custom table containing frequencies/counts for several variables, e.g. most frequent words, across a number of text samples. It can be either an R object (matrix or data frame), or a filename containing tab-delimited data. If you use an R object, make sure that the rows contain samples, and the columns – variables (words). If you use an external file, the variables should go vertically (i.e. in rows): this is because files containing vertically-oriented tables are far more flexible and easily editable using, say, Excel or any text editor. To flip your table horizontally/vertically use the generic function t().
`parsed.corpus`	another option is to pass a pre-processed corpus as an argument. It is assumed that this object is a list, each element of which is a vector containing one tokenized sample. The example shown below will give you some hints how to prepare such a corpus.
`features`	usually, a number of the most frequent features (words, word n-grams, character n-grams) are extracted automatically from the corpus, and they are used as variables for further analysis. However, in some cases it makes sense to use a set of tailored features, e.g. the words that are associated with emotions or, say,

a specific subset of function words. This optional argument allows to pass either a filename containing your custom list of features, or a vector (R object) of features to be assessed.

`path` if not specified, the current directory will be used for input/output procedures (reading files, outputting the results).

`corpus.dir` the subdirectory (within the current working directory) that contains the corpus text files. If not specified, the default subdirectory `corpus` will be used. This option is immaterial when an external corpus and/or external table with frequencies is loaded.

`...` any variable produced by `stylo.default.settings` can be set here, in order to overwrite the default values. An example of such a variable is `network = TRUE` (switched off as default) for producing stylometric bootstrap consensus networks (Eder, forthcoming); the function saves a csv file, containing a list of nodes that can be loaded into, say, Gephi.

Details

If no additional argument is passed, then the functions tries to load text files from the default subdirectory `corpus`. There are a lot of additional options that should be passed to this function; they are all loaded when `stylo.default.settings` is executed (which is typically called automatically from inside the `stylo` function).

Value

The function returns an object of the class `stylo.results`: a list of variables, including a table of word frequencies, vector of features used, a distance table and some more stuff. Additionally, depending on which options have been chosen, the function produces a number of files containing results, plots, tables of distances, etc.

Author(s)

Maciej Eder, Jan Rybicki, Mike Kestemont

References

Eder, M. Kestemont, M. and Rybicki, J. (2013). Stylometry with R: a suite of tools. In: "Digital Humanities 2013: Conference Abstracts". University of Nebraska-Lincoln, Lincoln, NE, pp. 487-89.

Eder, M. (forthcoming). Visualization in stylometry: some problems and solutions.

See Also

`classify`, `oppose`, `rolling.delta`

Examples

```
## Not run:
# standard usage (it builds a corpus from a set of text files):
stylo()
```

```
# loading word frequencies from a tab-delimited file:
stylo(frequencies = "my_frequencies.txt")

# using an existing corpus (a list containing tokenized texts):
txt1 = c("to", "be", "or", "not", "to", "be")
txt2 = c("now", "i", "am", "alone", "o", "what", "a", "slave", "am", "i")
txt3 = c("though", "this", "be", "madness", "yet", "there", "is", "method")
custom.txt.collection = list(txt1, txt2, txt3)
  names(custom.txt.collection) = c("hamlet_A", "hamlet_B", "polonius_A")
stylo(parsed.corpus = custom.txt.collection)

# using a custom set of features (words, n-grams) to be analyzed:
my.selection.of.function.words = c("the", "and", "of", "in", "if", "into",
                                   "within", "on", "upon", "since")
stylo(features = my.selection.of.function.words)

# loading a custom set of features (words, n-grams) from a file:
stylo(features = "wordlist.txt")

# batch mode, custom name of corpus directory:
my.test = stylo(gui = FALSE, corpus.dir = "ShakespeareCanon")
summary(my.test)

# batch mode, character 3-grams requested:
stylo(gui = FALSE, analyzed.features = "c", ngram.size = 3)


## End(Not run)
```

`stylo.default.settings`

Setting variables for the package stylo

Description

Function that sets a series of global variables to be used by the package `stylo` and which can be modified by users via arguments passed to the function and/or via `gui.stylo`, `gui.classify`, or `gui.oppose`.

Usage

```
stylo.default.settings(...)
```

Arguments

`...`	any variable as produced by this function can be set here to overwrite the default values.

Details

This function is typically called from inside `stylo`, `classify`, `oppose`, `gui.stylo`, `gui.classify` and `gui.oppose`.

Value

The function returns a list of a few dozen variables, mostly options and parameters for different stylometric tests.

Author(s)

Maciej Eder, Jan Rybicki, Mike Kestemont

See Also

`stylo`, `gui.stylo`

Examples

```
stylo.default.settings()

# to see which variables have been set:
names(stylo.default.settings())

# to use the elements of the list as if they were independent variables:
my.variables = stylo.default.settings()
attach(my.variables)
```

`stylo.pronouns` *List of pronouns*

Description

This function returns a list of pronouns that can be used as a stop word list for different stylometric analyses. It has been shown that pronoun deletion improves, to some extent, attribution accuracy of stylometric procedures (e.g. in English novels: Hoover 2004a; 2004b).

Usage

```
stylo.pronouns(language = "English")
```

Arguments

`language`	an optional argument specifying the language of the texts analyzed: available languages are `English`, `Latin`, `Polish`, `Dutch`, `French`, `German`, `Spanish`, `Italian`, and `Hungarian` (default is `English`).

Value

The function returns a vector of pronouns.

Author(s)

Jan Rybicki, Maciej Eder, Mike Kestemont

References

Hoover, D. (2004a). Testing Burrows's delta. "Literary and Linguistic Computing", 19(4): 453-75.

Hoover, D. (2004b). Delta prime?. "Literary and Linguistic Computing", 19(4): 477-95.

See Also

stylo

Examples

```
stylo.pronouns()
stylo.pronouns(language="Latin")
my.stop.words = stylo.pronouns(language="German")
```

---

txt.to.features *Split string of words or other countable features*

Description

Function that converts a vector of words into either words, or characters, and optionally parses them into n-grams.

Usage

```
txt.to.features(tokenized.text, features = "w", ngram.size = 1)
```

Arguments

tokenized.text	a vector of tokinzed words
features	an option for specifying the desired type of feature: w for words, c for characters (default: w).
ngram.size	an optional argument (integer) indicating the value of *n*, or the size of n-grams to be created. If this argument is missing, the default value of 1 is used.

Details

Function that carries out the preprocessing steps necessary for feature selection: converts an input text into the type of sequences needed (n-grams etc.) and returns a new vector of items. The function invokes make.ngrams to combine single units into pairs, triplets or longer n-grams. See help(make.ngrams) for details.

Author(s)

Maciej Eder, Mike Kestemont

See Also

txt.to.words, txt.to.words.ext, make.ngrams

Examples

```
# consider the string my.text:
my.text = "Quousque tandem abutere, Catilina, patientia nostra?"

# split it into a vector of consecutive words:
my.vector.of.words = txt.to.words(my.text)

# build a vector of word 2-grams:
txt.to.features(my.vector.of.words, ngram.size = 2)

# or produce character n-grams (in this case, character tetragrams):
txt.to.features(my.vector.of.words, features = "c", ngram.size = 4)
```

---

txt.to.words *Split text into words*

Description

Generic tokenization function for splitting a given input text into single words (chains of characters delimited by spaces or punctuation marks).

Usage

```
txt.to.words(input.text, splitting.rule = NULL, preserve.case = FALSE)
```

Arguments

input.text	a string of characters, usually a text.
splitting.rule	an optional argument indicating an alternative splitting regexp. E.g., if your corpus contains no punctuation, you can use a very simple splitting sequence: "[\t\n]+" or "[[:space:]]+" (in this case, any whitespace is assumed to be a word delimiter). If you deal with non-latin scripts, especially with those that are not supported by the stylo package yet (e.g. Chinese, Japanese, Vietnamese, Georgian), you can indicate your letter characters explicitly: for most Cyrillic scripts try the following code "[^\u0400-\u0482\u048A\u04FF]+". Remember, however, that your texts need to be properly loaded into R (which is quite tricky on Windows; see below).
preserve.case	Whether or not to lowercase all characters in the corpus (default is FALSE).

Details

The generic tokenization function for splitting a given input text into single words (chains of characters delimited with spaces or punctuation marks). In obsolete versions of the package `stylo`, the default splitting sequence of chars was `"[^[:alpha:]]+"` on Mac/Linux, and `"\\W+_"` on Windows. Two different splitting rules were used, because regular expressions are not entirely platform-independent; type `help(regexp)` for more details. For the sake of compatibility, then, in the version >=0.5.6 a lengthy list of dozens of letters in a few alphabets (Latin, Cyrillic, Greek, Hebrew, Arabic so far) has been indicated explicitly:

```
paste("[^A-Za-z",
    # Latin supplement (Western):
    "\U00C0-\U00FF",
    # Latin supplement (Eastern):
    "\U0100-\U01BF",
    # Latin extended (phonetic):
    "\U01C4-\U02AF",
    # modern Greek:
    "\U0386\U0388-\U03FF",
    # Cyrillic:
    "\U0400-\U0481\U048A-\U0527",
    # Hebrew:
    "\U05D0-\U05EA\U05F0-\U05F4",
    # Arabic:
    "\U0620-\U065F\U066E-\U06D3\U06D5\U06DC",
    # extended Latin:
    "\U1E00-\U1EFF",
    # ancient Greek:
    "\U1F00-\U1FBC\U1FC2-\U1FCC\U1FD0-\U1FDB\U1FE0-\U1FEC\U1FF2-\U1FFC",
    "]+",
    sep="")
```

Alternatively, different tokenization rules can be applied through the option `splitting.rule` (see above). ATTENTION: this is the only piece of coding in the library `stylo` that might depend on the operating system used. While on Mac/Linux the native encoding is Unicode, on Windows you never know if your text will be loaded proprely. A considerable solution for Windows users is to convert your texts into Unicode (a variety of freeware converters are available on the internet), and to use an appropriate encoding option when loading the files: `read.table("file.txt", encoding = "UTF-8"` or `scan("file.txt", what = "char", encoding = "UTF-8"`. If you use the functions provided by the library `stylo`, you should pass this option as an argument to your chosen function: `stylo(encoding = "UTF-8")`, `classify(encoding = "UTF-8")`, `oppose(encoding = "UTF-8")`.

Value

The function returns a vector of tokenized words (or other units) as elements.

Author(s)

Maciej Eder, Jan Rybicki, Mike Kestemont

See Also

txt.to.words.ext, txt.to.features, make.ngrams, load.corpus

Examples

```
txt.to.words("And now, Laertes, what's the news with you?")

# retrieving grammatical codes (POS tags) from a tagged text:
tagged.text = "The_DT family_NN of_IN Dashwood_NNP had_VBD long_RB
               been_VBN settled_VBN in_IN Sussex_NNP ._."
txt.to.words(tagged.text, splitting.rule = "([A-Za-z,.;!]+_)|[ \n\t]")
```

---

txt.to.words.ext *Split text into words: extended version*

Description

Function for splitting a string of characters into single words, removing punctuation etc., and preserving some language-dependent idiosyncracies, such as common contractions in English.

Usage

```
txt.to.words.ext(input.text, language = "English", splitting.rule = NULL,
                 preserve.case = FALSE)
```

Arguments

input.text	a string of characters, usually a text.
language	an optional argument specifying the language of the texts analyzed. Values that will affect the function's output are: English.contr, English.all, Latin.corr (their meaning is explained below). The default value is English.
splitting.rule	if you are not satisfied with the default language settings (or your input string of characters is not a regular text, but a sequence of, say, dance movements represented using symbolic signs), you can indicate your custom splitting regular expression here. This option will overwrite the above language settings. For further details, refer to help(txt.to.words).
preserve.case	Whether or not to lowercase all character in the corpus (default = F).

Details

Function for splitting a given input text into single words (chains of characters delimited with spaces or punctuation marks). It is build on top of the function txt.to.words and it is designed to manage some language-dependent text features during the tokenization process. In most languages, this is irrelevant. However, it might be important when with English or Latin texts: English.contr treats contractions as single, atomary words, i.e. strings such as "don't", "you've" etc. will not be split into two strings; English.all keeps the contractions (as above), and also prevents the function from splitting compound words (mother-in-law, double-decker, etc.). Latin.corr: since some editions

do not distinguish the letters v/u, this setting provides a consistent conversion to "u" in the whole string. The option `preserve.case` lets you specify whether you wish to lowercase all characters in the corpus.

Author(s)

Maciej Eder, Jan Rybicki, Mike Kestemont

See Also

`txt.to.words`, `txt.to.features`, `make.ngrams`

Examples

```
txt.to.words.ext("Nel mezzo del cammin di nostra vita / mi ritrovai per
    una selva oscura, che la diritta via era smarrita.")

# to see the difference between particular options for English,
# consider the following sentence from Joseph Conrad's "Nostromo":
sample.text = "That's how your money-making is justified here."
txt.to.words.ext(sample.text, language = "English")
txt.to.words.ext(sample.text, language = "English.contr")
txt.to.words.ext(sample.text, language = "English.all")
```

---

zeta.chisquare *Compare two subcorpora using a home-brew variant of Craig's Zeta*

Description

This is a function for comparing two sets of texts; unlike keywords analysis, it this method the goal is to split input texts into equal-sized slices, and to check the appearance of particular words over the slices. Number of slices in which a given word appeared in the subcorpus A and B is then compared using standard chisquare test (if p value exceeds 0.05, a difference is considered significant). This method is based on original Zeta as developed by Burrows and extended by Craig (Burrows 2007, Craig and Kinney 2009).

Usage

```
zeta.chisquare(input.data)
```

Arguments

`input.data`	a matrix of two columns.

Value

The function returns a list of two elements: the first contains words (or other units, like n-grams) statistically preferred by the authors of the primary subcorpus, while the second element contains avoided words. Since the applied measure is symmetrical, the preferred words are ipso facto avoided by the secondary authors, and vice versa.

Author(s)

Maciej Eder

References

Burrows, J. F. (2007). All the way through: testing for authorship in different frequency strata. "Literary and Linguistic Computing", 22(1): 27-48.

Craig, H. and Kinney, A. F., eds. (2009). Shakespeare, Computers, and the Mystery of Authorship. Cambridge: Cambridge University Press.

See Also

oppose, zeta.eder, zeta.craig

Examples

```
## Not run:
zeta.chisquare(input.data, filter.threshold)

## End(Not run)
```

---

zeta.craig *Compare two subcorpora using Craig's Zeta*

Description

This is a function for comparing two sets of texts; unlike keywords analysis, it this method the goal is to split input texts into equal-sized slices, and to check the appearance of particular words over the slices. Number of slices in which a given word appeared in the subcorpus A and B is then compared using Craig's formula, which is based on original Zeta as developed by Burrows (Craig and Kinney 2009, Burrows 2007).

Usage

```
zeta.craig(input.data, filter.threshold)
```

Arguments

input.data	a matrix of two columns.
filter.threshold	this parameter (default 0.1) gets rid of words of weak discrimination strength; the higher the number, the less words appear in the final wordlists. It does not normally exceed 0.5. In original Craig's Zeta, no threshold is used: instead, the results contain the fixed number of 500 top avoided and 500 top preferred words.

Value

The function returns a list of two elements: the first contains words (or other units, like n-grams) statistically preferred by the authors of the primary subcorpus, while the second element contains avoided words. Since the applied measure is symmetrical, the preferred words are ipso facto avoided by the secondary authors, and vice versa.

Author(s)

Maciej Eder

References

Burrows, J. F. (2007). All the way through: testing for authorship in different frequency strata. "Literary and Linguistic Computing", 22(1): 27-48.

Craig, H. and Kinney, A. F., eds. (2009). Shakespeare, Computers, and the Mystery of Authorship. Cambridge: Cambridge University Press.

See Also

oppose, zeta.eder, zeta.chisquare

Examples

```
## Not run:
zeta.craig(input.data, filter.threshold)

## End(Not run)
```

---

zeta.eder *Compare two subcorpora using Eder's Zeta*

Description

This is a function for comparing two sets of texts; unlike keywords analysis, it this method the goal is to split input texts into equal-sized slices, and to check the appearance of particular words over the slices. Number of slices in which a given word appeared in the subcorpus A and B is then compared using a distance derived from Canberra measure of similarity. Original Zeta was developed by Burrows and extended by Craig (Burrows 2007, Craig and Kinney 2009).

Usage

```
zeta.eder(input.data, filter.threshold)
```

Arguments

`input.data`	a matrix of two columns.
`filter.threshold`	this parameter (default 0.1) gets rid of words of weak discrimination strength; the higher the number, the less words appear in the final wordlists. It does not normally exceed 0.5.

Value

The function returns a list of two elements: the first contains words (or other units, like n-grams) statistically preferred by the authors of the primary subcorpus, while the second element contains avoided words. Since the applied measure is symmetrical, the preferred words are ipso facto avoided by the secondary authors, and vice versa.

Author(s)

Maciej Eder

References

Burrows, J. F. (2007). All the way through: testing for authorship in different frequency strata. "Literary and Linguistic Computing", 22(1): 27-48.

Craig, H. and Kinney, A. F., eds. (2009). Shakespeare, Computers, and the Mystery of Authorship. Cambridge: Cambridge University Press.

See Also

`oppose`, `zeta.eder`, `zeta.chisquare`

Examples

```
## Not run:
zeta.eder(input.data, filter.threshold)

## End(Not run)
```

Index