Package 'textreg'

October 4, 2018

Type Package

Title n-Gram Text Regression, aka Concise Comparative Summarization

Version 0.1.5

Date 2018-09-01

Author Luke Miratrix

Maintainer Luke Miratrix <lmiratrix@stat.harvard.edu>

Description Function for sparse regression on raw text, regressing a labeling vector onto a feature space consisting of all possible phrases.

License MIT + file LICENSE

Depends R (>= 2.10),

Imports tm (>= 0.7), NLP (>= 0.1-10), Rcpp (>= 0.12.9), stats, graphics, utils

Suggests corrplot, knitr, SnowballC (>= 0.5.1), xtable, testthat, plyr, rmarkdown

LinkingTo Rcpp

VignetteBuilder knitr

RoxygenNote 6.1.0

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-10-04 11:00:03 UTC

R topics documented:

`textreg-package` *Sparse regression package for text that allows for multiple word phrases.*

Description

Built on Georgiana Ifrim's work, but allowing for regularization of phrases, this package does sparse regression using greedy coordinate descent. In a nutshell, the textreg package allows for regressing a vector of +1/-1 labels onto raw text. The textreg package takes care of converting the text to all of the possible related features, allowing you to think of the more organic statement of regressing onto "text" in some broad sense.

Details

Implementation-wise, it is a wrapper for a modified version of the C++ code written by Georgiana Ifrim to do this regression. It is also designed to (somewhat) integrate with the tm package, a commonly used R package for dealing with text.

One warning: this package uses tm, but does need to generate vectors of character strings to pass to the textreg call, which can be quite expensive. You can also pass a filename to the textreg call instead, which allows one to avoid loading a large corpus into memory and then copying it over. You can use a prior build.corpus command before textreg to mitigate this cost, but it is an imperfect method.

The n-gram package is documented, but it is research code, meaning gaps and errors are possible; the author would appreciate notification of anything that is out of order.

The primary method in this package is the regression call 'textreg()'. This method takes a corpus and a labeling vector and returns a textreg.result object that contains the final regression result along with diagnostic information that can be of use.

Start by reading the "bathtub" vignette, which walks through most of the functionality of this package.

Special thanks and acknowledgements to Pavel Logacev, who found some subtle bugs on the windows platform and gave excellent advice in general. Also thanks to Kevin Wu, who wrote earlier versions of the stemming and cross-validation code. And Georgiana Ifrim, of course, for the earlier version of the C++ code.

References

Ifrim, G., Bakir, G., & Weikum, G. (2008). Fast logistic regression for text categorization with variable-length n-grams. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 354-362.

Ifrim, G., & Wiuf, C. (2011). Bounded coordinate-descent for biological sequence classification in high dimensional predictor space. 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 708-716.

Jia, J., Miratrix, L., Yu, B., Gawalt, B., Ghaoui, El, L., Barnesmoore, L., & Clavier, S. (2014). Concise Comparative Summaries (CCS) of Large Text Corpora with a Human Experiment. The Annals of Applied Statistics, 8(1), 499-529.

Miratrix, L., & Ackerman, R. (2014). A method for conducting text-based sparse feature selection for interpretability of selected phrases.

bathtub *Sample of cleaned OSHA accident summaries.*

Description

bathtub consists of several accident reports plus a labeling with a +1 for any report that had been tagged as related to METHELYNE CHLORIDE.

Format

Corpus object from the tm package. Has a meta info of the METHELYNE CHLORIDE labeling called "meth.chl"

See Also

Other bathtub: dirtyBathtub

Examples

```
library( tm )
data( bathtub )
meta( bathtub, "meth.chl" )
```

build.corpus *Build a corpus that can be used in the textreg call.*

Description

Pre-building a corpus allows for calling multiple textregs without doing a lot of initial data processing (e.g., if you want to explore different ban lists or regularization parameters)

Usage

```
build.corpus(corpus, labeling, banned = NULL, verbosity = 1,
  token.type = "word")
```

Arguments

corpus	A list of strings or a corpus from the tm package.
labeling	A vector of +1/-1 or TRUE/FALSE indicating which documents are considered relevant and which are baseline. The +1/-1 can contain 0 whcih means drop the document.
banned	List of words that should be dropped from consideration.
verbosity	Level of output. 0 is no printed output.
token.type	"word" or "character" as tokens.

Details

See the bathtub vignette for more complete discussion of this method and the options you might pass to it.

A textreg.corpus object is not a `tm`-style corpus. In particular, all text pre-processing, etc., to text should be done to the data *before* building the textreg.corpus object.

Value

A `textreg.corpus` object.

Note

Unfortunately, the process of seperating out the textreg call and the build.corpus call is not quite as clean as one would hope. The build.corpus call moves the text into the C++ memory, but the way the search tree is built for the regression it is hard to salvage it across runs and so this is of limited use. In particular, the labeling and banned words cannot be easily changed. Future versions of the package would ideally remedy this.

Examples

```
data( testCorpora )
textreg( testCorpora$testI$corpus, testCorpora$testI$labelI, c(), C=1, verbosity=1 )
```

`calc.loss` *Calculate total loss of model (Squared hinge loss).*

Description

Calculate the loss for a model in predicting the -1/+1 labeling. If new text and labeling given, then calc loss on the new text and labeling. This can be useful for cross validation and train-test splits.

Usage

```
calc.loss(model.blob, new.text = NULL, new.labeling = NULL,
  loss = c("square.hinge", "square", "hinge"))
```

Arguments

`model.blob`	The model returned from `textreg`
`new.text`	New text (string or tm Corpus) to predict labeling for
`new.labeling`	Labeling to go with new text.
`loss`	Type of loss to calc for.

Value

Three numbers: total loss, loss from prediction, loss from penalty term

Examples

```
data( testCorpora )
res = textreg( c( "", "", "A", "A" ), c( -1, -1, 1, 1 ), C=1, Lq=1,
          convergence.threshold=0.00000001, verbosity=0 )
calc.loss( res )
calc.loss( res, new.text=c("A B C A"), new.labeling=c(1) )
```

`clean.text` *Clean text and get it ready for textreg.*

Description

Changes multiline documents to single line. Strips extra whitespace and punctuation. Changes digits to 'X's. Non-alpha characters converted to spaces.

Usage

```
clean.text(bigcorp)
```

Arguments

`bigcorp`	A tm Corpus object.

Examples

```
library( tm )
txt = c( "thhis s! and bonkus  4:33pm and Jan 3, 2015. ",
         "   big    space\n     dawg-ness?")
a <- clean.text( VCorpus( VectorSource( txt ) ) )
a[[1]]
```

`cluster.phrases` *Cluster phrases based on similarity of appearance.*

Description

Cluster phrases based on similarity of their appearance in the positive documents. Can also plot this if so desired.

Uses hclust() with the "ward.D" method on 1-S with S from make.similarity.matrix

Warning: for 'negative weight' phrases this method does not do well since it ignores negative documents.

Usage

```
cluster.phrases(result, num.groups = 5, plot = TRUE, yaxt = "n",
  ylab = "", sub = "", main = "Association of Phrases", ...)
```

Arguments

result	A similarity matrix from make.similarity.matrix call or an textreg.result object
num.groups	Number of groups to box.
plot	Actually plot clustering or just calculate it.
yaxt	Whether to include a y-axis
ylab	Label for y-axis
sub	Subtitle for plot
main	Title of plot.
...	Extra arguments to pass to the plot command. See par.

See Also

Other Phrase Vizualization: make.appearance.matrix, make.phrase.correlation.chart, make.similarity.matrix

---

convert.tm.to.character

Convert tm corpus to vector of strings.

Description

A utility function useful for testing and some dirty hacks. This is because the tm package doesn't leave vector corpora of strings alone anymore.

and so sometimes you need to convert your tm object to a string vector for various reasons, the main one being handing it to the C++ method. It is ugly, but so it goes.

It is therefore a possibly better decision to pass a filename to a plain-text file to the textreg call to be loaded by C++ directly. See textreg.

Usage

```
convert.tm.to.character(corpus)
```

Arguments

corpus	The tm corpus to convert.

Value

vector of character.

cpp_build.corpus *Driver function for the C++ function.*

Description

Given a labeling and a corpus, create a corpus object for use in textreg. Generally you should use the buildCorpus method, not this method.

Usage

```
cpp_build.corpus(corpus, labeling, banned = c(), params)
```

Arguments

corpus	A list of strings or a corpus from the tm package.
labeling	A vector of +1/-1 or TRUE/FALSE indicating which documents are considered relevant and which are baseline. The +1/-1 can contain 0 whcih means drop the document.
banned	List of words that should be dropped from consideration.
params	List of parameters to pass to the call.

Details

Warning: do not call directly. Use textreg instead

See Also

textreg, find_C_threshold

cpp_textreg *Driver function for the C++ function.*

Description

Given a labeling and a corpus, find phrases that predict this labeling. Generally you should use the textreg method, not this method.

Usage

```
cpp_textreg(corpus, params)
```

Arguments

corpus	A list of strings or a corpus from the tm package.
params	List of parameters to pass to the call.

Details

Warning: do not call directly. Use textreg instead

See Also

textreg, find_C_threshold

`dirtyBathtub` *Sample of raw-text OSHA accident summaries.*

Description

dirtyBathtub consists of the (more) raw data from which the `bathtub` dataset is derived.

Format

Dataframe. Has a meta info of the METHELYNE CHLORIDE labeling, plus 100s of other labels.

See Also

Other bathtub: `bathtub`

Examples

```
data( dirtyBathtub )
table( dirtyBathtub$fatality )
```

`find.CV.C` *K-fold cross-validation to determine optimal tuning parameter*

Description

Given a corpus, divide into K-folds and do test-train spilts averaged over the folds.

Usage

```
find.CV.C(corpus, labeling, banned, K = 5, length.out = 10,
  max_C = NULL, verbose = FALSE, ...)
```

Arguments

corpus	The text
labeling	The labeling
banned	The words to drop.
K	Number of folds for K-fold cross-validation
length.out	number of values of C to examine from 0 to max_C.
max_C	upper bound for tuning parameter; if NULL, sets max_C to threshold C
verbose	Print progress
...	parameters to be passed to the original textreg() function

Details

Increments tuning parameter, performs K-fold cross-validation on each C giving a profile of predictive power for different C.

Value

a dataframe containing the mean/standard error of out-of-sample predictions under K-Fold Cross-validation

See Also

make.CV.chart

---

find.threshold.C *Conduct permutation test on labeling to get null distribution of regularization parameter.*

Description

First determines what regularization will give null model on labeling. Then permutes labeling repeatidly, recording what regularization will give null model for permuted labeling. This allows for permutation-style inference on the relationship of the labeling to the text, and allows for appropriate selection of the tuning parameter.

Usage

```
find.threshold.C(corpus, labeling, banned = NULL, R = 0,
  objective.function = 2, a = 1, verbosity = 0,
  step.verbosity = verbosity, positive.only = FALSE,
  binary.features = FALSE, no.regularization = FALSE,
  positive.weight = 1, Lq = 2, min.support = 1, min.pattern = 1,
  max.pattern = 100, gap = 0, token.type = "word",
  convergence.threshold = 1e-04)
```

Arguments

corpus	A list of strings or a corpus from the tm package.
labeling	A vector of +1/-1 or TRUE/FALSE indicating which documents are considered relevant and which are baseline. The +1/-1 can contain 0 whcih means drop the document.
banned	List of words that should be dropped from consideration.
R	Number of times to scramble labling. 0 means use given labeling and find single C value.
objective.function	2 is hinge loss. 0 is something. 1 is something else.
a	What percent of regularization should be L1 loss (a=1) vs L2 loss (a=0)
verbosity	Level of output. 0 is no printed output.
step.verbosity	Level of output for line searches. 0 is no printed output.
positive.only	Disallow negative features if true
binary.features	Just code presence/absence of a feature in a document rather than count of feature in document.
no.regularization	Do not renormalize the features at all. (Lq will be ignored.)
positive.weight	Scale weight pf all positively marked documents by this value. (1, i.e., no scaling) is default) NOT FULLY IMPLEMENTED
Lq	Rescaling to put on the features (2 is standard). Can be from 1 up. Values above 10 invoke an infinity-norm.
min.support	Only consider phrases that appear this many times or more.
min.pattern	Only consider phrases this long or longer
max.pattern	Only consider phrases this short or shorter
gap	Allow phrases that have wildcard words in them. Number is how many wildcards in a row.
token.type	"word" or "character" as tokens.
convergence.threshold	How to decide if descent has converged. (Will go for three steps at this threshold to check for flatness.)

Details

Important: use the same parameter values as used with the original textreg call!

Value

A list of numbers (the Cs) R+1 long. The first number is always the C used for the _passed_ labeling. The remainder are shuffles.

Examples

```
data( testCorpora )
find.threshold.C( testCorpora$testI$corpus, testCorpora$testI$labelI, c(), R=5, verbosity=1 )
```

---

grab.fragments *Grab all fragments in a corpus with given phrase.*

Description

Search corpus for passed phrase, using some wildcard notation. Return snippits of text containing this phrase, with a specified number of characters before and after. This gives context for phrases in documents.

Use like this `frags = grab.fragments( "israel", bigcorp )`

Can take phrases such as 'appl+' which means any word starting with "appl." Can also take phrases such as "big * city" which consist of any three-word phrase with "big" as the first word and "city" as the third word.

If a pattern matches overlapping phrases, it will return the first but not the second.

Usage

```
grab.fragments(phrase, corp, char.before = 80,
  char.after = char.before, cap.phrase = TRUE, clean = FALSE)
```

Arguments

phrase	Phrase to find in corpus
corp	is a tm corpus
char.before	Number of characters of document to pull before phrase to give context.
char.after	As above, but trailing characters. Defaults to char.before value.
cap.phrase	TRUE if the phrase should be put in ALL CAPS. False if left alone.
clean	True means drop all documents without phrase from list. False means leave NULLs in the list.

Value

fragments in corp that have given phrase.List of lists. First list is len(corp) long with NULL values for documents without phrase, and lists of phrases for those documents with the phrase

Examples

```
library( tm )
docs = c( "987654321 test 123456789", "987654321 test test word 123456789",
       "test at start", "a test b", "this is a test", "without the t-word",
       "a test for you and a test for me" )
corpus <- VCorpus(VectorSource(docs))
grab.fragments( "test *", corpus, char.before=4, char.after=4 )
```

is.fragment.sample *Is object a fragment.sample object?*

Description

Is object a fragment.sample object?

Usage

```
is.fragment.sample(x)
```

Arguments

x	the object to check.

See Also

Other sample.fragments: print.fragment.sample, sample.fragments

is.textreg.corpus *Is object a textreg.corpus object?*

Description

Is object a textreg.corpus object?

Usage

```
is.textreg.corpus(x)
```

Arguments

x	the object to check.

See Also

Other textreg.corpus: print.textreg.corpus

is.textreg.result *Is object a textreg.result object?*

Description

Is object a textreg.result object?

Usage

```
is.textreg.result(x)
```

Arguments

x	the object to check.

See Also

Other textreg.result: phrases, print.textreg.result, reformat.textreg.model

list.table.chart *Graphic showing multiple word lists side-by-side.*

Description

This method basically makes a visual plot of a list table (which you call first).

Usage

```
list.table.chart(model.list, M = 100, linespace = 4, ytick = NULL,
  dates = NULL, main = paste("Word Appearance for ", attr(model.list,
  "topic"), "\n(Method: ", attr(model.list, "method"), ")", sep = ""),
  xlab = "Model", mar = c(3, 5, 2.5, 0.1), xaxt = "y",
  color.breaks = NULL, color.ramp = NULL, ...)
```

Arguments

model.list	Matrix (or data.frame) from the make.list.table call.
M	is the max number of words to show in chart
linespace	Where to space
ytick	Put y tick marks
dates	Dates to put on bottom
main	Main title
xlab	Label for x-axis
mar	Margin of plot (see par)

xaxt	Plot an x-axis (see par)
color.breaks	Cut-points (like on a histogram) defining the different color levels.
color.ramp	List of colors to use from lowest value (potentially negative weights) to highest. If both color.breaks and color.ramp passed, color.breaks is list one longer than color.ramp.
...	Extra arguments for the core image() call that plots the word weights.

See Also

make.list.table

`make.appearance.matrix` *Make phrase appearance matrix from textreg result.*

Description

Make matrix of which phrases appear in which of the positively marked documents.

Usage

```
make.appearance.matrix(result)
```

Arguments

result	An textreg.result object.

Details

Very similar to phrase.matrix, except this looks only at positively marked documents and just returns 1 or 0 on whether any document has a phrase, rather than giving counts. This is used by the clustering vizualizations and make.similarity.matrix.

Value

A $n X p$ matrix for n documents and p phrases in the result object. Each entry is a 0/1 value indicating presence of the given phrase in the given document.

See Also

make.similarity.matrix

phrase.matrix

Other Phrase Vizualization: `cluster.phrases`, `make.phrase.correlation.chart`, `make.similarity.matrix`

`make.count.table` *Count number of times documents have a given phrase.*

Description

Given a list of phrases, count how many documents they appear in and subdivide by positive and negative appearance.

Usage

```
make.count.table(phrases, labeling, corpus)
```

Arguments

`phrases`	List of strings
`labeling`	Vector of +1/0/-1 labels
`corpus`	A corpus object from tm package

Details

This method does not consider multiple counts of phrases within documents. Phrases can have wildcards and stemming notation. See `grab.fragments`.

Value

a dataframe of statistics. per.pos is the percent of the documents with the phrase that are positively labeled. per.tag is the percent of the positively labeled documents that have the phrase.

See Also

grab.fragments

Other textregCounting: `make.phrase.matrix`, `phrase.count`

Examples

```
library( tm )
data( bathtub )
lbl = meta( bathtub )$meth.chl
make.count.table( c("bathtub","strip+", "vapor *"), lbl, bathtub )
```

`make.CV.chart` *Plot K-fold cross validation curves*

Description

Make a loess curve with loess() to predict the test error for different values of C by interpolating the passed evaluated points on the tbl dataframe.

Usage

```
make.CV.chart(tbl, plot = TRUE, ...)
```

Arguments

`tbl`	Table from find.CV.C
`plot`	TRUE means plot the chart. False means do not, but return the optimal C
`...`	Parameters to the plot function

Details

Then plot the test error with SE bars for the cross validation. Also calculate the spot that is 1 SE above the minimum. Fits the points with loess lines so, in principle, few actually evaluated points are needed in evaluating the function. All a bit ad hoc and worthy of improvement.

Not particularly well implemented.

Value

invisible list of the minimum C value and the estimated test error for both the minimum and the predicted C corresponding to 1 SE above the minimum estimate.

See Also

find.CV.C

`make.list.table` *Collate multiple regression runs.*

Description

This method makes a table of several regression runs side by side. The table has rows being phrases and the columns being the regression runs. A number is usually the weight found for that word at that window. If multiple runs have the same phrase, row will have multiple entries.

Usage

```
make.list.table(result.list, model.names = names(result.list), M = 100,
  topic = "Summary Collection", method = c("rank", "weight", "count",
  "word"), annotate = TRUE)
```

Arguments

result.list	List of mix of textreg.result objects and dataframes with two columns of "word" and "weight". (The latter is for merging lists from other regression packages.)
model.names	Names of the textreg.result objects
M	maximum number of words to keep
topic	String A name for the topic
method	Different ways to sort the phrases. 'word' means make a list of words.
annotate	Add summary statistics to table such as phrase counts, etc.

Details

Method will also order rows based on calculated importance of phrases. Multiple ways of ordering are possible, via the method argument.

Finally, the table can be annotated with descriptive statistics of the phrases.

Warning: this method DOES NOT flip negative weight words (so negative weight usually look less imporant in the ordering).

See the bathtub vignette for an example of this method.

Value

If annotate = true, a dataframe with each column corresponding to an textreg.result object (and possibly extra columns about phrases). Otherwise a matrix of the word scores.

---

make.path.matrix *Generate matrix describing gradient descent path of textreg.*

Description

Generate a matrix of the sequence of features as they are introduced with the textreg gradient descent program along with their coefficients with each step of the descent.

Usage

```
make.path.matrix(res)
```

Arguments

res	A textreg.result object.

See Also

Other plot.path.matrix: path.matrix.chart, plot.textreg.result

Examples

```
data( testCorpora )
testI = testCorpora$testI
res = textreg( testI$corpus, testI$labelI, c("frog","goat","bat"), C=2, verbosity=0 )
make.path.matrix( res )
```

---

make.phrase.correlation.chart

Generate visualization of phrase overlap.

Description

Make simple chart showing which phrases have substantial overlap with other phrases.

Usage

```
make.phrase.correlation.chart(result, count = FALSE, num.groups = 5,
  use.corrplot = FALSE, ...)
```

Arguments

result	textreg.result object or a similarity matrix from a make.similarity.matrix call.
count	Display counts rather than similarity scores.
num.groups	Number of groups to box.
use.corrplot	Use the corrplot package of Taiyun Wei (will need to install it).
...	Extra arguments to pass to the image() plotting command. See par.

See Also

Other Phrase Vizualization: cluster.phrases, make.appearance.matrix, make.similarity.matrix

`make.phrase.matrix` *Make a table of where phrases appear in a corpus*

Description

Generate a n by p phrase count matrix, with n being number of documents and p being number of phrases: \tabularrrrrr 0 \tab 0 \tab 0 \tab 0 \tab 0 \cr 1 \tab 6 \tab 2 \tab 0 \tab 0 \cr 8 \tab 0 \tab 0 \tab 0 \tab 0 This is the phrase equivilent of a document-term matrix.

Usage

```
make.phrase.matrix(phrase_list, corpus)
```

Arguments

`phrase_list`	List of strings
`corpus`	A corpus object from tm package

Value

a n X p matrix, n being number of documents, p being number of phrases.

See Also

Other textregCounting: `make.count.table`, `phrase.count`

Examples

```
library( tm )
data( bathtub )
lbl = meta( bathtub )$meth.chl
head( make.phrase.matrix( c("bathtub","strip+", "vapor *"), bathtub ) )
```

`make.similarity.matrix`
Calculate similarity matrix for set of phrases.

Description

First get phrase appearance pattern on positive labeling (if not directly passed) and then calculate similarity matrix of how they are similar to each other.

Usage

```
make.similarity.matrix(result)
```

Arguments

result	An textreg.result object or a matrix from make.appearance.matrix

Details

Warning: for 'negative weight' phrases this method does not do well since it ignores negative documents.

See Also

Other Phrase Vizualization: cluster.phrases, make.appearance.matrix, make.phrase.correlation.chart

---

make_search_phrases *Convert phrases to appropriate search string.*

Description

Will change, e.g., "test * pig+" to appropriate regular expression to find in the text.

Usage

```
make_search_phrases(phrases)
```

Arguments

phrases	List of strings denoting the phrases to be searched for.

---

path.matrix.chart *Plot optimization path of textreg.*

Description

Plot the sequence of features as they are introduced with the textreg gradient descent program.

Usage

```
path.matrix.chart(path.matrix, xlab = "step", ylab = "beta",
  bty = "n", ...)
```

Arguments

path.matrix	Either a textreg.result object or a matrix from the make.path.matrix call.
xlab	Label for x axis
ylab	Label for y axis
bty	Box for plot
...	Arguments to be passed to the matplot() command.

See Also

Other plot.path.matrix: make.path.matrix, plot.textreg.result

---

phrase.count *Count phrase appearance.*

Description

Count number of times a _single_ phrase appears in the corpus

Usage

```
phrase.count(phrase, corp)
```

Arguments

phrase	A string
corp	A corpus object from tm package

See Also

Other textregCounting: make.count.table, make.phrase.matrix

Examples

```
library( tm )
data( bathtub )
phrase.count( "bathtub", bathtub )
```

---

phrase.matrix *Make matrix of where phrases appear in corpus.*

Description

Construct a $n X p$ matrix of appearances for selected phrases out of textreg object. n is the number of documents, p is the number of phrases selected in the result object 'rules.'

Usage

```
phrase.matrix(rules, n)
```

Arguments

rules	Either a textreg.result object or the rules list from such an object.
n	(Optional) If giving a rules list, the number of documents in corpus.

phrases *Get the phrases from the textreg.result object?*

Description

Get the phrases from the textreg.result object?

Usage

```
phrases(x)
```

Arguments

x	the object to check.

See Also

Other textreg.result: `is.textreg.result`, `print.textreg.result`, `reformat.textreg.model`

plot.textreg.result *Plot the sequence of features as they are introduced with the textreg gradient descent program.*

Description

Simply calls path.matrix.chart.

Usage

```
## S3 method for class 'textreg.result'
plot(x, ...)
```

Arguments

x	A textreg.result object.
...	Parameters to be passed to path.matrix.chart.

See Also

path.matrix.chart

Other plot.path.matrix: `make.path.matrix`, `path.matrix.chart`

predict.textreg.result

Predict labeling with the selected phrases.

Description

Given raw text and a textreg model, predict the labeling by counting appearance of relevant phrases in text and then multiplying these counts by the beta vector associated with the textreg object. Just like linear regression.

Usage

```
## S3 method for class 'textreg.result'
predict(object, new.text = NULL,
  return.matrix = FALSE, ...)
```

Arguments

object	A textreg.result object
new.text	If you want to predict for new text, pass it along.
return.matrix	TRUE means hand back the phrase appearance pattern matrix.
...	Nothing can be passed extra.

Value

Vector of predictions (numbers).

Examples

```
res = textreg( c( "", "", "A", "A" ), c( -1, -1, 1, 1 ),
      C=1, Lq=1, convergence.threshold=0.00000001, verbosity=0 )
predict( res )
predict( res, new.text=c("A B C A") )
```

print.fragment.sample *Pretty print results of phrase sampling object.*

Description

Pretty print results of phrase sampling object.

Usage

```
## S3 method for class 'fragment.sample'
print(x, ...)
```

Arguments

x	A fragment.sample object.
...	No extra options passed.

See Also

Other sample.fragments: is.fragment.sample, sample.fragments

---

print.textreg.corpus *Pretty print textreg corpus object*

Description

Pretty print textreg corpus object

Usage

```
## S3 method for class 'textreg.corpus'
print(x, ...)
```

Arguments

x	A textreg.corpus object.
...	No extra options passed.

See Also

Other textreg.corpus: is.textreg.corpus

---

print.textreg.result *Pretty print results of textreg regression.*

Description

You can also reformat an textreg.result to get simpler diagnostics via reformat.textreg.model.

Usage

```
## S3 method for class 'textreg.result'
print(x, simple = FALSE, ...)
```

Arguments

x	A textreg.result object.
simple	TRUE means print out simpler results. False includes some ugly detail.
...	No extra options passed.

See Also

reformat.textreg.model

Other textreg.result: is.textreg.result, phrases, reformat.textreg.model

---

`reformat.textreg.model` *Clean up output from textreg.*

Description

Calculate some useful statistics (percents, etc) and return as dataframe.

Usage

```
reformat.textreg.model(model, short = TRUE)
```

Arguments

model	The model returned from textreg
short	True if the output should be abbrviated for easy consumption.

Value

Dataframe with statistics on the terms in the model

See Also

Other textreg.result: is.textreg.result, phrases, print.textreg.result

---

`sample.fragments` *Sample fragments of text to contextualize a phrase.*

Description

Take a phrase, a labeling and a corpus and return text fragments containing that phrase.

Grab all phrases and then give sample of N from positive class and N from negative class. Sampling is to first sample from documents and then sample a random phrase from each of those documents.

Usage

```
sample.fragments(phrases, labeling, corp, N = 10, char.before = 80,
  char.after = char.before, metainfo = NULL)
```

Arguments

phrases	Phrases to examine (a list of strings)
labeling	– a vector of the same length as the corpus
corp	Corpus object (tm package Corpus object)
N	size of sample to make.
char.before	Number of characters of document to pull before phrase to give context.
char.after	As above, but trailing characters. Defaults to char.before value.
metainfo	– extra string to add to the printout for clarity if many such printouts are being generated.

See Also

Other sample.fragments: is.fragment.sample, print.fragment.sample

Examples

```
library( tm )
data( bathtub )
sample.fragments( "bathtub", meta(bathtub)$meth.chl, bathtub )
```

---

save.corpus.to.files *Save corpus to text (and RData) file.*

Description

Small utility to save a corpus to a text file (and RData file) for ease of use.

It is possibly recommended to pass a filename to the C++ function textreg rather than the entire corpus for large text since I believe it will otherwise copy over everything due to the coder's (my) poor understanding of how RCpp converts objects.

Usage

```
save.corpus.to.files(bigcorp, filename = "corpus")
```

Arguments

bigcorp	A tm Corpus object.
filename	The first part of the filename. A rda and txt extension will be appended to the two generated files.

`stem.corpus` *Step corpus with annotation.*

Description

Given a `tm`-package VCorpus of original text, returns a VCorpus of stemmed text with '+' appended to all stemmed words.

Usage

```
stem.corpus(corpus, verbose = TRUE)
```

Arguments

`corpus`	Original text
`verbose`	True means print out text progress bar so you can watch progress.

Details

This is non-optimized code that is expensive to run. First the stemmer chops words. Then this method passes through and adds a "+" to all chopped words, and builds a list of stems. Finally, the method passes through and adds a "+" to all stems found without a suffix.

So, e.g., goblins and goblin will both be transformed to "goblin+".

Adding the '+' makes stemmed text more readible.

Code based on code from Kevin Wu, UC Berkeley Undergrad Thesis 2014.

Requires, via the tm package, the SnowballC package.

Warning: Do not use this on a `textreg.corpus` object. Do to text before building the `textreg.corpus` object.

Examples

```
library( tm )
texts <- c("texting goblins the dagger", "text these goblins",
           "texting 3 goblins appl daggers goblining gobble")
corpus <- VCorpus(VectorSource(texts))
stemmed_corpus<-stem.corpus(corpus, verbose=FALSE)
inspect( stemmed_corpus[[2]] )
```

testCorpora *Some small, fake test corpora.*

Description

A list of several fake documents along with some labeling schemes primarily used by the unit testing code. Also used in some examples.

Format

A list of dataframes

textreg *Sparse regression of labeling vector onto all phrases in a corpus.*

Description

Given a labeling and a corpus, find phrases that predict this labeling. This function calls a C++ function that builds a tree of phrases and searches it using greedy coordinate descent to solve the optimization problem associated with the associated sparse regression.

Usage

```
textreg(corpus, labeling, banned = NULL, objective.function = 2,
  C = 1, a = 1, maxIter = 40, verbosity = 1,
  step.verbosity = verbosity, positive.only = FALSE,
  binary.features = FALSE, no.regularization = FALSE,
  positive.weight = 1, Lq = 2, min.support = 1, min.pattern = 1,
  max.pattern = 100, gap = 0, token.type = "word",
  convergence.threshold = 1e-04)
```

Arguments

corpus	A list of strings or a corpus from the tm package.
labeling	A vector of +1/-1 or TRUE/FALSE indicating which documents are considered relevant and which are baseline. The +1/-1 can contain 0 whcih means drop the document.
banned	List of words that should be dropped from consideration.
objective.function	2 is hinge loss. 0 is something. 1 is something else.
C	The regularization term. 0 is no regularization.
a	What percent of regularization should be L1 loss (a=1) vs L2 loss (a=0)
maxIter	Number of gradient descent steps to take (not including intercept adjustments)

`verbosity` Level of output. 0 is no printed output.

`step.verbosity` Level of output for line searches. 0 is no printed output.

`positive.only` Disallow negative features if true

`binary.features`
Just code presence/absence of a feature in a document rather than count of feature in document.

`no.regularization`
Do not renormalize the features at all. (Lq will be ignored.)

`positive.weight`
Scale weight pf all positively marked documents by this value. (1, i.e., no scaling) is default) NOT FULLY IMPLEMENTED

`Lq` Rescaling to put on the features (2 is standard). Can be from 1 up. Values above 10 invoke an infinity-norm.

`min.support` Only consider phrases that appear this many times or more.

`min.pattern` Only consider phrases this long or longer

`max.pattern` Only consider phrases this short or shorter

`gap` Allow phrases that have wildcard words in them. Number is how many wildcards in a row.

`token.type` "word" or "character" as tokens.

`convergence.threshold`
How to decide if descent has converged. (Will go for three steps at this threshold to check for flatness.)

Details

See the bathtub vignette for more complete discussion of this method and the options you might pass to it.

Value

A `textreg.result` object.

Examples

```
data( testCorpora )
textreg( testCorpora$testI$corpus, testCorpora$testI$labelI, c(), C=1, verbosity=1 )
```

tm_gregexpr *Call gregexpr on the content of a tm Corpus.*

Description

Pull out content of a tm corpus and call gregexpr on that content represented as a list of character strings.

Usage

```
tm_gregexpr(pattern, corpus, ignore.case = FALSE, perl = FALSE,
  fixed = FALSE, useBytes = FALSE)
```

Arguments

pattern	See gregexpr
corpus	Either a character vector or tm Corpus object.
ignore.case	See gregexpr
perl	See gregexpr
fixed	See gregexpr
useBytes	See gregexpr

Details

If 'corpus' is already a character vector, it just calls gregexpr with no fuss (or warning).

Value

This method gives results exactly as if gregexpr were called on the Corpus represented as a list of strings.

See gregexpr.

See Also

gregexpr

Index