Package 'T4transport'

October 9, 2020

Type Package

Title Tools for Computational Optimal Transport

Version 0.1.0

Description Transport theory has seen much success in many fields of statistics and machine learning. We provide a variety of algorithms to compute Wasserstein distance, barycenter, and others. See Peyré and Cuturi (2019) <doi:10.1561/2200000073> for the general exposition to the study of computational optimal transport.

License MIT + file LICENSE

Imports Rcpp (>= 1.0.5), Rdpack, lpSolve, stats, utils

LinkingTo Rcpp, RcppArmadillo

Encoding UTF-8

RoxygenNote 7.1.1

RdMacros Rdpack

Suggests ggplot2

Depends R (>= 2.10)

NeedsCompilation yes

Author Kisung You [aut, cre] (<https://orcid.org/0000-0002-8584-459X>)

Maintainer Kisung You <kyoustat@gmail.com>

Repository CRAN

Date/Publication 2020-10-09 08:10:03 UTC

R topics documented:

barysinkhorn14 *Wasserstein Barycenter via Entropic Regularization by Cuturi & Doucet (2014)*

Description

Given K empirical measures $\mu_1, \mu_2, \ldots, \mu_K$, wasserstein barycenter μ^* is the solution to the following problem

$$\sum_{k=1}^{K} \pi_k \mathcal{W}_p^p(\mu, \mu_k)$$

where π_k's are relative weights of empirical measures. Here we assume either (1) support atoms in Euclidean space are given, or (2) all pairwise distances between atoms of the fixed support and empirical measures are given. Here the subgradient is approximated using the entropic regularization.

Usage

```
barysinkhorn14(
  support,
  measures,
  marginals = NULL,
  weights = NULL,
  lambda = 0.1,
  p = 2,
  ...
)

barysinkhorn14D(
  distances,
  marginals = NULL,
  weights = NULL,
  lambda = 0.1,
  p = 2,
  ...
)
```

Arguments

support	an $(N \times P)$ matrix of rows being atoms for the fixed support.
measures	a length-K list where each element is an $(N_k \times P)$ matrix of atoms.
marginals	marginal distribution for empirical measures; if NULL (default), uniform weights are set for all measures. Otherwise, it should be a length-K list where each element is a length-N_i vector of nonnegative weights that sum to 1.
weights	weights for each individual measure; if NULL (default), each measure is considered equally. Otherwise, it should be a length-K vector.
lambda	regularization parameter (default: 0.1).

p	an exponent for the order of the distance (default: 2).
...	extra parameters including
	abstol stopping criterion for iterations (default: 1e-10).
	init.vec an initial vector (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	print.progress a logical to show current iteration (default: FALSE).
distances	a length-K list where each element is an $(N \times N_k)$ pairwise distance between atoms of the fixed support and given measures.

References

Cuturi M, Doucet A (2014). "Fast computation of wasserstein barycenters." In Xing EP, Jebara T (eds.), *Proceedings of the 31st international conference on international conference on machine learning - volume 32*, volume 32 of *Proceedings of machine learning research*, 685–693. http://proceedings.mlr.press/v32/cuturi14.html.

Examples

```
#-----------------------------------------------------------------
#     Wasserstein Barycenter for Fixed Atoms with Two Gaussians
#
# * class 1 : samples from Gaussian with mean=(-4, -4)
# * class 2 : samples from Gaussian with mean=(+4, +4)
# * target support consists of 7 integer points from -6 to 6,
#   where ideally, weight is concentrated near 0 since it's average!
#-----------------------------------------------------------------
## GENERATE DATA
#  Empirical Measures
set.seed(100)
dat1 = matrix(rnorm(sample(20:50, 1)*2, mean=-4, sd=0.5),ncol=2)
dat2 = matrix(rnorm(sample(20:50, 1)*2, mean=+4, sd=0.5),ncol=2)

measures = list()
measures[[1]] = dat1
measures[[2]] = dat2
mydata = rbind(dat1, dat2)

#  Fixed Support
support = cbind(seq(from=-8,to=8,by=2),
                seq(from=-8,to=8,by=2))
## COMPUTE
comp1 = barysinkhorn14(support, measures, lambda=0.5, maxiter=10)
comp2 = barysinkhorn14(support, measures, lambda=1,   maxiter=10)
comp3 = barysinkhorn14(support, measures, lambda=5,   maxiter=10)

## VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
barplot(comp1, main="lambda=0.5")
barplot(comp2, main="lambda=1")
```

```
barplot(comp3, main="lambda=5")
par(opar)
```

---

digit3 *MNIST Images of Digit 3*

Description

digit3 contains 2000 images from the famous MNIST dataset of digit 3. Each element of the list is an image represented as an (28×28) matrix that sums to 1. This normalization is conventional and it does not hurt its visualization via a basic 'image()' function.

Usage

```
data(digit3)
```

Format

a length-2000 named list "digit3" of (28×28) matrices.

Examples

```
## LOAD THE DATA
data(digit3)

## SHOW A FEW
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,4), pty="s")
for (i in 1:8){
  image(digit3[[i]])
}
par(opar)
```

---

image14C *Barycenter of Images by Cuturi & Doucet (2014)*

Description

Using entropic regularization for Wasserstein barycenter computation, image14C finds a *barycentric* image X^* given multiple images $X_1, X_2, \ldots, X_N$. Please note the followings; (1) we only take a matrix as an image so please make it grayscale if not, (2) all images should be of same size - no resizing is performed.

Usage

```
image14C(images, p = 2, weights = NULL, lambda = NULL, ...)
```

Arguments

images	a length-N list of same-size image matrices of size $(m \times n)$.
p	an exponent for the order of the distance (default: 2).
weights	a weight of each image; if NULL (default), uniform weight is set. Otherwise, it should be a length-N vector of nonnegative weights.
lambda	a regularization parameter; if NULL (default), a paper's suggestion would be taken, or it should be a nonnegative real number.
...	extra parameters including
	abstol stopping criterion for iterations (default: 1e-8).
	init.image an initial weight image (default: uniform weight).
	maxiter maximum number of iterations (default: 496).
	nthread number of threads for OpenMP run (default: 1).
	print.progress a logical to show current iteration (default: TRUE).

Value

an $(m \times n)$ matrix of the barycentric image.

References

Cuturi M, Doucet A (2014). "Fast computation of wasserstein barycenters." In Xing EP, Jebara T (eds.), *Proceedings of the 31st international conference on international conference on machine learning - volume 32*, volume 32 of *Proceedings of machine learning research*, 685–693. http://proceedings.mlr.press/v32/cuturi14.html.

Examples

```
#-----------------------------------------------------------------------
#                       MNIST Data with Digit 3
#
# EXAMPLE 1 : Very Small  Example for CRAN; just showing how to use it!
# EXAMPLE 2 : Medium-size Example for Evolution of Output
#-----------------------------------------------------------------------
# EXAMPLE 1
data(digit3)
datsmall = digit3[1:2]
outsmall = image14C(datsmall, maxiter=3)

## Not run:
# EXAMPLE 2 : Barycenter of 100 Images
# RANDOMLY SELECT THE IMAGES
dat2 = digit3[sample(1:2000, 100)]  # select 100 images

# RUN SEQUENTIALLY
run10 = image14C(dat2, maxiter=10)                  # first 10 iterations
run20 = image14C(dat2, maxiter=10, init.image=run10) # run 40 more
run50 = image14C(dat2, maxiter=30, init.image=run20) # run 50 more
```

```
# VISUALIZE
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,3), pty="s")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(dat2[[sample(100,1)]], axes=FALSE, main="a random image")
image(run10, axes=FALSE, main="barycenter after 10 iter")
image(run20, axes=FALSE, main="barycenter after 20 iter")
image(run50, axes=FALSE, main="barycenter after 50 iter")
par(opar)

## End(Not run)
```

`sinkhorn` *Sinkhorn Distance by Cuturi (2013)*

Description

Due to high computational cost for linear programming approaches to compute Wasserstein distance, Cuturi (2013) proposed an entropic regularization scheme as an efficient approximation to the original problem. This comes with a regularization parameter $\lambda > 0$ in the term

$$\lambda h(\Gamma) = \lambda \sum_{m,n} \Gamma_{m,n} \log(\Gamma_{m,n}).$$

As $\lambda \to 0$, the solution to an approximation problem approaches to the solution of a true problem. However, we have an issue with numerical underflow. Our implementation returns an error when it happens, so please use a larger number when necessary.

Usage

```
sinkhorn(X, Y, p = 2, wx = NULL, wy = NULL, lambda = 0.1, ...)

sinkhornD(D, p = 2, wx = NULL, wy = NULL, lambda = 0.1, ...)
```

Arguments

`X`	an $(M \times P)$ matrix of row observations.
`Y`	an $(N \times P)$ matrix of row observations.
`p`	an exponent for the order of the distance (default: 2).
`wx`	a length-M marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`wy`	a length-N marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`lambda`	a regularization parameter (default: 0.1).
`...`	extra parameters including

maxiter maximum number of iterations (default: 496).

abstol stopping criterion for iterations (default: 1e-10).

D an $(M \times N)$ distance matrix $d(x_m, y_n)$ between two sets of observations.

Value

a named list containing

distance $\mathcal{W}_p$ distance value

iteration the number of iterations it took to converge.

plan an $(M \times N)$ nonnegative matrix for the optimal transport plan.

References

Cuturi M (2013). "Sinkhorn distances: Lightspeed computation of optimal transport." In *Proceedings of the 26th international conference on neural information processing systems - volume 2*, NIPS'13, 2292–2300. Number of pages: 9 Place: Lake Tahoe, Nevada.

Examples

```
#-----------------------------------------------------------------
#  Wasserstein Distance between Samples from Two Bivariate Normal
#
# * class 1 : samples from Gaussian with mean=(-1, -1)
# * class 2 : samples from Gaussian with mean=(+1, +1)
#-----------------------------------------------------------------
## SMALL EXAMPLE
set.seed(100)
m = 20
n = 10
X = matrix(rnorm(m*2, mean=-1),ncol=2) # m obs. for X
Y = matrix(rnorm(n*2, mean=+1),ncol=2) # n obs. for Y

## COMPARE WITH WASSERSTEIN
outw = wasserstein(X, Y)
skh1 = sinkhorn(X, Y, lambda=0.05)
skh2 = sinkhorn(X, Y, lambda=0.10)

## VISUALIZE : SHOW THE PLAN AND DISTANCE
pm1 = paste0("wasserstein plan ; distance=",round(outw$distance,2))
pm2 = paste0("sinkhorn lbd=0.05; distance=",round(skh1$distance,2))
pm5 = paste0("sinkhorn lbd=0.1 ; distance=",round(skh2$distance,2))

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
image(outw$plan, axes=FALSE, main=pm1)
image(skh1$plan, axes=FALSE, main=pm2)
image(skh2$plan, axes=FALSE, main=pm5)
par(opar)
```

`wasserstein` *Wasserstein Distance between Empirical Measures*

Description

Given two empirical measures μ, ν consisting of M and N observations on $\mathcal{X}$, p-Wasserstein distance for $p \geq 1$ between two empirical measures is defined as

$$\mathcal{W}_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{\mathcal{X} \times \mathcal{X}} d(x,y)^p d\gamma(x,y) \right)^{1/p}$$

where $\Gamma(\mu, \nu)$ denotes the collection of all measures/couplings on $\mathcal{X} \times \mathcal{X}$ whose marginals are μ and ν on the first and second factors, respectively. Please see the section for detailed description on the usage of the function.

Usage

```
wasserstein(X, Y, p = 2, wx = NULL, wy = NULL)

wassersteinD(D, p = 2, wx = NULL, wy = NULL)
```

Arguments

`X`	an $(M \times P)$ matrix of row observations.
`Y`	an $(N \times P)$ matrix of row observations.
`p`	an exponent for the order of the distance (default: 2).
`wx`	a length-M marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`wy`	a length-N marginal density that sums to 1. If `NULL` (default), uniform weight is set.
`D`	an $(M \times N)$ distance matrix $d(x_m, y_n)$ between two sets of observations.

Value

a named list containing

distance $\mathcal{W}_p$ distance value

plan an $(M \times N)$ nonnegative matrix for the optimal transport plan.

Usage `wasserstein()` function

We assume empirical measures are defined on the Euclidean space $\mathcal{X} = \mathbf{R}^d$,

$$\mu = \sum_{m=1}^{M} \mu_m \delta_{X_m} \quad \text{and} \quad \nu = \sum_{n=1}^{N} \nu_n \delta_{Y_n}$$

and the distance metric used here is standard Euclidean norm $d(x, y) = \|x - y\|$. Here, the marginals $(\mu_1, \mu_2, \ldots, \mu_M)$ and $(\nu_1, \nu_2, \ldots, \nu_N)$ correspond to `wx` and `wy`, respectively.

Using `wassersteinD()` function

If other distance measures or underlying spaces are one's interests, we have an option for users to provide a distance matrix `D` rather than vectors, where

$$D := D_{M \times N} = d(X_m, Y_n)$$

for flexible modeling.

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. ISSN 1935-8237, 1935-8245, doi: 10.1561/2200000073.

Examples

```
#-----------------------------------------------------------------
#  Wasserstein Distance between Samples from Two Bivariate Normal
#
# * class 1 : samples from Gaussian with mean=(-1, -1)
# * class 2 : samples from Gaussian with mean=(+1, +1)
#-----------------------------------------------------------------
## SMALL EXAMPLE
m = 20
n = 10
X = matrix(rnorm(m*2, mean=-1),ncol=2) # m obs. for X
Y = matrix(rnorm(n*2, mean=+1),ncol=2) # n obs. for Y

## COMPUTE WITH DIFFERENT ORDERS
out1 = wasserstein(X, Y, p=1)
out2 = wasserstein(X, Y, p=2)
out5 = wasserstein(X, Y, p=5)

## VISUALIZE : SHOW THE PLAN AND DISTANCE
pm1 = paste0("plan p=1; distance=",round(out1$distance,2))
pm2 = paste0("plan p=2; distance=",round(out2$distance,2))
pm5 = paste0("plan p=5; distance=",round(out5$distance,2))

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
image(out1$plan, axes=FALSE, main=pm1)
image(out2$plan, axes=FALSE, main=pm2)
image(out5$plan, axes=FALSE, main=pm5)
par(opar)

## Not run:
## COMPARE WITH ANALYTIC RESULTS
#  For two Gaussians with same covariance, their
#  2-Wasserstein distance is known so let's compare !

niter = 5000          # number of iterations
vdist = rep(0,niter)
```

```
for (i in 1:niter){
  mm = sample(30:50, 1)
  nn = sample(30:50, 1)

  X = matrix(rnorm(mm*2, mean=-1),ncol=2)
  Y = matrix(rnorm(nn*2, mean=+1),ncol=2)

  vdist[i] = wasserstein(X, Y, p=2)$distance
  if (i%%10 == 0){
    print(paste0("iteration ",i,"/", niter," complete."))
  }
}

# Visualize
opar <- par(no.readonly=TRUE)
hist(vdist, main="Monte Carlo Simulation")
abline(v=sqrt(8), lwd=2, col="red")
par(opar)

## End(Not run)
```

Index