Package: rwig (via r-universe)

May 22, 2026

Type Package

Title Wasserstein Index Generation (WIG) Model

Version 0.1.0

Description Efficient implementation of several Optimal Transport algorithms in Fangzhou Xie (2025) <doi:10.48550/arXiv.2504.08722> and the Wasserstein Index Generation (WIG) model in Fangzhou Xie (2020) <doi:10.1016/j.econlet.2019.108874>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

URL https://github.com/fangzhou-xie/rwig, https://fangzhou-xie.github.io/rwig/

BugReports https://github.com/fangzhou-xie/rwig/issues

LinkingTo Rcpp (>= 1.0.8), RcppArmadillo (>= 0.12.8.4.0)

Imports utils, stats, rlang, cli, lubridate, Rcpp, RhpcBLASctl, word2vec, tokenizers, stopwords

Roxygen list(markdown = TRUE)

SystemRequirements autoconf, CUDA (> 13.0, only for Linux)

Suggests knitr, rmarkdown, tinytest

Depends R (>= 4.2)

VignetteBuilder knitr

Config/pak/sysreqs nvidia-cuda-dev libicu-dev

Repository https://fangzhou-xie.r-universe.dev

Date/Publication 2026-04-21 19:42:26 UTC

RemoteUrl https://github.com/fangzhou-xie/rwig

RemoteRef HEAD

RemoteSha c9cdecf4cd1c673011202e97a4baed066f3637b8

Contents

`barycenter` *Barycenter algorithm*

Description

Barycenter algorithm to solve for entropy-regularized Optimal Transport Barycenter problems. For a more detailed explaination, please refer to `vignette("barycenter")`.

Usage

```
barycenter(
  A,
  C,
  w,
  b_ext = NULL,
 barycenter_control = list(reg = 0.1, with_grad = FALSE, use_cuda = TRUE, n_threads = 0,
  method = "auto", threshold = 0.1, max_iter = 1000, zero_tol = 1e-06, verbose = 0)
)
```

Arguments

`A`	numeric matrix, source discrete densities (M * S)
`C`	numeric matrix, cost matrix between source and target (M * N)
`w`	numeric vector, weight vector (S)
`b_ext`	numeric vector, only used to calculate quadratic loss against the computed barycenter (default = NULL)
`barycenter_control`	list, control parameters for the computation • reg: double, regularization parameter (default = .1) • with_grad: bool, whether to calculate the gradient w.r.t. A • n_threads: int, number of threads (only used for `method = "log"`, ignored by `method = "parallel"`, default = 0) • threshold: double, threshold value below which "auto" method will default to `method = "log"` for stablized computation in log-domain (default = .1)

- max_iter: int, maximum iteration of `barycenter` algorithm (default = 1000)
- zero_tol: double, determine covergence (default = 1e-6)
- verbose: int, print out debug info for the algorithm for every `verbose` iteration (default to 0, i.e. not printing anything)

Details

This is the general function to solve OT Barycenter problem, and it will use either parallel (`method = "parallel"`) or log-stablized Barycenter algorithm (`method = "log"`) for solving the problem.

Value

list of results

- b: numeric vector, computed barycenter
- grad_A: gradient w.r.t. A (only with `with_grad = TRUE`)
- grad_w: gradient w.r.t. w (only with `with_grad = TRUE`)
- loss: double, quadratic loss between `b` and `b_ext` (only with `with_grad = TRUE`)
- U, V: scaling variables for the Sinkhorn algorithm (only with `method = "parallel"`)
- F, G: scaling variables for the Sinkhorn algorithm (only with `method = "log"`)
- iter: iterations of the algorithm
- err: condition for convergence
- return_status: 0 (convergence), 1 (max iteration reached), 2 (other)

References

Peyré, G., & Cuturi, M. (2019). Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning*, 11(5–6), 355–607. https://doi.org/10.1561/2200000073

Xie, F. (2025). Deriving the Gradients of Some Popular Optimal Transport Algorithms (No. arXiv:2504.08722). *arXiv*. https://doi.org/10.48550/arXiv.2504.08722

See Also

`vignette("gradient")`, `vignette("threading")`

Examples

```
A <- rbind(c(.3, .2), c(.2, .1), c(.1, .2), c(.1, .1), c(.3, .4))
C <- rbind(
  c(.1, .2, .3, .4, .5),
  c(.2, .3, .4, .3, .2),
  c(.4, .3, .2, .1, .2),
  c(.3, .2, .1, .2, .5),
  c(.5, .5, .4, .0, .2)
)
w <- c(.4, .6)
b <- c(.2, .2, .2, .2, .2)
reg <- .1
```

```
# simple barycenter example
sol <- barycenter(A, C, w, barycenter_control = list(reg = reg))

# you can also supply arguments to control the computation
# for example, including the loss and gradient w.r.t. `A`
sol <- barycenter(A, C, w, b, barycenter_control = list(reg = reg, with_grad = TRUE))
```

---

check_cuda *Check if CUDA is available*

Description

Check if CUDA is available for GPU computations.

Usage

```
check_cuda()
```

Value

logical, TRUE if CUDA is available, FALSE otherwise

Examples

```
if (check_cuda()) {
  cat("CUDA is available for GPU computations.\n")
} else {
  cat("CUDA is not available.\n")
}
```

---

headlines *NYT headlines by economic policy uncertainty*

Description

NYT headlines by economic policy uncertainty

Usage

```
headlines
```

Format

headlines:

A data frame with 18,507 rows and 2 columns:

headline Headline of the NYT News

ref_date Date of the News

Source

www.nytimes.com

References

Xie, F. (2020). Wasserstein index generation model: Automatic generation of time-series index with application to economic policy uncertainty. *Economics Letters*, 186, 108874. https://doi.org/10.1016/j.econlet.2019.108874

`sinkhorn` *Sinkhorn algorithm*

Description

Sinkhorn algorithm to solve entropy-regularized Optimal Transport problems. For a more detailed explaination, please refer to `vignette("sinkhorn")`.

Usage

```
sinkhorn(
  a,
  b,
  C,
 sinkhorn_control = list(reg = 0.1, with_grad = FALSE, use_cuda = TRUE, n_threads = 0,
  method = "auto", threshold = 0.1, max_iter = 1000L, zero_tol = 1e-06, verbose = 0L)
)
```

Arguments

`a`	numeric vector, source discrete density (probability vector)
`b`	numeric vector, target discrete density (probability vector)
`C`	numeric matrix, cost matrix between source and target
`sinkhorn_control`	list, control parameters for the computation • reg double, regularization parameter (default = .1) • with_grad: bool, whether to calculate the gradient w.r.t. a • n_threads: int, number of threads (only used for `method = "log"`, ignored by the `method = "vanilla"`, default = 0)

- method: character, which method to use: "auto", "vanilla", "log" "auto" with try to calculate minimum value of the Gibbs kernel K and switch to `method = "log"` if the minimum value is less than `threshold` (default = "auto")
- threshold: double, threshold value below which "auto" method will default to `method = "log"` for stablized computation in log-domain (default = .1)
- max_iter: int, maximum iteration of `sinkhorn` algorithm (default = 1000)
- zero_tol: double, determine covergence (default = 1e-6)
- verbose: int, print out debug info for the algorithm for every `verbose` iteration (default to 0, i.e. not printing anything)

Details

This is the general function to solve the OT problem, and it will use either vanilla (`method = "vanilla"`) or log-stabilized Sinkhorn algorithm (`method = "log"`) for solving the problem.

Value

list of results

- P: optimal coupling matrix
- grad_a: gradient of loss w.r.t. `a` (only with `with_grad = TRUE`)
- u, v: scaling vectors
- loss: regularized loss
- iter: iterations of the algorithm
- err: condition for convergence
- return_status: 0 (convergence), 1 (max iteration reached), 2 (other)

References

Peyré, G., & Cuturi, M. (2019). Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning*, 11(5–6), 355–607. https://doi.org/10.1561/2200000073

Xie, F. (2025). Deriving the Gradients of Some Popular Optimal Transport Algorithms (No. arXiv:2504.08722). *arXiv*. https://doi.org/10.48550/arXiv.2504.08722

See Also

`vignette("gradient")`, `vignette("threading")`

Examples

```
# simple sinkhorn example
a <- c(.3, .4, .1, .1, .1)
b <- c(.4, .5, .1)
C <- rbind(
  c(.1, .2, .3),
  c(.2, .3, .4),
  c(.4, .3, .2),
```

```
  c(.3, .2, .1),
  c(.5, .5, .4)
)
reg <- .1
sol <- sinkhorn(a, b, C, sinkhorn_control = list(reg = reg, verbose = 0))

# you can also supply arguments to control the computation
# for example, calculate the gradient w.r.t. a
sol <- sinkhorn(a, b, C,
sinkhorn_control = list(reg = reg, with_grad = TRUE, verbose = 0))
```

`tsvd` *Truncated SVD*

Description

Truncated Singular Value Decomposition algorithm

Usage

```
tsvd(M, k = 1, flip_sign = c("auto", "sklearn", "none"))
```

Arguments

`M`	matrix, data to be analyzed
`k`	int, number of columns/features to be kept
`flip_sign`	character, one of the following: "auto", "sklearn", "none"

Details

Compute the truncated SVD for dimension reduction. Note that SVD suffers from "sign indeterminacy," which means that the signs of the output vectors could depend on the algorithm and random state. Two variants of "sign flipping methods" are implemented here, one following the sklearn implementation on Truncated SVD, another by Bro et al. (2008).

Value

matrix after dimension reduction

References

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

Bro, R., Acar, E., & Kolda, T. G. (2008). Resolving the sign ambiguity in the singular value decomposition. Journal of Chemometrics, 22(2), 135–140. https://doi.org/10.1002/cem.1122

See Also

Truncated SVD (sklearn) `vignette("tsvd")`

Examples

```
A <- rbind(c(1,3), c(2,4))
tsvd(A)
```

---

wdl *Wasserstein Dictionary Learning model*

Description

Wasserstein Dictionary Learning (WDL) model for topic modeling

Usage

```
wdl(docs, ...)

## S3 method for class 'character'
wdl(docs, specs = wdl_specs(), verbose = TRUE, ...)

## S3 method for class 'wdl'
print(x, topic = 0, token_per_topic = 10, ...)

## S3 method for class 'wdl'
summary(object, topic = 1, token_per_topic = 10, ...)
```

Arguments

docs	character vector, sentences to be analyzed
...	only for compatibility
specs	list, model specification for the WDL see wdl_specs for reference
verbose	bool, whether to print useful info
x	WDL model
topic	int, number of topic to be printed
token_per_topic	int, number of tokens to be printed
object	WDL model

Details

This is the re-implementation of WDL model from ground up, and it calls the barycenter under the hood (to be precise directly calling the underlying C++ routine for barycenter)

Value

topics and weights computed from the WDL given the input data

References

Peyré, G., & Cuturi, M. (2019). Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning*, 11(5–6), 355–607. https://doi.org/10.1561/2200000073

Schmitz, M. A., Heitz, M., Bonneel, N., Ngolè, F., Coeurjolly, D., Cuturi, M., Peyré, G., & Starck, J.-L. (2018). Wasserstein dictionary learning: Optimal transport-based unsupervised nonlinear dictionary learning. *SIAM Journal on Imaging Sciences*, 11(1), 643–678. https://doi.org/10.1137/17M1140431

Xie, F. (2025). Deriving the Gradients of Some Popular Optimal Transport Algorithms (No. arXiv:2504.08722). *arXiv*. https://doi.org/10.48550/arXiv.2504.08722

See Also

```
vignette("wdl-model")
```

Examples

```
# simple WDL example
sentences <- c("this is a sentence", "this is another one")
wdl_fit <- wdl(
   sentences,
  specs = wdl_specs(wdl_control = list(num_topics = 2),word2vec_control = list(min_count = 1)),
   verbose = TRUE)
```

---

wdl_specs *Model Specs for WDL and WIG models*

Description

Control the parameters of WDL and WIG models

Usage

```
wdl_specs(
  wdl_control = list(num_topics = 4, batch_size = 64, epochs = 2, shuffle = TRUE,
    rng_seed = 42),
  tokenizer_control = list(stopwords = stopwords::stopwords()),
  word2vec_control = list(type = "cbow", dim = 10, min_count = 3),
 barycenter_control = list(reg = 0.1, with_grad = TRUE, use_cuda = TRUE, n_threads = 0,
    method = "auto", threshold = 0.1, max_iter = 20, zero_tol = 1e-06),
 optimizer_control = list(optimizer = "adamw", lr = 0.005, decay = 0.01, beta1 = 0.9,
    beta2 = 0.999, eps = 1e-08)
)

wig_specs(
 wig_control = list(group_unit = "month", svd_method = "topics", standardize = TRUE),
  wdl_control = list(num_topics = 4, batch_size = 64, epochs = 2, shuffle = TRUE,
    rng_seed = 42),
```

```
  tokenizer_control = list(stopwords = stopwords::stopwords()),
  word2vec_control = list(type = "cbow", dim = 10, min_count = 1),
 barycenter_control = list(reg = 0.1, with_grad = TRUE, use_cuda = TRUE, method =
    "auto", threshold = 0.1, max_iter = 20, zero_tol = 1e-06),
optimizer_control = list(optimizer = "adamw", lr = 0.005, decay = 0.01, beta1 = 0.9,
    beta2 = 0.999, eps = 1e-08)
)
```

Arguments

wdl_control	list, parameters for WDL
tokenizer_control	list, parameters for tokenizers::tokenize_words()
word2vec_control	list, parameters for word2vec::word2vec()
barycenter_control	list, parameters for barycenter()
optimizer_control	list, parameters for the optimizer (SGD, Adam, AdamW)
wig_control	list, parameters for WIG model

Details

See vignette("specs") for details on the parameters.

Value

list of the control lists

References

Peyré, G., & Cuturi, M. (2019). Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning*, 11(5–6), 355–607. https://doi.org/10.1561/2200000073

Schmitz, M. A., Heitz, M., Bonneel, N., Ngolè, F., Coeurjolly, D., Cuturi, M., Peyré, G., & Starck, J.-L. (2018). Wasserstein dictionary learning: Optimal transport-based unsupervised nonlinear dictionary learning. *SIAM Journal on Imaging Sciences*, 11(1), 643–678. https://doi.org/10.1137/17M1140431

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*.

Loshchilov, I., & Hutter, F. (2019). Decoupled Weight Decay Regularization (No. arXiv:1711.05101). *arXiv*. https://doi.org/10.48550/arXiv.1711.05101

Xie, F. (2020). Wasserstein index generation model: Automatic generation of time-series index with application to economic policy uncertainty. *Economics Letters*, 186, 108874. https://doi.org/10.1016/j.econlet.2019.108874

Xie, F. (2025). Deriving the Gradients of Some Popular Optimal Transport Algorithms (No. arXiv:2504.08722). *arXiv*. https://doi.org/10.48550/arXiv.2504.08722

See Also

wig_specs(), barycenter(), word2vec::word2vec(), tokenizers::tokenize_words(), vignette("specs")

wig *Wasserstein Index Generation model*

Description

Wasserstein Index Generation (WIG) model for time-series sentiment index autogeneration

Usage

```
wig(.data, date_col, docs_col, ...)

## S3 method for class 'data.frame'
wig(.data, date_col, docs_col, specs = wig_specs(), verbose = TRUE, ...)

## S3 method for class 'wig'
print(x, topic = 1, token_per_topic = 10, ...)

## S3 method for class 'wig'
summary(object, topic = 1, token_per_topic = 10, ...)
```

Arguments

.data	a dataframe containing the dates/datetimes and documents
date_col	name of the column for dates/datetimes
docs_col	name of the column for the texts/documents
...	only for compatibility
specs	list, model specification for WIG see wig_specs for reference
verbose	bool, whether to print useful info
x	WIG model
topic	int, number of topic to be printed
token_per_topic	int, number of tokens to be printed
object	WIG model

Details

This is the re-implementation of WIG model from scratch in R.

Value

"wig" class, i.e. list of the index and the WDL model

References

Xie, F. (2020). Wasserstein index generation model: Automatic generation of time-series index with application to economic policy uncertainty. *Economics Letters*, 186, 108874. https://doi.org/10.1016/j.econlet.2019.108874

See Also

```
vignette("wdl-model")
```

Examples

```
# create a small dataset
wigdf <- data.frame(
  ref_date = as.Date(c("2012-01-01", "2012-02-01")),
  docs = c("this is a sentence", "this is another sentence"))

wigfit <- wig(wigdf, ref_date, docs,
 specs = wig_specs(wdl_control = list(num_topics = 2),word2vec_control = list(min_count = 1)),
  verbose = FALSE)
```

Index