Package 'quanteda.textmodels'

April 11, 2024

Type Package

Title Scaling Models and Classifiers for Textual Data

Version 0.9.7

Description Scaling models and classifiers for sparse matrix objects representing textual data in the form of a document-feature matrix. Includes original implementations of 'Laver', 'Benoit', and Garry's (2003) <doi:10.1017/S0003055403000698>, 'Wordscores' model, the Perry and 'Benoit' (2017) <doi:10.48550/arXiv.1710.08963> class affinity scaling model, and the 'Slapin' and 'Proksch' (2008) <doi:10.1111/j.1540-5907.2008.00338.x> 'wordfish' model, as well as methods for correspondence analysis, latent semantic analysis, and fast Naive Bayes and linear 'SVMs' specially designed for sparse textual data.

Depends R (>= 3.1.0), methods

Imports glmnet, LiblineaR, Matrix (>= 1.2), quanteda (>= 4.0.0), RSpectra, Rcpp (>= 0.12.12), SparseM, stringi

Suggests ca, covr, fastNaiveBayes, knitr, lsa, microbenchmark, naivebayes, quanteda.textplots, spelling, testthat, rmarkdown

LinkingTo Rcpp, RcppArmadillo (>= 0.7.600.1.0), quanteda

URL https://github.com/quanteda/quanteda.textmodels

License GPL-3

Encoding UTF-8

LazyData true

Language en-GB

RoxygenNote 7.3.1

Collate 'RcppExports.R' 'quanteda.textmodels-package.R' 'data-documentation.R' 'textmodel-methods.R' 'textmodel_affinity.R' 'textmodel_ca.R' 'textmodel_lsa.R' 'textmodel_lr.R' 'textmodel_nb.R' 'textmodel_svm.R' 'textmodel_svmlin.R' 'textmodel_wordfish.R' 'textmodel_wordscores.R' 'textplot_influence.R' 'utils.R'

VignetteBuilder knitr

NeedsCompilation yes

Author Kenneth Benoit [cre, aut, cph]
(<https://orcid.org/0000-0002-0797-564X>),
Kohei Watanabe [aut] (<https://orcid.org/0000-0001-6519-5265>),
Haiyan Wang [aut] (<https://orcid.org/0000-0003-4992-4311>),
Patrick O. Perry [aut] (<https://orcid.org/0000-0001-7460-127X>),
Benjamin Lauderdale [aut] (<https://orcid.org/0000-0003-3090-0969>),
Johannes Gruber [aut] (<https://orcid.org/0000-0001-9177-1772>),
William Lowe [aut] (<https://orcid.org/0000-0002-1549-6163>),
Vikas Sindhwani [cph] (authored svmlin C++ source code),
European Research Council [fnd] (ERC-2011-StG 283794-QUANTESS)

Maintainer Kenneth Benoit <kbenoit@lse.ac.uk>

Repository CRAN

Date/Publication 2024-04-11 08:10:11 UTC

R topics documented:

`data_corpus_dailnoconf1991`

Confidence debate from 1991 Irish Parliament

Description

Texts of speeches from a no-confidence motion debated in the Irish Dáil from 16-18 October 1991 over the future of the Fianna Fail-Progressive Democrat coalition. (See Laver and Benoit 2002 for details.)

Usage

```
data_corpus_dailnoconf1991
```

Format

data_corpus_dailnoconf1991 is a corpus with 58 texts, including docvars for name, party, and position.

Source

https://www.oireachtas.ie/en/debates/debate/dail/1991-10-16/10/

References

Laver, M. & Benoit, K.R. (2002). Locating TDs in Policy Spaces: Wordscoring Dáil Speeches. *Irish Political Studies*, 17(1), 59–73.

Laver, M., Benoit, K.R., & Garry, J. (2003). Estimating Policy Positions from Political Text using Words as Data. *American Political Science Review*, 97(2), 311–331.

Examples

```
## Not run:
library("quanteda")
data_dfm_dailnoconf1991 <- data_corpus_dailnoconf1991 %>%
    tokens(remove_punct = TRUE) %>%
    dfm()
tmod <- textmodel_affinity(data_dfm_dailnoconf1991,
                           c("Govt", "Opp", "Opp", rep(NA, 55)))
(pred <- predict(tmod))
dat <-
    data.frame(party = as.character(docvars(data_corpus_dailnoconf1991, "party")),
               govt = coef(pred)[, "Govt"],
               position = as.character(docvars(data_corpus_dailnoconf1991, "position")))
bymedian <- with(dat, reorder(paste(party, position), govt, median))
oldpar <- par(no.readonly = TRUE)
par(mar = c(5, 6, 4, 2) + .1)
boxplot(govt ~ bymedian, data = dat,
        horizontal = TRUE, las = 1,
        xlab = "Degree of support for government",
        ylab = "")
abline(h = 7.5, col = "red", lty = "dashed")
text(c(0.9, 0.9), c(8.5, 6.5), c("Goverment", "Opposition"))
par(oldpar)

## End(Not run)
```

---

data_corpus_EPcoaldebate

Crowd-labelled sentence corpus from a 2010 EP debate on coal subsidies

---

Description

A multilingual text corpus of speeches from a European Parliament debate on coal subsidies in 2010, with individual crowd codings as the unit of observation. The sentences are drawn from officially translated speeches from a debate over a European Parliament debate concerning a Commission report proposing an extension to a regulation permitting state aid to uncompetitive coal mines.

Each speech is available in six languages: English, German, Greek, Italian, Polish and Spanish. The unit of observation is the individual crowd coding of each natural sentence. For more information on the coding approach see Benoit et al. (2016).

Usage

```
data_corpus_EPcoaldebate
```

Format

The corpus consists of 16,806 documents (i.e. codings of a sentence) and includes the following document-level variables:

sentence_id character; a unique identifier for each sentence

crowd_subsidy_label factor; whether a coder labelled the sentence as "Pro-Subsidy", "Anti-Subsidy" or "Neutral or inapplicable"

language factor; the language (translation) of the speech

name_last character; speaker's last name

name_first character; speaker's first name

ep_group factor; abbreviation of the EP party group of the speaker

country factor; the speaker's country of origin

vote factor; the speaker's vote on the proposal (For/Against/Abstain/NA)

coder_id character; a unique identifier for each crowd coder

coder_trust numeric; the "trust score" from the Crowdflower platform used to code the sentences, which can theoretically range between 0 and 1. Only coders with trust scores above 0.8 are included in the corpus.

A corpus object.

References

Benoit, K., Conway, D., Lauderdale, B.E., Laver, M., & Mikhaylov, S. (2016). Crowd-sourced Text Analysis: Reproducible and Agile Production of Political Data. *American Political Science Review*, 100,(2), 278–295. doi:10.1017/S0003055416000058

data_corpus_irishbudget2010

Irish budget speeches from 2010

Description

Speeches and document-level variables from the debate over the Irish budget of 2010.

Usage

```
data_corpus_irishbudget2010
```

Format

The corpus object for the 2010 budget speeches, with document-level variables for year, debate, serial number, first and last name of the speaker, and the speaker's party.

Details

At the time of the debate, Fianna Fáil (FF) and the Greens formed the government coalition, while Fine Gael (FG), Labour (LAB), and Sinn Féin (SF) were in opposition.

Source

Dáil Éireann Debate, Budget Statement 2010. 9 December 2009. vol. 697, no. 3.

References

Lowe, W. & Benoit, K.R. (2013). Validating Estimates of Latent Traits From Textual Data Using Human Judgment as a Benchmark. *Political Analysis*, 21(3), 298–313. doi:10.1093/pan/mpt002.

data_corpus_moviereviews

Movie reviews with polarity from Pang and Lee (2004)

Description

A corpus object containing 2,000 movie reviews classified by positive or negative sentiment.

Usage

```
data_corpus_moviereviews
```

Format

The corpus includes the following document variables:

sentiment factor indicating whether a review was manually classified as positive `pos` or negative `neg`.

id1 Character counting the position in the corpus.

id2 Random number for each review.

Details

For more information, see `cat(meta(data_corpus_moviereviews, "readme"))`.

Source

https://www.cs.cornell.edu/people/pabo/movie-review-data/

References

Pang, B., Lee, L. (2004) "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts.", Proceedings of the ACL.

Examples

```
# check polarities
table(data_corpus_moviereviews$sentiment)

# make the data into sentences, because each line is a sentence
data_corpus_moviereviewsents <-
    quanteda::corpus_segment(data_corpus_moviereviews, "\n", extract_pattern = FALSE)
print(data_corpus_moviereviewsents, max_ndoc = 3)
```

---

textmodel_affinity *Class affinity maximum likelihood text scaling model*

Description

`textmodel_affinity()` implements the maximum likelihood supervised text scaling method described in Perry and Benoit (2017).

Usage

```
textmodel_affinity(
  x,
  y,
  exclude = NULL,
  smooth = 0.5,
  ref_smooth = 0.5,
  verbose = quanteda_options("verbose")
)
```

Arguments

x	the dfm or bootstrap_dfm object on which the model will be fit. Does not need to contain only the training documents, since the index of these will be matched automatically.
y	vector of training classes/scores associated with each document identified in data
exclude	a set of words to exclude from the model
smooth	a smoothing parameter for class affinities; defaults to 0.5 (Jeffreys prior). A plausible alternative would be 1.0 (Laplace prior).
ref_smooth	a smoothing parameter for token distributions; defaults to 0.5
verbose	logical; if TRUE print diagnostic information during fitting.

Value

A textmodel_affinity class list object, with elements:

- smooth a numeric vector of length two for the smoothing parameters smooth and ref_smooth x the input model matrix x y the vector of class training labels y p a feature $\times$ class sparse matrix of estimated class affinities
- support logical vector indicating whether a feature was included in computing class affinities
- call the model call

Author(s)

Patrick Perry and Kenneth Benoit

References

Perry, P.O. & Benoit, K.R. (2017). Scaling Text with the Class Affinity Model. doi:10.48550/arXiv.1710.08963.

See Also

predict.textmodel_affinity() for methods of applying a fitted textmodel_affinity() model object to predict quantities from (other) documents.

Examples

```
(af <- textmodel_affinity(quanteda::data_dfm_lbgexample, y = c("L", NA, NA, NA, "R", NA)))
predict(af)
predict(af, newdata = quanteda::data_dfm_lbgexample[6, ])

## Not run:
# compute bootstrapped SEs
dfmat <- quanteda::bootstrap_dfm(data_corpus_dailnoconf1991, n = 10, remove_punct = TRUE)
textmodel_affinity(dfmat, y = c("Govt", "Opp", "Opp", rep(NA, 55)))

## End(Not run)
```

`textmodel_ca` *Correspondence analysis of a document-feature matrix*

Description

`textmodel_ca` implements correspondence analysis scaling on a dfm. The method is a fast/sparse version of function ca.

Usage

```
textmodel_ca(x, smooth = 0, nd = NA, sparse = FALSE, residual_floor = 0.1)
```

Arguments

`x`	the dfm on which the model will be fit
`smooth`	a smoothing parameter for word counts; defaults to zero.
`nd`	Number of dimensions to be included in output; if `NA` (the default) then the maximum possible dimensions are included.
`sparse`	retains the sparsity if set to `TRUE`; set it to `TRUE` if `x` (the dfm) is too big to be allocated after converting to dense
`residual_floor`	specifies the threshold for the residual matrix for calculating the truncated svd.Larger value will reduce memory and time cost but might reduce accuracy; only applicable when `sparse = TRUE`

Details

svds in the **RSpectra** package is applied to enable the fast computation of the SVD.

Value

`textmodel_ca()` returns a fitted CA textmodel that is a special class of **ca** object.

Note

You may need to set `sparse = TRUE`) and increase the value of `residual_floor` to ignore less important information and hence to reduce the memory cost when you have a very big dfm. If your attempt to fit the model fails due to the matrix being too large, this is probably because of the memory demands of computing the $V \times V$ residual matrix. To avoid this, consider increasing the value of `residual_floor` by 0.1, until the model can be fit.

Author(s)

Kenneth Benoit and Haiyan Wang

References

Nenadic, O. & Greenacre, M. (2007). Correspondence Analysis in R, with Two- and Three-dimensional Graphics: The ca package. *Journal of Statistical Software*, 20(3). doi:10.18637/jss.v020.i03

See Also

`coef.textmodel_lsa()`, ca

Examples

```
library("quanteda")
dfmat <- dfm(tokens(data_corpus_irishbudget2010))
tmod <- textmodel_ca(dfmat)
summary(tmod)
```

---

textmodel_lr *Logistic regression classifier for texts*

Description

Fits a fast penalized maximum likelihood estimator to predict discrete categories from sparse dfm objects. Using the **glmnet** package, the function computes the regularization path for the lasso or elasticnet penalty at a grid of values for the regularization parameter lambda. This is done automatically by testing on several folds of the data at estimation time.

Usage

```
textmodel_lr(x, y, ...)
```

Arguments

x	the dfm on which the model will be fit. Does not need to contain only the training documents.
y	vector of training labels associated with each document identified in `train`. (These will be converted to factors if not already factors.)
...	additional arguments passed to `cv.glmnet()`

Value

an object of class `textmodel_lr`, a list containing:

- `x`, `y` the input model matrix and input training class labels
- `algorithm` character; the type and family of logistic regression model used in calling `cv.glmnet()`
- `type` the type of associated with `algorithm`
- `classnames` the levels of training classes in `y`
- `lrfitted` the fitted model object from `cv.glmnet()`
- `call` the model call

References

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software* 33(1), 1-22. doi:10.18637/jss.v033.i01

See Also

`cv.glmnet()`, `predict.textmodel_lr()`, `coef.textmodel_lr()`

Examples

```
## Example from 13.1 of _An Introduction to Information Retrieval_
library("quanteda")
corp <- corpus(c(d1 = "Chinese Beijing Chinese",
                 d2 = "Chinese Chinese Shanghai",
                 d3 = "Chinese Macao",
                 d4 = "Tokyo Japan Chinese",
                 d5 = "London England Chinese",
                 d6 = "Chinese Chinese Chinese Tokyo Japan"),
               docvars = data.frame(train = factor(c("Y", "Y", "Y", "N", "N", NA))))
dfmat <- dfm(tokens(corp), tolower = FALSE)

## simulate bigger sample as classification on small samples is problematic
set.seed(1)
dfmat <- dfm_sample(dfmat, 50, replace = TRUE)

## train model
(tmod1 <- textmodel_lr(dfmat, docvars(dfmat, "train")))
summary(tmod1)
coef(tmod1)

## predict probability and classes
predict(tmod1, type = "prob")
predict(tmod1)
```

---

textmodel_lsa *Latent Semantic Analysis*

Description

Fit the Latent Semantic Analysis scaling model to a dfm, which may be weighted (for instance using `quanteda::dfm_tfidf()`).

Usage

```
textmodel_lsa(x, nd = 10, margin = c("both", "documents", "features"))
```

Arguments

x	the dfm on which the model will be fit
nd	the number of dimensions to be included in output
margin	margin to be smoothed by the SVD

Details

svds in the **RSpectra** package is applied to enable the fast computation of the SVD.

Value

a textmodel_lsa class object, a list containing:

- sk a numeric vector containing the d values from the SVD
- docs document coordinates from the SVD (u)
- features feature coordinates from the SVD (v)
- matrix_low_rank the multiplication of udv'
- data the input data as a CSparseMatrix from the **Matrix** package

Note

The number of dimensions nd retained in LSA is an empirical issue. While a reduction in k can remove much of the noise, keeping too few dimensions or factors may lose important information.

Author(s)

Haiyan Wang and Kohei Watanabe

References

Rosario, B. (2000). Latent Semantic Indexing: An Overview. *Technical report INFOSYS 240 Spring Paper, University of California, Berkeley.*

Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., & Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6): 391.

See Also

predict.textmodel_lsa(), coef.textmodel_lsa()

Examples

```
library("quanteda")
dfmat <- dfm(tokens(data_corpus_irishbudget2010))
# create an LSA space and return its truncated representation in the low-rank space
tmod <- textmodel_lsa(dfmat[1:10, ])
head(tmod$docs)

# matrix in low_rank LSA space
```

```
tmod$matrix_low_rank[,1:5]

# fold queries into the space generated by dfmat[1:10,]
# and return its truncated versions of its representation in the new low-rank space
pred <- predict(tmod, newdata = dfmat[11:14, ])
pred$docs_newspace
```

`textmodel_nb` *Naive Bayes classifier for texts*

Description

Fit a multinomial or Bernoulli Naive Bayes model, given a dfm and some training labels.

Usage

```
textmodel_nb(
  x,
  y,
  smooth = 1,
  prior = c("uniform", "docfreq", "termfreq"),
  distribution = c("multinomial", "Bernoulli")
)
```

Arguments

`x`	the dfm on which the model will be fit. Does not need to contain only the training documents.
`y`	vector of training labels associated with each document identified in `train`. (These will be converted to factors if not already factors.)
`smooth`	smoothing parameter for feature counts, added to the feature frequency totals by training class
`prior`	prior distribution on texts; one of `"uniform"`, `"docfreq"`, or `"termfreq"`. See Prior Distributions below.
`distribution`	count model for text features, can be `multinomial` or `Bernoulli`. To fit a "binary multinomial" model, first convert the dfm to a binary matrix using `[quanteda::dfm_weight](x, sc`

Value

`textmodel_nb()` returns a list consisting of the following (where I is the total number of documents, J is the total number of features, and k is the total number of training classes):

`call`	original function call
`param`	$k \times V$; class conditional posterior estimates
`x`	the $N \times V$ training dfm `x`

y	the N-length y training class vector, where NAs will not be used will be retained in the saved x matrix
distribution	character; the distribution of x for the NB model
priors	numeric; the class prior probabilities
smooth	numeric; the value of the smoothing parameter

Prior distributions

Prior distributions refer to the prior probabilities assigned to the training classes, and the choice of prior distribution affects the calculation of the fitted probabilities. The default is uniform priors, which sets the unconditional probability of observing the one class to be the same as observing any other class.

"Document frequency" means that the class priors will be taken from the relative proportions of the class documents used in the training set. This approach is so common that it is assumed in many examples, such as the worked example from Manning, Raghavan, and Schütze (2008) below. It is not the default in **quanteda**, however, since there may be nothing informative in the relative numbers of documents used to train a classifier other than the relative availability of the documents. When training classes are balanced in their number of documents (usually advisable), however, then the empirically computed "docfreq" would be equivalent to "uniform" priors.

Setting prior to "termfreq" makes the priors equal to the proportions of total feature counts found in the grouped documents in each training class, so that the classes with the largest number of features are assigned the largest priors. If the total count of features in each training class was the same, then "uniform" and "termfreq" would be the same.

Smoothing parameter

The smooth value is added to the feature frequencies, aggregated by training class, to avoid zero frequencies in any class. This has the effect of giving more weight to infrequent term occurrences.

Author(s)

Kenneth Benoit

References

Manning, C.D., Raghavan, P., & Schütze, H. (2008). *An Introduction to Information Retrieval*. Cambridge: Cambridge University Press (Chapter 13). Available at https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.

Jurafsky, D. & Martin, J.H. (2018). From *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Draft of September 23, 2018 (Chapter 6, Naive Bayes). Available at https://web.stanford.edu/~jurafsky/slp3/.

See Also

predict.textmodel_nb()

Examples

```
## Example from 13.1 of _An Introduction to Information Retrieval_
library("quanteda")
txt <- c(d1 = "Chinese Beijing Chinese",
         d2 = "Chinese Chinese Shanghai",
         d3 = "Chinese Macao",
         d4 = "Tokyo Japan Chinese",
         d5 = "Chinese Chinese Chinese Tokyo Japan")
x <- dfm(tokens(txt), tolower = FALSE)
y <- factor(c("Y", "Y", "Y", "N", NA), ordered = TRUE)

## replicate IIR p261 prediction for test set (document 5)
(tmod1 <- textmodel_nb(x, y, prior = "docfreq"))
summary(tmod1)
coef(tmod1)
predict(tmod1, type = "prob")
predict(tmod1)

# contrast with other priors
predict(textmodel_nb(x, y, prior = "uniform"))
predict(textmodel_nb(x, y, prior = "termfreq"))

## replicate IIR p264 Bernoulli Naive Bayes
tmod2 <- textmodel_nb(x, y, distribution = "Bernoulli", prior = "docfreq")
predict(tmod2, newdata = x[5, ], type = "prob")
predict(tmod2, newdata = x[5, ])
```

---

textmodel_svm *Linear SVM classifier for texts*

Description

Fit a fast linear SVM classifier for texts, using the **LiblineaR** package.

Usage

```
textmodel_svm(
  x,
  y,
  weight = c("uniform", "docfreq", "termfreq"),
  type = 1,
  ...
)
```

Arguments

x	the dfm on which the model will be fit. Does not need to contain only the training documents.

y	vector of training labels associated with each document identified in train. (These will be converted to factors if not already factors.)
weight	weights for different classes for imbalanced training sets, passed to wi in LiblineaR::LiblineaR(). "uniform" uses default; "docfreq" weights by the number of training examples, and "termfreq" by the relative sizes of the training classes in terms of their total lengths in tokens.
type	argument passed to the type argument in LiblineaR::LiblineaR(); default is 1 for L2-regularized L2-loss support vector classification (dual)
...	additional arguments passed to LiblineaR::LiblineaR()

Value

an object of class textmodel_svm, a list containing:

- x, y, weights, type: argument values from the call parameters
- algorithm character label of the algorithm used in the call to LiblineaR::LiblineaR()
- classnames levels of y
- bias the value of Bias returned from LiblineaR::LiblineaR()
- svmlinfitted the fitted model object passed from the call to LiblineaR::LiblineaR()]
- call the model call

References

R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, and C. J. Lin. (2008) LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research* 9: 1871-1874. https://www.csie.ntu.edu.tw/~cjlin/liblinear/.

See Also

LiblineaR::LiblineaR() predict.textmodel_svm()

Examples

```
# use party leaders for govt and opposition classes
library("quanteda")
docvars(data_corpus_irishbudget2010, "govtopp") <-
    c(rep(NA, 4), "Gov", "Opp", NA, "Opp", NA, NA, NA, NA, NA, NA)
dfmat <- dfm(tokens(data_corpus_irishbudget2010))
tmod <- textmodel_svm(dfmat, y = dfmat$govtopp)
predict(tmod)

# multiclass problem - all party leaders
tmod2 <- textmodel_svm(dfmat,
    y = c(rep(NA, 3), "SF", "FF", "FG", NA, "LAB", NA, NA, "Green", rep(NA, 3)))
predict(tmod2)
```

`textmodel_wordfish` *Wordfish text model*

Description

Estimate Slapin and Proksch's (2008) "wordfish" Poisson scaling model of one-dimensional document positions using conditional maximum likelihood.

Usage

```
textmodel_wordfish(
  x,
  dir = c(1, 2),
  priors = c(Inf, Inf, 3, 1),
  tol = c(1e-06, 1e-08),
  dispersion = c("poisson", "quasipoisson"),
  dispersion_level = c("feature", "overall"),
  dispersion_floor = 0,
  abs_err = FALSE,
  residual_floor = 0.5
)
```

Arguments

`x`	the dfm on which the model will be fit
`dir`	set global identification by specifying the indexes for a pair of documents such that $\hat{\theta}_{dir[1]} < \hat{\theta}_{dir[2]}$.
`priors`	prior precisions for the estimated parameters α_i, ψ_j, β_j, and θ_i, where i indexes documents and j indexes features
`tol`	tolerances for convergence. The first value is a convergence threshold for the log-posterior of the model, the second value is the tolerance in the difference in parameter values from the iterative conditional maximum likelihood (from conditionally estimating document-level, then feature-level parameters).
`dispersion`	sets whether a quasi-Poisson quasi-likelihood should be used based on a single dispersion parameter (`"poisson"`), or quasi-Poisson (`"quasipoisson"`)
`dispersion_level`	sets the unit level for the dispersion parameter, options are `"feature"` for term-level variances, or `"overall"` for a single dispersion parameter
`dispersion_floor`	constraint for the minimal underdispersion multiplier in the quasi-Poisson model. Used to minimize the distorting effect of terms with rare term or document frequencies that appear to be severely underdispersed. Default is 0, but this only applies if `dispersion = "quasipoisson"`.
`abs_err`	specifies how the convergence is considered
`residual_floor`	specifies the threshold for residual matrix when calculating the svds, only applies when `sparse = TRUE`

Details

The returns match those of Will Lowe's R implementation of `wordfish` (see the austin package), except that here we have renamed `words` to be `features`. (This return list may change.) We have also followed the practice begun with Slapin and Proksch's early implementation of the model that used a regularization parameter of $\text{se}(\sigma) = 3$, through the third element in `priors`.

Value

An object of class `textmodel_fitted_wordfish`. This is a list containing:

`dir`	global identification of the dimension
`theta`	estimated document positions
`alpha`	estimated document fixed effects
`beta`	estimated feature marginal effects
`psi`	estimated word fixed effects
`docs`	document labels
`features`	feature labels
`sigma`	regularization parameter for betas in Poisson form
`ll`	log likelihood at convergence
`se.theta`	standard errors for theta-hats
`x`	dfm to which the model was fit

Note

In the rare situation where a warning message of "The algorithm did not converge." shows up, removing some documents may work.

Author(s)

Benjamin Lauderdale, Haiyan Wang, and Kenneth Benoit

References

Slapin, J. & Proksch, S.O. (2008). A Scaling Model for Estimating Time-Series Party Positions from Texts. doi:10.1111/j.15405907.2008.00338.x. *American Journal of Political Science*, 52(3), 705–772.

Lowe, W. & Benoit, K.R. (2013). Validating Estimates of Latent Traits from Textual Data Using Human Judgment as a Benchmark. doi:10.1093/pan/mpt002. *Political Analysis*, 21(3), 298–313.

See Also

`predict.textmodel_wordfish()`

Examples

```
(tmod1 <- textmodel_wordfish(quanteda::data_dfm_lbgexample, dir = c(1,5)))
summary(tmod1, n = 10)
coef(tmod1)
predict(tmod1)
predict(tmod1, se.fit = TRUE)
predict(tmod1, interval = "confidence")

## Not run:
library("quanteda")
dfmat <- dfm(tokens(data_corpus_irishbudget2010))
(tmod2 <- textmodel_wordfish(dfmat, dir = c(6,5)))
(tmod3 <- textmodel_wordfish(dfmat, dir = c(6,5),
                             dispersion = "quasipoisson", dispersion_floor = 0))
(tmod4 <- textmodel_wordfish(dfmat, dir = c(6,5),
                             dispersion = "quasipoisson", dispersion_floor = .5))
plot(tmod3$phi, tmod4$phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5",
     xlim = c(0, 1.0), ylim = c(0, 1.0))
plot(tmod3$phi, tmod4$phi, xlab = "Min underdispersion = 0", ylab = "Min underdispersion = .5",
     xlim = c(0, 1.0), ylim = c(0, 1.0), type = "n")
underdispersedTerms <- sample(which(tmod3$phi < 1.0), 5)
which(featnames(dfmat) %in% names(topfeatures(dfmat, 20)))
text(tmod3$phi, tmod4$phi, tmod3$features,
     cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "grey90")
text(tmod3$phi['underdispersedTerms'], tmod4$phi['underdispersedTerms'],
     tmod3$features['underdispersedTerms'],
     cex = .8, xlim = c(0, 1.0), ylim = c(0, 1.0), col = "black")
if (requireNamespace("austin")) {
    tmod5 <- austin::wordfish(quanteda::as.wfm(dfmat), dir = c(6, 5))
    cor(tmod1$theta, tmod5$theta)
}
## End(Not run)
```

`textmodel_wordscores` *Wordscores text model*

Description

`textmodel_wordscores` implements Laver, Benoit and Garry's (2003) "Wordscores" method for scaling texts on a single dimension, given a set of anchoring or *reference* texts whose values are set through reference scores. This scale can be fitted in the linear space (as per LBG 2003) or in the logit space (as per Beauchamp 2012). Estimates of *virgin* or unknown texts are obtained using the `predict()` method to score documents from a fitted `textmodel_wordscores` object.

Usage

```
textmodel_wordscores(x, y, scale = c("linear", "logit"), smooth = 0)
```

Arguments

x	the dfm on which the model will be trained
y	vector of training scores associated with each document in x
scale	scale on which to score the words; "linear" for classic LBG linear posterior weighted word class differences, or "logit" for log posterior differences
smooth	a smoothing parameter for word counts; defaults to zero to match the LBG (2003) method. See Value below for additional information on the behaviour of this argument.

Details

The textmodel_wordscores() function and the associated predict() method are designed to function in the same manner as stats::predict.lm(). coef() can also be used to extract the word coefficients from the fitted textmodel_wordscores object, and summary() will print a nice summary of the fitted object.

Value

A fitted textmodel_wordscores object. This object will contain a copy of the input data, but in its original form without any smoothing applied. Calling predict.textmodel_wordscores() on this object without specifying a value for newdata, for instance, will predict on the unsmoothed object. This behaviour differs from versions of **quanteda** <= 1.2.

Author(s)

Kenneth Benoit

References

Laver, M., Benoit, K.R., & Garry, J. (2003). Estimating Policy Positions from Political Text using Words as Data. *American Political Science Review*, 97(2), 311–331.

Beauchamp, N. (2012). Using Text to Scale Legislatures with Uninformative Voting. New York University Mimeo.

Martin, L.W. & Vanberg, G. (2007). A Robust Transformation Procedure for Interpreting Political Text. *Political Analysis* 16(1), 93–100. doi:10.1093/pan/mpm010

See Also

predict.textmodel_wordscores() for methods of applying a fitted textmodel_wordscores model object to predict quantities from (other) documents.

Examples

```
(tmod <- textmodel_wordscores(quanteda::data_dfm_lbgexample, y = c(seq(-1.5, 1.5, .75), NA)))
summary(tmod)
coef(tmod)
predict(tmod)
predict(tmod, rescaling = "lbg")
predict(tmod, se.fit = TRUE, interval = "confidence", rescaling = "mv")
```

Index