Package 'readme'

December 23, 2025

Title An Algorithm for Text Quantification

Version 2.0

Author 'Gary King <king@harvard.edu> [aut], Anton Strezhnev <astrezhn@law.upenn.edu> [aut, cre], Connor Jerzak <cjerzak@g.harvard.edu> [aut, cre]'

Description An R package for estimating category proportions in an unlabeled set of documents given a labeled set, by implementing the method described in Jerzak, King, and Strezhnev (2023, copy at <http://GaryKing.org/words>). This method is meant to improve on the ideas in Hopkins and King (2010, AJPS), which introduced a quantification algorithm that harnesses the Law of Total Expectation. We apply this law in a feature space we craft to minimize the error of the resulting estimate. Automatic differentiation, stochastic gradient descent, and batch re-normalization are used to carry out the optimization. Other pre-processing functions are available, as well as an interface to the earlier version of the algorithm for comparison. The package also provides users with the ability to extract the generated features for use in other tasks.

Depends R (>= 3.5.0)

License CC BY-NC-ND 4.0 + file LICENSE

Encoding UTF-8

LazyData true

Maintainer 'Connor Jerzak' <connor.jerzak@gmail.com>

Imports tensorflow, limSolve, FNN, data.table, tokenizers, reticulate

Suggests testthat (>= 3.0.0), text

Config/testthat/edition 3

RoxygenNote 7.3.3

NeedsCompilation no

Contents

readme-package 2
build_backend 4
cleanme 4
clinton 5
download_wordvecs 6
initialize_tensorflow 6
readme 7

readme0 10
tensorflow_available 10
undergrad 11

Index **13**

`readme-package` *A algorithm for quantification that harnesses the Law of Total Expectations in an optimal feature space*

Description

An R package for estimating category proportions in an unlabeled set of documents given a labeled set, by implementing the method described in Jerzak, King, and Strezhnev (2023, copy at http://GaryKing.org/words). This method is meant to improve on the ideas in Hopkins and King (2010), which introduced a quantification algorithm that harnesses the Law of Total Expectation. We apply this law in a feature space we craft minimizes the error of the resulting estimate. Automatic differentiation, stochastic gradient descent, and batch re-normalization are used to carry out the optimization. Other pre-processing functions are available, as well as an interface to the earlier version of the algorithm for comparison. The package also provides users with the ability to extract the generated features for other tasks.

The package provides the following main functions:

- `undergrad` takes as an input a word vector corpus (or pointer to such a corpus) and a vector housing cleaned text for cross-referencing with the vector corpus. It returns document-level summaries of each of the dimensions of the word vectors (`10th`, `50th`, and `90th` quantiles of each dimension within each document are calculated). Options also exist for generating a document-term matrix from the text. Useful for those wanting control over the linkup between documents and word vector corpus.
- `readme` takes as an input raw text (or optionally, the output from `undergrad`). It also takes as an input an indicator vector denoting which documents are labeled and a vector indicating category membership (`NA`s for unlabeled documents). The algorithm then generates an optimal projection for harnessing the Law of Total Expectation in calculating the estimated category proportions in the unlabeled set.
- `build_backend` creates a conda environment with TensorFlow installed. Run this once before using the package for the first time.
- `initialize_tensorflow` loads the TensorFlow backend. Call this at the start of each R session before using `readme`.
- `tensorflow_available` checks whether TensorFlow is available and properly configured.

Usage

For advice on usage, see **Examples**. Many users will just interface with the `readme` function, as this approach takes care of much of the pre-processing in an automatic fashion. Some users may want more control over the linkup between the word vector corpus and the raw text; in that case, combining `undergrad` with `readme` is a good option.

For bug reports or support, please contact <connor.jerzak@gmail.com>.

Authors

- Connor Jerzak, Anton Strezhnev, and Gary King.
- Maintainer: Connor Jerzak <cjerzak@gmail.com>

References

- Hopkins, Daniel, and King, Gary (2010), A Method of Automated Nonparametric Content Analysis for Social Science, *American Journal of Political Science*, Vol. 54, No. 1, January 2010, p. 229-247.
- Jerzak, Connor, King, Gary, and Strezhnev, Anton. (2023), An Improved Method of Automated Nonparametric Content Analysis for Social Science, *Political Analysis*, Vol. 31, No. 1, p. 42-58. https://doi.org/10.1017/pan.2021.36

Examples

```
## Not run:
#set seed
set.seed(1)

#Generate synthetic 25-d word vector corpus.
my_wordVecs <- matrix(rnorm(11*25), ncol = 25)
row.names(my_wordVecs) <- c("the","true", "thine", "stars", "are",
                            "fire", ".", "to", "own", "self", "be")

#Generate 100 ``documents'' of 5-10 words each.
my_documentText <- replicate(100,
                             paste(sample(row.names(my_wordVecs),
                                          sample(5:10, 1),
                                          replace = TRUE),
                                   collapse = " ") )

#Assign labeled/unlabeled sets. The first 50 will be labeled; the rest unlabeled.
my_labeledIndicator <- rep(1, times = 100)
my_labeledIndicator[51:100] <- 0

#Assign category membership randomly
my_categoryVec <- sample(c("C1", "C2", "C3", "C4"), 100, replace = TRUE)
true_unlabeled_pd <- prop.table(table(my_categoryVec[my_labeledIndicator==0]))
my_categoryVec[my_labeledIndicator == 0] <- NA

#Get word vector summaries
my_dfm <- undergrad(documentText = my_documentText, wordVecs = my_wordVecs)

#perform estimation
readme_results <- readme(dfm = my_dfm,
                         labeledIndicator = my_labeledIndicator,
                         categoryVec = my_categoryVec,
                         nBoot = 2)
print(readme_results$point_readme)

## End(Not run)
```

build_backend *Build TensorFlow Backend*

Description

Creates a conda environment with TensorFlow installed for use with readme. This function sets up the Python backend required for the readme() function.

Usage

```
build_backend(
  conda_env = "readme_env",
  conda = "auto",
  python_version = "3.11",
  tensorflow_version = NULL
)
```

Arguments

conda_env	Name of the conda environment to create (default: "readme_env")
conda	Path to conda executable (default: "auto" to auto-detect)
python_version	Python version to install (default: "3.11")
tensorflow_version	TensorFlow version to install (default: NULL for latest)

Value

Invisibly returns the conda environment name

Examples

```
## Not run:
# Build the default environment
build_backend()

# Build with a custom environment name
build_backend(conda_env = "my_readme_env")

## End(Not run)
```

cleanme *cleanme*

Description

Standard preprocessing code for ASCII texts. Removes HTML tags, URLs, linebreaks. Converts standard emoticons to tokens. Removes non-informative punctuation.

Usage

```
cleanme(my_text, finalEncoding = "ASCII")
```

Arguments

`my_text`	Vector of character strings containing the raw document texts.
`finalEncoding`	A character string indicating the desired encoding for the text vector (default = "ASCII")

Value

A vector of character strings with the processed texts, each token is separated by a space.

`clinton` *Clinton Email Dataset*

Description

A dataset of email documents from the Clinton email corpus, with category labels for a subset of documents. This dataset is used for demonstrating text quantification methods.

Usage

```
clinton
```

Format

A data frame with columns:

ROWID Numeric. Row identifier.

TEXT Character. The raw email text content.

TRAININGSET Numeric. Indicator for whether document is labeled (1) or unlabeled (0). Labeled documents are in rows 1-427, unlabeled documents start at row 428.

TRUTH Character. The true category label for each document.

Source

Derived from the Clinton email corpus for text quantification research.

Examples

```
data(clinton)
head(clinton)

# See structure
str(clinton)

# Count labeled vs unlabeled
table(clinton$TRAININGSET)
```

download_wordvecs *download_wordvecs*

Description

Downloads default word vector dictionary to 'readme' install directory.

Usage

```
download_wordvecs(
  url = "http://gking-projects.iq.harvard.edu/files/glove.6B.200d.zip",
  targetDir = NULL
)
```

Arguments

url	URL of word vector file. Defaults to the pre-trained GloVe Wikipedia 200-dimensional vectors.
targetDir	Target directory to download files. If NULL uses readme installation directory.

Value

Path to downloaded word vector dictionary

initialize_tensorflow *Initialize TensorFlow Backend*

Description

Loads the conda environment and initializes TensorFlow for use with readme. This function should be called before using the readme() function.

Usage

```
initialize_tensorflow(
  conda_env = "readme_env",
  conda_env_required = TRUE,
  verbose = TRUE
)
```

Arguments

conda_env	Name of the conda environment (default: "readme_env")
conda_env_required	Whether the conda environment is required (default: TRUE). If TRUE and the environment doesn't exist, an error is raised.
verbose	Whether to print status messages (default: TRUE)

Value

Invisibly returns the TensorFlow version string

Examples

```
## Not run:
# Initialize the default environment
initialize_tensorflow()

# Initialize a custom environment
initialize_tensorflow(conda_env = "my_readme_env")

## End(Not run)
```

---

readme *readme*

Description

Implements the quantification algorithm described in Jerzak, King, and Strezhnev (2018) which is meant to improve on the ideas in Hopkins and King (2010). Employs the Law of Total Expectation in a feature space that is tailoed to minimize the error of the resulting estimate. Automatic differentiation, stochastic gradient descent, and knn_adaptbatch re-normalization are used to carry out the optimization. Takes an inputs (a.) a vector holding the raw documents (1 entry = 1 document), (b.) a vector indicating category membership (with NAs for the unlabeled documents), and (c.) a vector indicating whether the labeled or unlabeled status of each document. Other options exist for users wanting more control over the pre-processing protocol (see undergrad and the dfm parameter).

Usage

```
readme(
  dfm,
  labeledIndicator,
  categoryVec,
  nBoot = 15,
  sgdIters = 500,
  numProjections = NULL,
  batchSizePerCat = 10,
  kMatch = 3,
  batchSizePerCat_match = 20,
  minMatch = 8,
  nbootMatch = 50,
  justTransform = FALSE,
  verbose = FALSE,
  diagnostics = FALSE,
  nCores = 1L,
  sigma_ep = 0,
  nCores_OnJob = 1L,
  regraph = FALSE,
  conda_env = NULL,
  otherOption = NULL,
```

```
  wt_catDistinctiveness = NULL,
  wt_featDistinctiveness = NULL,
  tensorflowSeed = NULL
)
```

Arguments

dfm	'document-feature matrix'. A data frame where each row represents a document and each column a unique feature.
labeledIndicator	An indicator vector where each entry corresponds to a row in dfm. 1 represents document membership in the labeled class. 0 represents document membership in the unlabeled class.
categoryVec	An factor vector where each entry corresponds to the document category. The entires of this vector should correspond with the rows of dtm. If wordVecs_corpus, wordVecs_corpusPointer, and dfm are all NULL, readme will download and use the GloVe 50-dimensional embeddings trained on Wikipedia.
nBoot	A scalar indicating the number of times the estimation procedure will be re-run (useful for reducing the variance of the final output).
sgdIters	How many stochastic gradient descent iterations should be used? Input should be a positive number.
numProjections	How many projections should be calculated? Input should be a positive number. Minimum number of projections = number of categories + 2.
batchSizePerCat	What should the batch size per category be in the sgd optimization and knn matching?
kMatch	What should k be in the k-nearest neighbor matching? Input should be a positive number.
batchSizePerCat_match	What should the batch size per category be in the bagged knn matching?
minMatch	Minimum number of labeled documents per category to include via KNN matching. Default is 8.
nbootMatch	How many bootstrap samples should we aggregiate when doing the knn matching?
justTransform	A Boolean indicating whether the user wants to extract the quanficiation-optimized features only.
verbose	Should progress updates be given? Input should be a Boolean.
diagnostics	Should diagnostics be returned? Boolean, default is FALSE.
nCores	How many CPU cores are available? Default is 1.
sigma_ep	Small constant added to variance for numerical stability. Default is 0.
nCores_OnJob	How many CPU cores should we make available to tensorflow? Default is 1.
regraph	Should the TensorFlow graph be rebuilt? Boolean, default is FALSE.
conda_env	Name of the conda environment to use for TensorFlow. Default is NULL (uses default environment).
otherOption	Reserved for future use. Default is NULL.
wt_catDistinctiveness	Weight for category distinctiveness in the loss function. Default is NULL (uses default weighting).

`wt_featDistinctiveness`
: Weight for feature distinctiveness in the loss function. Default is NULL (uses default weighting).

`tensorflowSeed`
: Random seed for TensorFlow reproducibility. Default is NULL.

Value

A list consiting of

- estimated category proportions in the unlabeled set (`point_readme`);
- the transformed dfm optimized for quantification (`transformed_dfm`);
- (optional) a list of diagnostics (`diagnostics`);

References

- Hopkins, Daniel, and King, Gary (2010), A Method of Automated Nonparametric Content Analysis for Social Science, *American Journal of Political Science*, Vol. 54, No. 1, January 2010, p. 229-247.
- Jerzak, Connor, King, Gary, and Strezhnev, Anton. (2023), An Improved Method of Automated Nonparametric Content Analysis for Social Science, *Political Analysis*, Vol. 31, No. 1, p. 42-58. https://doi.org/10.1017/pan.2021.36

Examples

```
## Not run:
#set seed
set.seed(1)

#Generate synthetic 25-d word vector corpus.
my_wordVecs <- matrix(rnorm(11*25), ncol = 25)
row.names(my_wordVecs) <- c("the","true", "thine", "stars", "are",
                            "fire", ".", "to", "own", "self", "be")

#Generate 100 ``documents'' of 5-10 words each.
my_documentText <- replicate(100,
                             paste(sample(row.names(my_wordVecs),
                                          sample(5:10, 1),
                                          replace = TRUE),
                                   collapse = " ") )

#Assign labeled/unlabeled sets. The first 50 will be labeled; the rest unlabeled.
my_labeledIndicator <- rep(1, times = 100)
my_labeledIndicator[51:100] <- 0

#Assign category membership randomly
my_categoryVec <- sample(c("C1", "C2", "C3", "C4"), 100, replace = TRUE)
true_unlabeled_pd <- prop.table(table(my_categoryVec[my_labeledIndicator==0]))
my_categoryVec[my_labeledIndicator == 0] <- NA

#Get word vector summaries
my_dfm <- undergrad(documentText = my_documentText, wordVecs = my_wordVecs)

#perform estimation
readme_results <- readme(dfm = my_dfm,
                         labeledIndicator = my_labeledIndicator,
```

```
                          categoryVec = my_categoryVec,
                          nBoot = 2, sgdIters = 500)
print(readme_results$point_readme)

## End(Not run)
```

`readme0` *readme0 Internal*

Description

readme0 Internal

Usage

```
readme0(
  dtm,
  labeledIndicator,
  categoryVec,
  features = 15,
  nboot = 10,
  probWt = 1
)
```

`tensorflow_available` *Check TensorFlow Availability*

Description

Checks whether TensorFlow is available in the current Python environment.

Usage

```
tensorflow_available()
```

Value

Logical indicating whether TensorFlow is available

Examples

```
## Not run:
if (tensorflow_available()) {
  result <- readme(dfm, labeledIndicator, categoryVec)
}

## End(Not run)
```

undergrad *undergrad*

Description

Preprocessing for readme function - creates a document-feature matrix (saved as a data frame in output) to be passed to readme. Users can either input word-specific vectors using the wordVecs_corpus or wordVecs_corpusPointer parameters. Primarily intended for users wanting control over the pre-processing protocol. numericization_method controls whether word vector summaries (the default, numericization_method = "vector_based") or transformer-based document features (if using "transformer_based").

Usage

```
undergrad(
  documentText,
  wordVecs = NULL,
  word_quantiles = c(0.1, 0.5, 0.9),
  reattempt = TRUE,
  reattempt_regex = list(c("\\#", ""), c("#\\S+", "<hashtag>"), c("[[:punct:]]+",
    ""), c("ing\\b", ""), c("s\\b", ""), c("ed\\b", ""), c("ies\\b", "y")),
  unique_terms = TRUE,
  verbose = TRUE,
  numericization_method = "vector_summaries",
  textEmbed_control = list(tokenizer_parallelism = TRUE, model =
    "bert-base-multilingual-uncased", layers = -2L)
)
```

Arguments

documentText	A vector in which each entry corresponds to a "clean" document. Note that the function will take as a "word" all whitespace-separated elements in each vector entry. For example, "star." would have to have an exact analogue in the vector corpus, otherwise it will be dropped in the calculations.
wordVecs	A matrix where each row denotes a word and each column a word vector. Words should be stored as the rownames of the matrix.
word_quantiles	A numeric vector denoting the quantiles (0-1) used to summarize each word vector dimension. Defaults to 0.10th, 0.50th and 0.90th quantiles.
reattempt	If TRUE, attempts to match terms missing from the wordVec corpus with alternate representations.
reattempt_regex	A list of character vectors containing regular expression pairs to be used for generating alternate representations of words to attempt to match with the wordVec corpus when terms initially cannot be matched. Order matters.
unique_terms	If TRUE, removes duplicate terms from each document - each document is represented only by the presence or absence of a term.
verbose	If TRUE, prints updates as function runs

numericization_method
: Determines whether word vector summaries are used for documents (faster, specified by 'numericization_method = "vector_summaries"'), or whether transformer-based document features are used (slower, specified by 'numericization_method = "transformer_based"')

textEmbed_control
: A list with elements denoting the parameters passed to 'text::textEmbed' if 'numericization_method = "transformer_based"'. Default is 'list(tokenizer_parallelism = TRUE, model = "bert-base-multilingual-uncased", layers = -2L)')

Value

A data.frame consisting of the word_quantiles quantiles of the word vectors by document. Each row corresonds to a document, and the columns to a particular summary of a particular word vector dimension.

Examples

```
#set seed
set.seed(1)

#Generate synthetic word vector corpus.
my_wordVecs <- matrix(rnorm(11*50), ncol = 50)
row.names(my_wordVecs) <- c("the","true", "thine", "stars", "are" ,
                            "fire", ".", "to", "own", "self", "be")

#Setup ``documents''
my_documentText <- c(
"the stars are fire .", #document 1
"to thine own self be true ", #document 2
"true stars be true ." #document 3
)

#Get document-level word vector summaries.
my_dfm <- undergrad(documentText = my_documentText, wordVecs = my_wordVecs)
print( my_dfm )
```

Index

∗ **datasets**
clinton, 5
∗ **internal**
readme-package, 2
readme0, 10

build_backend, 4

cleanme, 4
clinton, 5

download_wordvecs, 6

initialize_tensorflow, 6

readme, 7
readme-package, 2
readme0, 10

tensorflow_available, 10

undergrad, 11