Package ‘ppMeasures’

February 15, 2013

Type Package

Title Point pattern distances and prototypes.

Version 0.2

Date 2012-11-07

Author David M Diez, Katherine E Tranbarger Freier, and Frederic P Schoenberg

Maintainer David M Diez <david.m.diez@gmail.com>

Description The package focuses on distances and prototypes for point patterns. There are three algorithms provided to compute spike-time distance, and one of these algorithms is generalized to compute variations of spike-time distance. Multiple algorithms are also provided to estimate prototypes of collections of point patterns.

License GPL (>= 2)

LazyLoad yes

Depends graphics, grDevices, stats, utils, R (>= 2.10)

Repository CRAN

Date/Publication 2012-11-07 06:04:05

NeedsCompilation yes

R topics documented:

ppMeasures-package .................................................. 2
collEx1 ................................................................. 3
collEx2 ................................................................. 4
neurBd ................................................................. 5
neurCond .............................................................. 6
neurNaive ............................................................. 8
pattEx1 ................................................................. 9
Description

The package focuses on distances and prototypes for point patterns. There are three algorithms provided to compute spike-time distance, and one of these algorithms is generalized to compute variations of spike-time distance. Multiple algorithms are also provided to estimate prototypes of collections of point patterns.

Details

```
Package:               ppMeasures
Type:                 Package
Version:              0.2
Date:                 2012-11-07
License:              GPL (>= 2)
LazyLoad:             yes
```

~ An overview of how to use the package, including the most important ~ ~ functions ~

Author(s)

DM Diez, KE Tranbarger Freier, and FP Schoenberg
Maintainer: DM Diez <david.m.diez@gmail.com>

References


See Also

`stDist, ppColl, ppPrototype`
Examples

data(pattEx2)
x <- pattEx2[pattEx2[,1] == 1,c(2,3)]
y <- pattEx2[pattEx2[,1] == 2,c(2,3)]
(hold2 <- stDist(x, y, 2))
summary(hold2)
plot(hold2)

data(collEx2)
(ppc2 <- ppColl(collEx2[,2:3], collEx2[,1]))
summary(ppc2)
plot(ppc2, pch=0.5)
hold2 <- ppPrototype(ppc2, pm=0.05)
points(hold2, pch=20, cex=3, col='#FF000088'

### reproducing results from Diez et al. (in review) ###
# results differ slightly due to default use of "margPT" algorithm # and that cats are not weighted equally in the below analysis
par(mfrow=c(3,1))

data(neurNaive)
(ppc3 <- ppColl(neurNaive[,2], neurNaive[,1], nMissing=))
summary(ppc3)
plot(ppc3, cex=0.5)
hold3 <- ppPrototype(ppc3, pm=0.0106, bypassCheck=TRUE)
points(hold3, pch=20, cex=3, col='#FF00044'

data(neurCond)
(ppc4 <- ppColl(neurCond[,2], neurCond[,1]))
summary(ppc4)
plot(ppc4, cex=0.5)
hold4 <- ppPrototype(ppc4, pm=0.0075, bypassCheck=TRUE)
points(hold4, pch=20, cex=3, col='#FF00044'

data(neurBd)
(ppc5 <- ppColl(neurBd[,2], neurBd[,1]))
summary(ppc5)
plot(ppc5, cex=0.5)
hold5 <- ppPrototype(ppc5, pm=0.0078, bypassCheck=TRUE)
points(hold5, pch=20, cex=3, col='#FF00044'

---

**collEx1**

*Collection of one dimensional patterns*

Description

A collection of one dimensional patterns.

Usage

data(collEx1)
collEx2

Collection of two dimensional patterns

description

A collection of two dimensional patterns.

Usage

data(collEx2)

Format

A data frame with 134 observations on the following 3 variables.

key  A numeric vector corresponding to the pattern numbers.

x    A numeric vector corresponding to the first dimension of the points.

y    A numeric vector corresponding to the second dimension of the points.
Examples

data(collEx2)

#===> construct the collection <===#
ppc <- ppColl(collEx2[,2:3], collEx2[,1])
plot(ppc)

#===> compute the prototype via use margPT <===#
protoMP <- ppPrototype(ppc, 0.1)
points(protoMP, pch=20, cex=3.5, col='#FF000088')
# both algorithms produce very similar results

neurBd  

Collection of cell prototypes from backwards conditioned cats

Description

Collection of cell prototypes from backwards conditioned cats. This data originates from an experiment testing the effect of conditioning and backwards conditioning on the neural responses of cats (Woody et al., 2000). This data pertains to the backwards conditioned cats (after stimulus conditioning and backwards conditioning) and their neuron cells’ prototypical responses to a click stimulus at time 0 ms that were observed between -320 ms and 1280 ms. Originally each cell had multiple runs, and these have been simplified to cell prototypes in this data set.

Usage

data(neurBd)

Format

A data frame with 752 observations on the following 3 variables.

key  A numeric vector corresponding to the cell numbers.
pts  A numeric vector corresponding to neuronal responses.
cat  A numeric vector corresponding to the cat.

Details

Each cell originally had several observed patterns. To simplify this complex collection, a prototype was found for each cell.

An analysis incorporating the weights of the cats is included in the examples (Diez et al., 2010). Six cells had zero points in their prototype (0 for cat 1, 1 for 2, 5 for 3).

Source

References


Examples

data(neurBd)

#==> construct weights <===#
# wts found separately
counts <- c(0,1,5)
wtsKey <- 1:78
# cat information from original data
cats <- c(rep(1,22), rep(2,6), rep(3,50))
for(i in 1:3){
  these <- unique(neurBd[,3] == i,1)
  counts[i] <- counts[i] + length(these)
}
sum(counts)
wts <- 1/counts[cats]

#==> setup collection <===#
pts <- neurBd[,2]
key <- neurBd[,1]
ppc <- ppColl(pts, key, wts, wtsKey)

#==> identify prototype <===#
(protoKP <- ppPrototype(ppc, 0.0078, pa=1.125, pd=0.885, alg='kernPT', ppd=5000, bypassCheck=TRUE))
plot(ppc)
points(protoKP, col=4, cex=1.5, at=-5)

neurCond

Collection of cell prototypes from conditioned cats

Description

Collection of cell prototypes from conditioned cats. This data originates from an experiment testing the effect of conditioning and backwards conditioning on the neural responses of cats (Woody et al., 2000). This data pertains to the cats following stimulus conditioning and their neuron cells’ prototypical responses to a click stimulus at time 0 ms that were observed between -320 ms and 1280 ms. Originally each cell had multiple runs, and these have been simplified to cell prototypes in this data set.

Usage

data(neurCond)
**Format**

A data frame with 2855 observations on the following 3 variables.

- **key**: A numeric vector corresponding to the cell numbers.
- **pts**: A numeric vector corresponding to neuronal responses.
- **cat**: A numeric vector corresponding to the cat.

**Details**

Each cell originally had several observed patterns. To simplify this complex collection, a prototype was found for each cell.

An analysis incorporating the weights of the cats is included in the examples (Diez et al., 2010). Thirty-five cells had zero points in their prototype (6 for cat 1, 4 for 2, 15 for 3, 5 for 4, 5 for 5).

**Source**


**References**


**Examples**

data(neurCond)

```r
#===> construct weights <===#
# wts found separately
counts <- c(6,4,15,5,5)
wtsKey <- 1:256

# cat information from original data
cats <- c(rep(1,29), rep(2,55), rep(3,31), rep(4,16),
rep(3,54), rep(5,44), rep(3,27))
for(i in 1:5){
  these <- unique(neurCond[,3] == i,1])
counts[i] <- counts[i] + length(these)
}
sum(counts)
wts <- 1/counts[cats]

#===> setup collection <===#
pts <- neurCond[,2]
key <- neurCond[,1]
ppc <- ppColl(pts, key, wts, wtsKey)
# did not need to list nMissing since implied through wts/wtsKey
```
neurNaive

Collection of cell prototypes from naive cats

Description

Collection of cell prototypes from naive cats. This data originates from an experiment testing the effect of conditioning and backwards conditioning on the neural responses of cats (Woody et al., 2000). This data pertains to naive cats (prior to stimulus conditioning) and their neuron cells’ prototypical responses to a click at time 0 ms that were observed between -320 ms and 1280 ms. Originally each cell had multiple runs, and these have been simplified to cell prototypes in this data set.

Usage

data(neurNaive)

Format

A data frame with 3177 observations on the following 3 variables.

key  A numeric vector corresponding to the cell numbers.
pts  A numeric vector corresponding to neuronal responses.
cat  A numeric vector corresponding to the cat.

Details

Each cell originally had several observed patterns. To simplify this complex collection, a prototype was found for each cell.

An analysis incorporating the weights of the cats is included in the examples (Diez et al., 2010). Twelve cells had zero points in their prototype (3 for cat 1, 1 for 2, 3 for 3, 5 for 4).

Source


References

Examples

```r
data(neurNaive)

### construct weights ###
# wts found separately
counts <- c(3,1,3,5)
wtsKey <- 1:268
# cat information from original data
cats <- c(rep(1,66), rep(2,39), rep(3,80), rep(4,83))
for(i in 1:4){
  these <- unique(neurNaive[neurNaive[,3] == i,])
counts[i] <- counts[i] + length(these)
}
sum(counts)
wts <- 1/countscats

### setup collection ###
pts <- neurNaive[,2]
key <- neurNaive[,1]
ppc <- ppColl(pts, key, wts, wtsKey)

### identify prototype ###
(protoKP <- ppPrototype(ppc, pa=1.1, pd=0.9, alg='kernPT', ppd=5000, bypassCheck=TRUE))
plot(ppc)
points(protoKP, col=4, cex=1.5, at=-10)
```

---

`pattEx1`  
*Collection of 2 single dimensional patterns*

Description

Collection of 2 single dimensional point patterns.

Usage

```r
data(pattEx1)
```

Format

A data frame with 16 observations on the following 2 variables.

- **key**  A numeric vector indicating which points belong to which patterns.
- **pts**  A numeric vector corresponding to the points in the patterns.
Examples

data(pattEx1)

#===> obtain the patterns <====#
 x <- pattEx1[pattEx1[,1] == 1,2]
 y <- pattEx1[pattEx1[,1] == 2,2]
 par(mfrow=1:2)

#===> compute distance via VP97 <====#
(dVP <- stDist(x, y, 0.5, alg='VP97'))
 summary(dVP)
 plot(dVP)

#===> compute distance via IMA <====#
(dIMA <- stDist(x, y, 0.5, alg='IMA'))
 summary(dIMA)
 plot(dIMA)

#===> compute distance via MSU <====#
(dMSU <- stDist(x, y, 0.5, alg='MSU'))
 summary(dMSU)

pattEx2

Collection of two 2D patterns

Description

Collection of two 2D patterns.

Usage

data(pattEx2)

Format

A data frame with 18 observations on the following 3 variables.

key  A numeric vector indicating which points belong to which patterns.
 x  A numeric vector corresponding to the first dimension of the points in the patterns.
 y  A numeric vector corresponding to the second dimension of the points in the patterns.

Examples

data(pattEx2)

#===> obtain the patterns <====#
 x <- pattEx2[pattEx2[,1] == 1:2]
 y <- pattEx2[pattEx2[,1] == 2:3]
#===> compute distance (via IMA) <===#
    (d <- stDist(x, y, 0.5))
    summary(d)
    plot(d)

---

**ppColl**

Creating a point pattern collection object

**Description**

Prepare a point pattern collection for use in the prototype function, *ppPrototype*.

**Usage**

```
ppColl(pts, key, wts = NULL, wtsKey = NULL, nMissing = 0)
```

**Arguments**

- **pts**
  A data matrix where rows represent points and columns dimensions.

- **key**
  A vector designating which points (rows of *pts*) belong to which patterns. Patterns without points should also have weights assigned.

- **wts**
  A numerical vector to specify weights for the patterns.

- **wtsKey**
  A vector. If *wts* are specified, then also specify the keys that those weights correspond to using *wtsKey*. This vector should have length equal to that of *wts* and include all unique values in *key*. Patterns without points should have a name assigned.

- **nMissing**
  The number of patterns with no points. This argument is ignored if weights are specified, in which case patterns without points should be assigned a weight in *wts* and a pattern name in *wtsKey*.

**Value**

An object of class "ppColl" is a list containing the following components:

- **points**
  A data matrix of the points.

- **pattID**
  Original key proposed by the user.

- **key**
  Key for identifying which points belong to which pattern.

- **keyMax**
  The number of patterns.

- **wts**
  Weights for each pattern.

- **dim**
  Dimension of data.

- **maxObs**
  The maximum number of observations in any of the patterns.

- **obs**
  The number of observations in each pattern.
Author(s)
DM Diez

See Also
ppPrototype, stDist

Examples

```r
data(collEx1)
(ppc1 <- ppColl(collEx1[,2], collEx1[,1]))
summary(ppc1)
plot(ppc1, ylim=c(-3, ppc1$keyMax+3))
hold1 <- ppPrototype(ppc1, pm=0.5)
points(hold1, pch=20, cex=3, col='#FF000088')

data(collEx2)
(ppc2 <- ppColl(collEx2[,2:3], collEx2[,1]))
summary(ppc2)
plot(ppc2, cex=0.5)
hold2 <- ppPrototype(ppc2, pm=0.05)
points(hold2, pch=20, cex=3, col='#FF000088')
```

---

**ppPrototype**

**Point pattern prototype estimation**

Description

This function estimates the prototype of a collection of point patterns.

Usage

```r
ppPrototype(ppcoll, pm, pa = 1, pd = 1, alg = c("margPT", "kernPT", "VP97", "MSU"),
            costEps = 10^(-6), posEps = 10^(-6), ppd = 4, space = 0.1, euclid = NULL,
            lossOrder = 1, maxBranch = 4, bypassCheck = FALSE)
```

Arguments

- `ppcoll` A collection of point patterns, output from the function `ppColl`.
- `pm` A number or numerical vector representing the moving penalty. If there are multiple dimensions, different moving penalties may be specified for each dimension by using a vector for `pm`.
- `pa` A number representing the addition penalty.
- `pd` A number representing the deletion penalty.
alg
Character string specifying the algorithm. "margPT" corresponds to a marginal prototype technique tied that utilizes the Incremental Matching Algorithm, which is suitable for multiple dimensions. Other (one-dimensional) methods check all possible prototype point locations and utilize different algorithms for the distance: "kernPT" for kernel smoothing, "VP97" for the algorithm presented in Victor and Purpura (1997), or "MSU" for the Modified Single Unit Algorithm.

costEps
If this reduction in cost is not achieved, the algorithm will stop.

posEps
A number or numerical vector for precision control for identification of prototype points. For instance, if the algorithm converges to a region of diameter smaller than posEps, it will complete its search for the prototype point. If working in multidimensional space and a numerical vector is used, the value in position i of posEps corresponds to the precision control for dimension i.

ppd
An integer or numerical vector of integers utilized in alg="margPT" to specify the number of points per dimension to examine at a single time. Each value must be at least 2. If a vector, element i corresponds to the number of points to examine in dimension i at a single time.

space
A value greater than 0 but less than 0.5, which governs the spread of the points in each dimension when using alg="margPT". Just as with ppd, space can be input as a vector where the i-th value corresponds to dimension i. Restriction on space: for any dimension, their product must be less than 0.5.

euclid
A vector of 1s and 0s. The vector should have one entry per dimension, and dimensions corresponding to a 1 are incorporated as a Euclidean dimension for Euclidean distance between points. This argument is only utilized when alg="margPT".

lossOrder
A number. The moving penalty is computed using the moving penalty and distance a point is moved. The lossOrder specifies the power of the distance moved in the penalty computation, e.g. for one non-Euclidean dimension, the cost of moving a point a distance d is pm*d^lossOrder. This argument is only utilized when alg="margPT".

maxBranch
A non-negative integer used as a parameter for accuracy when alg="margPT". Values larger than 2 typically result in an exact computation of distance while using a maxBranch of 0 or 1 will reduce accuracy but results in a faster computation.

bypassCheck
If patterns may be too large for the specified algorithm, an error will occur. To bypass this error and proceed anyways, specify bypassCheck=TRUE.

Details
Doing a test with a smaller data set is recommended since large data sets may require a significant amount of time for computation. Computation time is dependent not only on the number of patterns or the number of points in the patterns, but also dimension, ppd, and other settings and collection characteristics.

Value
A list containing the following items:
algorithm The algorithm utilized.
ppColl The collection for which the prototype was found.
ppColl The collection for which the prototype was found.
prototype Matrix representing the prototype.
mismatch How points in the collection are matched to the prototype.
cost Total cost: weighted sum of the distances from the prototype to each pattern.
ap Additon penalty.
ap Deletion penalty.
ap Moving penalty.
ppd Points per dimension, if utilized the "margPT" algorithm.
space Spacing parameter of prototype search, if utilized the "margPT" algorithm.
dim Dimension of the space.
costEps costEps parameter.
posEps posEps parameter.
maxBranch maxBranch parameter.
lossOrder lossOrder parameter.
euclid euclid parameter.
dist The distances from the prototype to each pattern in the collection.

Author(s)
DM Diez

References

See Also
ppColl, stDist

Examples
#===> one dimensional example <===#
data(collEx1)
ppc <- ppColl(collEx1[,2], collEx1[,1])
plot(ppc, ylim=c(-3, 28))
protoVP <- ppPrototype(ppc, 0.5, alg='VP97')
summary(protoVP)
points(protoVP, pch=20, cex=2, col='#FF000088')
stDist <- ppPrototype(ppc, 0.5)
points(stDist, pch=20, cex=2, col='#00FF0088', at=-1)

protoKP <- ppPrototype(ppc, 0.5, alg='kernPT', ppd=500)
points(protoKP, pch=20, cex=2, col='#FFFF0088', at=-2)
protoSP <- ppPrototype(ppc, 0.5, alg='MSU', ppd=500, bypassCheck=TRUE)
points(protoSP, pch=20, cex=2, col='#FFFF0088', at=-3)

#===> two dimensional example <===#
data(collEx2)
ppc <- ppColl(collEx2[,2:3], collEx2[,1])
plot(ppc)
protoMP <- ppPrototype(ppc, 0.1)
points(protoMP, pch=20, cex=3.5, col='#FF000088')

---

stDist

Spike-time distance computation

Description

This function computes spike-time distance and its variants.

Usage

stDist(x, y, pm, pa = 1, pd = 1, alg = c("default", "IMA", "VP97", "MSU"), euclid = NULL, lossOrder = 1, maxBranch = 4, eps = 1e-10, bypassCheck = FALSE)

Arguments

x A matrix representing a point pattern where rows represent points and columns dimensions. If there is only one dimension, x may be a vector.
y A matrix representing a point pattern where rows represent points and columns dimensions. If there is only one dimension, y may be a vector.
pm A number or numerical vector representing the moving penalty. If there are multiple dimensions, different moving penalties may be specified for each dimension by using a vector for pm.
apa A number representing the addition penalty.
pd A number representing the deletion penalty.
alg Character string specifying which algorithm should be applied. "IMA" corresponds to an incremental matching algorithm for spike-time distance and its variants in any finite dimensional space. "VP97" corresponds to a slightly generalized algorithm from Victor and Purpura (1997) for spike-time distance. "MSU" corresponds to the Modified Single Unit algorithm for spike-time distance (Trabarger and Schoenberg, 2010). By default, "VP97" will be used for one-dimensional data and "IMA" will be used for multiple dimensions. If variants of spike-time distance are used (euclid or lossOrder are adjusted), then the "IMA" technique will be applied regardless of the specified algorithm.
euclid  A vector of 1s and 0s. The vector should have one entry per dimension, and dimensions corresponding to a 1 are incorporated as a Euclidean dimension for Euclidean distance between points.

lossOrder  A number. The moving penalty is computed using the moving penalty and distance a point is moved. The lossOrder specifies the power of the distance moved in the penalty computation, e.g. for one non-Euclidean dimension, the cost of moving a point a distance d is pm*d^lossOrder.

maxBranch  A non-negative integer used as a parameter for accuracy when alg="IMA". Values larger than 2 typically result in an exact computation of distance while using a maxBranch of 0 or 1 will reduce accuracy but results in a faster computation.

eps  If reductions in distance are not larger than this amount, the algorithm will stop.

bypassCheck  If patterns may be too large for the specified algorithm, an error will occur. To bypass this error and proceed anyways, specify bypassCheck=TRUE.

Details
This function computes the distance from x to y.

Value
An object of class "stDist" is a list containing the following components:

distance  The computed distance.
x  The first pattern.
y  The second pattern.
pa  Addition penalty.
pd  Deletion penalty.
pm  Moving penalty.
xtoy  Under the optimal matching, x[i,] matches to y[xtoy[i],]. Not available when alg="MSU".

ytox  Under the optimal matching, y[i,] matches to x[ytox[i],]. Not available when alg="MSU".
dim  Number of dimensions.
eps  The specified eps.
maxBranch  The specified maxBranch.
lossOrder  The specified lossOrder.
algorithm  The algorithm utilized.
euclid  The specified euclid.

Author(s)
DM Diez, KE Tranbarger Freier, and FP Schoenberg
stDist

References


See Also

ppColl, ppPrototype

Examples

data(pattEx1)
x <- pattEx1[pattEx1[,1] == 1,2]
y <- pattEx1[pattEx1[,1] == 2,2]
(hold1 <- stDist(x, y, 4))
(hold11 <- stDist(x, y, 4, alg='MSU'))
summary(hold1)
summary(hold11)
plot(hold1)

data(pattEx2)
x <- pattEx2[pattEx2[,1] == 1,c(2,3)]
y <- pattEx2[pattEx2[,1] == 2,c(2,3)]
(hold2 <- stDist(x, y, 2))
summary(hold2)
plot(hold2)
Index

*Topic **Point pattern collection**
  ppColl, 11
*Topic **Point pattern prototype**
  ppColl, 11
*Topic **datasets**
  collEx1, 3
collEx2, 4
neurBd, 5
neurCond, 6
neurNaive, 8
pattEx1, 9
pattEx2, 10
*Topic **package**
  ppMeasures-package, 2
*Topic **point pattern collection**
  ppPrototype, 12
*Topic **point pattern distance**
  ppMeasures-package, 2
*Topic **point pattern prototypes**
  ppPrototype, 12
*Topic **point pattern prototype**
  ppMeasures-package, 2
*Topic **point pattern**
  ppMeasures-package, 2
  stDist, 15
*Topic **point process**
  ppMeasures-package, 2
*Topic **spike-time distance**
  ppPrototype, 12
  stDist, 15

collEx1, 3
collEx2, 4

neurBd, 5
neurCond, 6
neurNaive, 8

pattEx1, 9
pattEx2, 10

plot.ppColl (ppColl), 11
plot.ppPrototype (ppPrototype), 12
plot.stDist (stDist), 15
points.ppColl (ppColl), 11
points.ppPrototype (ppPrototype), 12
ppColl, 2, 11, 12, 14, 17
ppMeasures (ppMeasures-package), 2
ppMeasures-package, 2
ppPrototype, 2, 11, 12, 12, 17
print.ppColl (ppColl), 11
print.ppPrototype (ppPrototype), 12
print.stDist (stDist), 15

stDist, 2, 12, 14, 15
summary.ppColl (ppColl), 11
summary.ppPrototype (ppPrototype), 12
summary.stDist (stDist), 15