Package 'RiemBaseExt'

March 21, 2020

Type Package

Title Extension to 'RiemBase' Package

Version 0.1.2

Description
We provide a number of off-the-shelf algorithms for clustering, elementary computation, and others. See Bhattacharya and Bhattacharya (2012) <doi:10.1017/CBO9781139094764> for more details regarding statistics on manifolds.

Encoding UTF-8

License GPL-3

Imports Rcpp, Rdpack, RiemBase (>= 0.2.0), cluster, dbscan, fastcluster, kernlab, parallel, stats, utils, energy

LinkingTo Rcpp, RcppArmadillo, RiemBase

RdMacros Rdpack

RoxygenNote 7.0.2

NeedsCompilation yes

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

Maintainer Kisung You <kyoustat@gmail.com>

Repository CRAN

Date/Publication 2020-03-21 08:50:02 UTC

R topics documented:

RiemBaseExt-package 2
rclust.dbscan 2
rclust.hclust 4
rclust.kmedoids 5
rstat.frechet 6
rstat.pdist 8
rstat.pdist2 9

Index **11**

RiemBaseExt-package *Extension to 'RiemBase' Package*

Description

We provide a number of off-the-shelf algorithms for clustering, elementary computation, and others. See Bhattacharya and Bhattacharya (2012) <doi:10.1017/CBO9781139094764> for more details regarding Statistics on Manifolds.

rclust.dbscan *DBSCAN for Manifold-valued Data*

Description

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a generally applicable clustering algorithm as long as we have concept of dissimilarity. We adopt `dbscan` algorithm by **dbscan** package. See `dbscan` for more details.

Usage

```
rclust.dbscan(input, type = c("extrinsic", "intrinsic"), eps, minPts = 5, ...)
```

Arguments

`input`	a S3 object of `riemdata` class. See `riemfactory` for more details.
`type`	type of distance, either `"intrinsic"` or `"extrinsic"`.
`eps`	size of the epsilon neighborhood.
`minPts`	number of minimum points in the eps region (for core points). Default is 5 points.
`...`	extra parameters for DBSCAN algorithms including `eps` or `minPts`. See `dbscan` for more details.

Value

an object of class `dbscan_fast` containing

eps value of the eps parameter.

minPts value of the minPts parameter.

cluster An integer vector with cluster assignments. Zero indicates noise points.

References

Ester M, Kriegel H, Sander J, Xu X (1996). "A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, 226–231. event-place: Portland, Oregon.

Hahsler M, Piekenbrock M, Doran D (2019). "dbscan : Fast Density-Based Clustering with R." *Journal of Statistical Software*, **91**(1). ISSN 1548-7660, doi: 10.18637/jss.v091.i01.

See Also

See dbscan for details.

Examples

```
## generate 50 points near (0,0,1)  and
##          50 points near (0,0,-1) on Sphere S^2
ndata = 50
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

## wrap it as 'riemdata' class
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],-1)
  data[[i+ndata]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
data = RiemBase::riemfactory(data, name="sphere")

## compare extrinsic and intrinsic DBSCAN
dbext <- rclust.dbscan(data, eps=0.5, type="extrinsic")
dbint <- rclust.dbscan(data, eps=0.5, type="intrinsic")

## let's visualize the results via MDS in R^2
pdist = stats::as.dist(RiemBase::rbase.pdist(data))
dat2d = stats::cmdscale(pdist, k=2)

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
plot(dat2d[,1], dat2d[,2], col=dbext$cluster, pch=19, main="extrinsic+dbscan")
plot(dat2d[,1], dat2d[,2], col=dbint$cluster, pch=19, main="intrinsic+dbscan")
par(opar)
```

rclust.hclust *Hierarchical Agglomerative Clustering for Manifold-valued Data*

Description

Hierarchical clustering is a generally applicable clustering algorithm as long as we have concept of dissimilarity. We adopt `hclust` algorithm by **fastcluster** package. See `hclust` for more details.

Usage

```
rclust.hclust(
  input,
  type = c("extrinsic", "intrinsic"),
  method = c("single", "complete", "average", "mcquitty", "ward.D", "ward.D2",
    "centroid", "median"),
  members = NULL
)
```

Arguments

`input`	a S3 object of `riemdata` class. See `riemfactory` for more details.
`type`	type of distance, either `"intrinsic"` or `"extrinsic"`.
`method`	the agglomeration method to be used. This must be (an unambiguous abbreviation of) one of `"single"`, `"complete"`, `"average"`, `"mcquitty"`, `"ward.D"`, `"ward.D2"`, `"centroid"` or `"median"`.
`members`	`NULL` or a vector whose length equals the number of observations. See `hclust` for details.

Value

an object of class `hclust`. See `hclust` for details.

References

Müllner D (2013). "fastcluster : Fast Hierarchical, Agglomerative Clustering Routines for R and Python." *Journal of Statistical Software*, **53**(9). ISSN 1548-7660, doi: 10.18637/jss.v053.i09.

Examples

```
## generate 50 points near (0,0,1)  and
##          50 points near (0,0,-1) on Sphere S^2
ndata = 50
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

## wrap it as 'riemdata' class
```

```
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],-1)
  data[[i+ndata]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
data = RiemBase::riemfactory(data, name="sphere")

## compare extrinsic and intrinsic hierarchical clustering
hext <- rclust.hclust(data, type="extrinsic")
hint <- rclust.hclust(data, type="intrinsic")

## visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2), pty="s")
plot(hext, main="extrinsic+single")
plot(hint, main="intrinsic+single")
par(opar)
```

`rclust.kmedoids` *k-Medoids Clustering Manifold-valued Data*

Description

k-Medoids is a generally applicable clustering algorithm as long as we have concept of dissimilarity. We adopt pam algorithm by **cluster** package. See pam for more details.

Usage

```
rclust.kmedoids(input, k = 2, type = c("extrinsic", "intrinsic"))
```

Arguments

input	a S3 object of riemdata class. See riemfactory for more details.
k	the number of clusters.
type	type of distance, either "intrinsic" or "extrinsic".

Value

an object of class pam. See pam for details.

Examples

```
## generate 50 points near (0,0,1)  and
##          50 points near (0,0,-1) on Sphere S^2
ndata = 50
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

## wrap it as 'riemdata' class
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],-1)
  data[[i+ndata]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
data = RiemBase::riemfactory(data, name="sphere")

## compare k-medoids with intrinsic distance for different k's
k2 <- rclust.kmedoids(data, k=2, type="intrinsic")
k3 <- rclust.kmedoids(data, k=3, type="intrinsic")
k4 <- rclust.kmedoids(data, k=4, type="intrinsic")

## let's visualize the results via MDS in R^2
pdist = stats::as.dist(RiemBase::rbase.pdist(data))
dat2d = stats::cmdscale(pdist, k=2)

opar <- par(no.readonly=TRUE)
par(mfrow=c(1,3))
plot(dat2d[,1], dat2d[,2], col=k2$clustering, pch=19, main="k=2")
plot(dat2d[,1], dat2d[,2], col=k3$clustering, pch=19, main="k=3")
plot(dat2d[,1], dat2d[,2], col=k4$clustering, pch=19, main="k=4")
par(opar)
```

`rstat.frechet` *Fréchet Mean and Variation of Manifold-valued Data*

Description

For manifold-valued data, Fréchet mean is the solution of following cost function,

$$\min_x \sum_{i=1}^{n} \rho^2(x, x_i), \quad x \in \mathcal{M}$$

for a given data $\{x_i\}_{i=1}^n$ and $\rho(x, y)$ is the geodesic distance between two points on manifold $\mathcal{M}$. It uses a gradient descent method with a backtracking search rule for updating.

Usage

```
rstat.frechet(
  input,
  type = c("intrinsic", "extrinsic"),
  int.eps = 1e-06,
  parallel = FALSE
)
```

Arguments

input	a S3 object of riemdata class. See riemfactory for more details.
type	type of distance, either "intrinsic" or "extrinsic".
int.eps	stopping criterion for the norm of gradient.
parallel	a flag for enabling parallel computation.

Value

a named list containing

mu an estimated Fréchet mean matrix.

variation Fréchet variation with the estimated mean.

References

Karcher H (1977). "Riemannian center of mass and mollifier smoothing." *Communications on Pure and Applied Mathematics*, **30**(5), 509–541. ISSN 00103640, 10970312, doi: 10.1002/cpa.3160300502.

Kendall WS (1990). "Probability, Convexity, and Harmonic Maps with Small Image I: Uniqueness and Fine Existence." *Proceedings of the London Mathematical Society*, **s3-61**(2), 371–406. ISSN 00246115, doi: 10.1112/plms/s361.2.371.

Afsari B, Tron R, Vidal R (2013). "On the Convergence of Gradient Descent for Finding the Riemannian Center of Mass." *SIAM Journal on Control and Optimization*, **51**(3), 2230–2260. ISSN 0363-0129, 1095-7138, doi: 10.1137/12086282X.

Examples

```
### Generate 50 data points on Sphere S^2 near (0,0,1).
ndata = 50
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

### Wrap it as 'riemdata' class
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
data = RiemBase::riemfactory(data, name="sphere")
```

```
### Compute Fréchet Mean and Variation
out1 = rstat.frechet(data)                  # intrinsic
out2 = rstat.frechet(data,parallel=TRUE)    # parallel implementation
out3 = rstat.frechet(data, type="extrinsic") # extrinsic
```

`rstat.pdist` *Pairwise Distance of Data on Manifolds*

Description

For two points $x, y \in \mathcal{M}$, two modes of distances are available; *intrinsic* for geodesic distance on the manifold and *extrinsic* for standard norm after equivariant embedding into Euclidean space.

Usage

```
rstat.pdist(input, type = c("intrinsic", "extrinsic"), as.dist = TRUE)
```

Arguments

`input`	a S3 object of `riemdata` class. See `riemfactory` for more details.
`type`	type of distance, either `"intrinsic"` or `"extrinsic"`.
`as.dist`	logical; if `TRUE`, it returns `dist` object, else it returns a symmetric matrix.

Value

a S3 `dist` object or symmetric matrix of pairwise distances according to `type` parameter.

Examples

```
### Generate 50 data points on Sphere S^2 near (0,0,1).
ndata = 50
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta) + rnorm(ndata,sd=0.1)
tmpy  = sin(theta) + rnorm(ndata,sd=0.1)

### Wrap it as 'riemdata' class
data  = list()
for (i in 1:ndata){
  tgt = c(tmpx[i],tmpy[i],1)
  data[[i]] = tgt/sqrt(sum(tgt^2)) # project onto Sphere
}
spdata = RiemBase::riemfactory(data, name="sphere")

### Compute Two Types of Distances and Visualize
dist.int = rstat.pdist(spdata, type="intrinsic", as.dist=FALSE)
dist.ext = rstat.pdist(spdata, type="extrinsic", as.dist=FALSE)
```

```
### Visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
image(dist.int, main="intrinsic")
image(dist.ext, main="extrinsic")
par(opar)
```

`rstat.pdist2` *Pairwise Distance for Two Sets of Data on Manifolds*

Description

For two points $x, y \in \mathcal{M}$, two modes of distances are available; *intrinsic* for geodesic distance on the manifold and *extrinsic* for standard norm after equivariant embedding into Euclidean space. This function differs from `rstat.pdist` in that it now computes distances between two sets of data.

Usage

```
rstat.pdist2(input1, input2, type = c("intrinsic", "extrinsic"))
```

Arguments

`input1`	a S3 object of `riemdata` class of N objects.
`input2`	a S3 object of `riemdata` class of M objects. See `riemfactory` for more details.
`type`	type of distance, either `"intrinsic"` or `"extrinsic"`.

Value

an $(N \times M)$ matrix of pairwise distances.

See Also

`rstat.pdist`

Examples

```
### Generate 100 data points on Sphere S^2.
#   50 points from near (0,0,1), and
#   50 points from near (0,0,-1)

ndata = 50
theta = seq(from=-0.99,to=0.99,length.out=ndata)*pi
tmpx  = cos(theta)
tmpy  = sin(theta)

### Wrap those as 'riemdata' class
data1 = list()
data2 = list()
```

```
for (i in 1:ndata){
  tgt1 = c(tmpx[i],tmpy[i],1)  + stats::rnorm(3,sd=0.1)
  tgt2 = c(tmpx[i],tmpy[i],-1) + stats::rnorm(3,sd=0.1)

  data1[[i]] = tgt1/sqrt(sum(tgt1^2)) # projection near (0,0,1)
  data2[[i]] = tgt2/sqrt(sum(tgt2^2)) #                 (0,0,-1)
}
spdata1 = RiemBase::riemfactory(data1, name="sphere")
spdata2 = RiemBase::riemfactory(data2, name="sphere")

### Compute Two Types of Distances and Visualize
dist.int = rstat.pdist2(spdata1, spdata2, type="intrinsic")
dist.ext = rstat.pdist2(spdata1, spdata2, type="extrinsic")

### Visualize
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
image(dist.int, main="intrinsic")
image(dist.ext, main="extrinsic")
par(opar)
```

Index

dbscan, *2, 3*

hclust, *4*

pam, *5*

rclust.dbscan, 2
rclust.hclust, 4
rclust.kmedoids, 5
RiemBaseExt-package, 2
riemfactory, *2*, *4, 5*, *7–9*
rstat.frechet, 6
rstat.pdist, 8, *9*
rstat.pdist2, 9