Package ‘TextForecast’

January 21, 2025

Type Package

Title Regression Analysis and Forecasting Using Textual Data from a Time-Varying Dictionary

Version 0.1.3

Description Provides functionalities based on the paper ``Time Varying Dictionary and the Predictive Power of FED Minutes'' (Lima, 2018) <doi:10.2139/ssrn.3312483>. It selects the most predictive terms, that we call time-varying dictionary using supervised machine learning techniques as lasso and elastic net.

Depends R (>= 3.1.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2

Imports forecast, stats, tidyr, tidytext, tm, wordcloud, dplyr, plyr, udpipe, RColorBrewer, ggplot2, glmnet, pdftools, parallel, doParallel, pracma, forcats, Matrix

URL https://github.com/lucasgodeiro/TextForecast

BugReports https://github.com/lucasgodeiro/TextForecast/issues

Suggests knitr, rmarkdown, covr

VignetteBuilder knitr

NeedsCompilation no

Author Luiz Renato Lima [aut],
Lucas Godeiro [aut, cre]

Maintainer Lucas Godeiro <lucas.godeiro@hotmail.com>

Repository CRAN

Date/Publication 2022-04-25 08:50:02 UTC

Contents

`get_collocations` *get_collocations function*

Description

get_collocations function

Usage

```
get_collocations(
  corpus_dates,
  path_name,
  ntrms,
  ngrams_number,
  min_freq,
  language
)
```

Arguments

`corpus_dates`	a character vector indicating the subfolders where are located the texts.
`path_name`	the folders path where the subfolders with the dates are located.
`ntrms`	maximum numbers of collocations that will be filtered by tf-idf. We rank the collocations by tf-idf in a decreasing order. Then, after we select the words with the ntrms highest tf-idf.

`ngrams_number`	integer indicating the size of the collocations. Defaults to 2, indicating to compute bigrams. If set to 3, will find collocations of bigrams and trigrams.
`min_freq`	integer indicating the frequency of how many times a collocation should at least occur in the data in order to be returned.
`language`	the texts language. Default is english.

Value

a list containing a sparse matrix with the all collocations couting and another with a tf-idf filtered collocations counting according to the ntrms.

Examples

```
st_year=2017
end_year=2018
path_name=system.file("news",package="TextForecast")
#qt=paste0(sort(rep(seq(from=st_year,to=end_year,by=1),12)),
#c("m1","m2","m3","m4","m5","m6","m7","m8","m9","m10","m11","m12"))
#z_coll=get_collocations(corpus_dates=qt[1:23],path_name=path_name,
#ntrms=500,ngrams_number=3,min_freq=10)
#
path_name=system.file("news",package="TextForecast")
days=c("2019-30-01","2019-31-01")
z_coll=get_collocations(corpus_dates=days[1],path_name=path_name,
ntrms=500,ngrams_number=3,min_freq=1)
```

`get_terms` *get_terms function*

Description

get_terms function

Usage

```
get_terms(
  corpus_dates,
  ntrms_words,
  st,
  path.name,
  ntrms_collocation,
  ngrams_number,
  min_freq,
  language
)
```

Arguments

- `corpus_dates` a character vector indicating the subfolders where the texts are located.
- `ntrms_words` maximum numbers of words that will be filtered by tf-idf. We rank the word by tf-idf in a decreasing order. Then, we select the words with the ntrms highest tf-idf.
- `st` set 0 to stem the words and 1 otherwise.
- `path.name` the folders path where the subfolders with the dates are located.
- `ntrms_collocation` maximum numbers of collocations that will be filtered by tf-idf. We rank the collocations by tf-idf in a decreasing order. Then, after we select the words with the ntrms highest tf-idf.
- `ngrams_number` integer indicating the size of the collocations. Defaults to 2, indicating to compute bigrams. If set to 3, will find collocations of bigrams and trigrams.
- `min_freq` integer indicating the frequency of how many times a collocation should at least occur in the data in order to be returned.
- `language` the texts language. Default is english.

Value

a list containing a sparse matrix with the all collocations and words couting and another with a tf-idf filtered collocations and words counting according to the ntrms.

Examples

```
st_year=2017
end_year=2018
path_name=system.file("news",package="TextForecast")
#qt=paste0(sort(rep(seq(from=st_year,to=end_year,by=1),12)),
#c("m1","m2","m3","m4","m5","m6","m7","m8","m9","m10","m11","m12"))
#z_terms=get_terms(corpus_dates=qt[1:23],path.name=path_name,
#ntrms_words=500,ngrams_number=3,st=0,ntrms_collocation=500,min_freq=10)
#
path_name=system.file("news",package="TextForecast")
days=c("2019-30-01","2019-31-01")
z_terms=get_terms(corpus_dates=days[1],path.name=path_name,
ntrms_words=500,ngrams_number=3,st=0,ntrms_collocation=500,min_freq=1)
```

`get_words` *get_words function*

Description

get_words function

Usage

```
get_words(corpus_dates, ntrms, st, path_name, language)
```

Arguments

corpus_dates	A vector of characters indicating the subfolders where are located the texts.
ntrms	maximum numbers of words that will be filtered by tf-idf. We rank the word by tf-idf in a decreasing order. Then, we select the words with the ntrms highest tf-idf.
st	set 0 to stem the words and 1 otherwise.
path_name	the folders path where the subfolders with the dates are located.
language	The texts language.

Value

a list containing a sparse matrix with the all words couting and another with a td-idf filtered words counting according to the ntrms.

Examples

```
st_year=2017
end_year=2018
path_name=system.file("news",package="TextForecast")
#qt=paste0(sort(rep(seq(from=st_year,to=end_year,by=1),12)),
#c("m1","m2","m3","m4","m5","m6","m7","m8","m9","m10","m11","m12"))
#z_wrd=get_words(corpus_dates=qt[1:23],path_name=path_name,ntrms=500,st=0)
#
path_name=system.file("news",package="TextForecast")
days=c("2019-31-01","2019-31-01")
z_wrd=get_words(corpus_dates=days,path_name=path_name,ntrms=500,st=0)
```

---

hard_thresholding *hard thresholding*

Description

hard thresholding

Usage

```
hard_thresholding(x, w, y, p_value, newx)
```

Arguments

`x`	the input matrix x.
`w`	the optional input matrix w, that cannot be selected.
`y`	the response variable.
`p_value`	the threshold p-value.
`newx`	matrix that selection will applied. Useful for time series, when we need the observation at time t.

Value

the variables less than p-value.

Examples

```
data("stock_data")
data("optimal_factors")
y=as.matrix(stock_data[,2])
y=as.vector(y)
w=as.matrix(stock_data[,3])
pc=as.matrix(optimal_factors)
t=length(y)
news_factor <- hard_thresholding(w=w[1:(t-1),],x=pc[1:(t-1),],y=y[2:t],p_value = 0.01,newx = pc)
```

`news_data` *News Data*

Description

A simple `tibble` containing the term counting of the financial news from the wall street journal and the news york times from 1992:01 through 2018:11.

Usage

```
news_data
```

Format

A tibble with 1631 components.

dates The vector of dates.

X The terms counting.

optimal_alphas *Title optimal alphas function*

Description

Title optimal alphas function

Usage

```
optimal_alphas(x, w, y, grid_alphas, cont_folds, family)
```

Arguments

x	A matrix of variables to be selected by shrinkrage methods.
w	A matrix or vector of variables that cannot be selected(no shrinkrage).
y	response variable.
grid_alphas	a grid of alphas between 0 and 1.
cont_folds	Set TRUE for contiguous folds used in time depedent data.
family	The glmnet family.

Value

lambdas_opt a vector with the optimal alpha and lambda.

Examples

```
set.seed(1)
data("stock_data")
data("news_data")
y=as.matrix(stock_data[1:200,2])
w=as.matrix(stock_data[1:200,3])
data("news_data")
X=news_data[1:200,2:ncol(news_data)]
x=as.matrix(X)
grid_alphas=seq(by=0.25,to=1,from=0.5)
cont_folds=TRUE
t=length(y)
optimal_alphas=optimal_alphas(x[1:(t-1),],
w[1:(t-1),],y[2:t],grid_alphas,TRUE,"gaussian")
```

optimal_factors *Optimal Factors*

Description

A simple vector containing the Optimal factors select by optimal_number_factors function.

Usage

```
optimal_factors
```

Format

A vector with 1 component.

optimal fators x The vector of factor.

optimal_number_factors *optimal number of factors function*

Description

optimal number of factors function

Usage

```
optimal_number_factors(x, kmax)
```

Arguments

x	a matrix x.
kmax	the maximum number of factors

Value

a list with the optimal factors.

Examples

```
data("optimal_x")
optimal_factor <- optimal_number_factors(x=optimal_x,kmax=8)
```

`optimal_x` *Optimal x*

Description

A simple `matrix` containing the optimal words selected by Elastic Net from 1992:01 through 2018:11.

Usage

```
optimal_x
```

Format

A matrix with the most predictive terms.

x The matrix with 4 components.

`stock_data` *Stock Data*

Description

A simple `tibble` containing the S&P 500 return and the VIX volatility index from 1992:01 through 2018:11.

Usage

```
stock_data
```

Format

A tibble with 3 components.

dates The vector of dates.

sp_return The S&P 500 returns.

vix The volatility index.

text_forecast *Text Forecast function*

Description

Text Forecast function

Usage

```
text_forecast(x, y, h, intercept)
```

Arguments

x	the input matrix x.
y	the response variable
h	the forecast horizon
intercept	TRUE for include intercept in the forecast equation.

Value

The h step ahead forecast

Examples

```
set.seed(1)
data("stock_data")
data("news_data")
y=as.matrix(stock_data[,2])
w=as.matrix(stock_data[,3])
data("news_data")
data("optimal_factors")
pc=optimal_factors
z=cbind(w,pc)
fcsts=text_forecast(z,y,1,TRUE)
```

text_nowcast *text nowcast*

Description

text nowcast

Usage

```
text_nowcast(x, y, intercept)
```

Arguments

x	the input matrix x. It should have 1 observation more that y.
y	the response variable
intercept	TRUE for include intercept in the forecast equation.

Value

the nowcast h=0 for the variable y.

Examples

```
set.seed(1)
data("stock_data")
data("news_data")
y=as.matrix(stock_data[,2])
w=as.matrix(stock_data[,3])
data("news_data")
data("optimal_factors")
pc=optimal_factors
z=cbind(w,pc)
t=length(y)
ncsts=text_nowcast(z,y[1:(t-1)],TRUE)
```

---

tf_idf *tf-idf function*

Description

tf-idf function

Usage

```
tf_idf(x)
```

Arguments

x	a input matrix x of terms counting.

Value

a list with the terms tf-idf and the terms tf-idf in descending order.

Examples

```
data("news_data")
X=as.matrix(news_data[,2:ncol(news_data)])
tf_idf_terms = tf_idf(X)
```

top_terms *Top Terms Function*

Description

Top Terms Function

Usage

```
top_terms(
  x,
  w,
  y,
  alpha,
  lambda,
  k,
  wordcloud,
  max.words,
  scale,
  rot.per,
  family
)
```

Arguments

x	the input matrix of terms to be selected.
w	optional argument. the input matrix of structured data to not be selected.
y	the response variable
alpha	the glmnet alpha
lambda	the glmnet lambda
k	the k top terms
wordcloud	set TRUE to plot the wordcloud
max.words	the maximum number of words in the wordcloud
scale	the wordcloud size.
rot.per	wordcloud proportion 90 degree terms
family	glmnet family

Value

the top k terms and the corresponding wordcloud.

Examples

```
set.seed(1)
data("stock_data")
data("news_data")
y=as.matrix(stock_data[,2])
w=as.matrix(stock_data[,3])
data("news_data")
X=news_data[,2:ncol(news_data)]
x=as.matrix(X)
grid_alphas=seq(by=0.05,to=0.95,from=0.05)
cont_folds=TRUE
t=length(y)
optimal_alphas=optimal_alphas(x[1:(t-1),],w[1:(t-1),],
y[2:t],grid_alphas,TRUE,"gaussian")
top_trms<- top_terms(x[1:(t-1),],w[1:(t-1),],y[2:t],
optimal_alphas[[1]], optimal_alphas[[2]],10,TRUE,
10,c(2,0.3),.15,"gaussian")
```

`tv_dictionary` *tv dictionary function*

Description

tv dictionary function

Usage

```
tv_dictionary(x, w, y, alpha, lambda, newx, family)
```

Arguments

x	A matrix of variables to be selected by shrinkrage methods.
w	Optional Argument. A matrix of variables to be selected by shrinkrage methods.
y	the response variable.
alpha	the alpha required in glmnet.
lambda	the lambda required in glmnet.
newx	Matrix that selection will applied. Useful for time series, when we need the observation at time t.
family	the glmnet family.

Value

X_star: a list with the coefficients and a sparse matrix with the most predictive terms.

Examples

```
set.seed(1)
data("stock_data")
data("news_data")
y=as.matrix(stock_data[1:200,2])
w=as.matrix(stock_data[1:200,3])
data("news_data")
X=news_data[1:200,2:ncol(news_data)]
x=as.matrix(X)
grid_alphas=seq(by=0.5,to=1,from=0.5)
cont_folds=TRUE
t=length(y)
optimal_alphas=optimal_alphas(x[1:(t-1),],w[1:(t-1),],
y[2:t],grid_alphas,TRUE,"gaussian")
x_star=tv_dictionary(x=x[1:(t-1),],w=w[1:(t-1),],y=y[2:t],
alpha=optimal_alphas[1],lambda=optimal_alphas[2],newx=x,family="gaussian")
```

---

tv_sentiment_index *tv sentiment index function*

Description

tv sentiment index function

Usage

```
tv_sentiment_index(x, w, y, alpha, lambda, newx, family, k)
```

Arguments

x	A matrix of variables to be selected by shrinkrage methods.
w	Optional Argument. A matrix of variables to be selected by shrinkrage methods.
y	the response variable.
alpha	the alpha required in glmnet.
lambda	the lambda required in glmnet.
newx	Matrix that selection will be applied. Useful for time series, when we need the observation at time t.
family	the glmnet family.
k	the highest positive and negative coefficients to be used.

Value

The time-varying sentiment index. The index is based on the word/term counting and is computed using: tv_index=(pos-neg)/(pos+neg).

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
data("stock_data")
data("news_data")
y=as.matrix(stock_data[,2])
w=as.matrix(stock_data[,3])
data("news_data")
X=news_data[,2:ncol(news_data)]
x=as.matrix(X)
grid_alphas=0.05
cont_folds=TRUE
t=length(y)
optimal_alphas=optimal_alphas(x[1:(t-1),],w[1:(t-1),],
y[2:t],grid_alphas,TRUE,"gaussian")
tv_index <- tv_sentiment_index(x[1:(t-1),],w[1:(t-1),],y[2:t],
optimal_alphas[[1]],optimal_alphas[[2]],x,"gaussian",2)
```

---

tv_sentiment_index_all_coefs

TV sentiment index using all positive and negative coefficients.

Description

TV sentiment index using all positive and negative coefficients.

Usage

```
tv_sentiment_index_all_coefs(
  x,
  w,
  y,
  alpha,
  lambda,
  newx,
  family,
  scaled,
  k_mov_avg,
  type_mov_avg
)
```

Arguments

x	A matrix of variables to be selected by shrinkrage methods.
w	Optional Argument. A matrix of variables to be selected by shrinkrage methods.
y	the response variable.

alpha	the alpha required in glmnet.
lambda	the lambda required in glmnet.
newx	Matrix that selection will be applied. Useful for time series, when we need the observation at time t.
family	the glmnet family.
scaled	Set TRUE for scale and FALSE for no scale.
k_mov_avg	The moving average order.
type_mov_avg	The type of moving average. See movavg.

Value

A list with the net, postive and negative sentiment index. The net time-varying sentiment index. The index is based on the word/term counting and is computed using: tv_index=(pos-neg)/(pos+neg). The postive sentiment index is computed using: tv_index_pos=pos/(pos+neg) and the negative tv_index_neg=neg/(pos+neg).

Examples

```
suppressWarnings(RNGversion("3.5.0"))
set.seed(1)
data("stock_data")
data("news_data")
y=as.matrix(stock_data[,2])
w=as.matrix(stock_data[,3])
data("news_data")
X=news_data[,2:ncol(news_data)]
x=as.matrix(X)
grid_alphas=0.05
cont_folds=TRUE
t=length(y)
optimal_alphas=optimal_alphas(x=x[1:(t-1),],
                     y=y[2:t],grid_alphas=grid_alphas,cont_folds=TRUE,family="gaussian")
tv_idx=tv_sentiment_index_all_coefs(x=x[1:(t-1),],y=y[2:t],alpha = optimal_alphas[1],
                                  lambda = optimal_alphas[2],newx=x,
                                  scaled = TRUE,k_mov_avg = 4,type_mov_avg = "s")
```

Index