Package: cccd (via r-universe)

November 4, 2024

Version 1.6

Date 2022-04-08

Title Class Cover Catch Digraphs

Author David J. Marchette <dmarchette@gmail.com>

Maintainer David J. Marchette <dmarchette@gmail.com>

Depends igraph

Imports proxy, deldir, FNN

Suggests Matrix

Description Class Cover Catch Digraphs, neighborhood graphs, and relatives.

License GPL (>= 2)

Date/Publication 2022-04-08 12:22:29 UTC

NeedsCompilation no

Repository https://dmarchette.r-universe.dev

RemoteUrl https://github.com/cran/cccd

RemoteRef HEAD

RemoteSha de4ce40da7a3b119ee4d19055a910ee9a4ad2a63

Contents

cccd *Class Cover Catch Digraph*

Description

Constructs a class cover catch digraph from points or interpoint distance matrices.

Usage

```
cccd(x = NULL, y = NULL, dxx = NULL, dyx = NULL, method = NULL,
     k = NA, algorithm = 'cover_tree')
cccd.rw(x=NULL,y=NULL,dxx=NULL,dyx=NULL,method=NULL,m=1,d=2)
cccd.classifier(x,y,dom.method='greedy',proportion=1,...)
cccd.classify(data, C,method=NULL)
cccd.classifier.rw(x,y,m=1,d=2)
cccd.multiclass.classifier(data, classes, dom.method='greedy',proportion=1,...)
cccd.multiclass.classify(data,C,method=NULL)
## S3 method for class 'cccd'
plot(x, ..., plot.circles = FALSE, dominate.only = FALSE,
          D = NULL, vertex.size = 2, vertex.label = NA,
    vertex.color = "SkyBlue2", dom.color = "Blue",
    ypch = 20, ycex = 1.5, ycol = 2,
    use.circle.radii = FALSE, balls = FALSE,
    ball.color = gray(0.8), square = FALSE, xlim, ylim)
## S3 method for class 'cccdClassifier'
plot(x, ..., xcol=1,ycol=2,xpch=20,ypch=xpch,
                                balls=FALSE,add=FALSE)
```

Arguments

x, y	the target class and non-target class points. Either x,y or dxx,dyx must be provided. In the case of `plot`, x is an object of class cccd.
dxx, dyx	interpoint distances (x against x and y against x). If these are not provided they are computed using x and y.
method	the method used for the distance. See `dist`.
dom.method, proportion	the method used for the domination set computation, and the proportion of points required to dominate. See `dominate`.
k	If given, `get.knn` is used from FNN to approximate the class cover catch graph. Each x covers no more than the `k` nearest neighbors to it. This will be much faster and use less memory for large data sets, but is an approximation unless `k` is sufficiently large.
algorithm	See `get.knn`.
m	slope of the null hypothesis curve
data	data to be classified

`d`	dimension of the data
`classes`	class labels of the data
`C`	cccd object
`plot.circles`	logical. Plot the circles around the points if TRUE.
`dominate.only`	logical. Only plot the digraph induced by the dominating set.
`D`	a dominating set. Only used if dominate.only is TRUE. If dominate.only is TRUE and D is NULL, then `dominate` is called.
`vertex.size`, `vertex.color`, `vertex.label`, `dom.color`	parameters controling the plotting of the vertices. `dom.color` is the color of the vertices in the dominating set.
`balls`, `ball.color`	if `balls`=TRUE, the cover is plotted as filled balls, with `ball.color` controling their color. In the cass of `cccdClassifier`, `balls` can be "`x`" or "`y`" indicating that only one of the balls should be plotted.
`ypch`, `ycex`, `ycol`	parameters for plotting the non-target points.
`xpch`, `xcol`	parameters for plotting the first class points.
`add`	logical. Should the classifier plot be added to an existing plot?
`use.circle.radii`	logical. Ensure that the circles fit within the plot.
`square`	logical. Make the plot square.
`xlim`, `ylim`	if present, these control the plotting region.
`...`	arguments passed to `cccd` or `plot`.

Details

The class cover catch digraph is a graph with vertices defined by the points of `x` and edges defined according to the balls $B(x, d(x, Y))$. There is an edge between vertices x_1, x_2 if $x_2 \in B(x_1, d(x_1, Y))$. If `dyx` is not given and the method is 'euclidean', then `get.knnx` is used to find the nearest `y` to each `x`. If `k` is given, only the `k` nearest neighbors to each point are candidates for covering. Thus the cccd will be approximate, but the computation will (generally) be faster. Since `get.knn` uses Euclidean distance, these choices will only be valid for this distance metric. Since the graph will tend to be larger than otherwise, the dominating set computation will be slower, so one should trade-off speed of calculation, approximation, and the `proportion` option to the dominating set (which can make that calculation faster at the cost of returning a subset of the dominating set).

Value

an object of class igraph. In addition, it contains the attributes:

`R`	a vector of radii.
`Y`	the y vectors.
`layout`	the x vectors.

In the case of the classifier, the attributes are:

`Rx`, `Ry`	vectors of radii.
`Cx`, `Cy`	the ball centers.

Note

The plotting assumes the cccd used Euclidean distance, and so the balls/circles will be Euclidean balls/circles. If the method used in the distance was some other metric, you'll have to plot the balls/circles yourself if you want them to be correct on the plot.

Author(s)

David J. Marchette, david.marchette@navy.mil

References

D.J. Marchette, "Class Cover Catch Digraphs", Wiley Interdisciplinary Reviews: Computational Statistics, 2, 171-177, 2010.

D.J. Marchette, Random Graphs for Statistical Pattern Recognition, John Wiley & Sons, 2004.

C.E. Priebe, D.J. Marchette, J. DeVinney and D. Socolinsky, "Classification Using Class Cover Catch Digraphs", Journal of Classification, 20, 3-23, 2003.

See Also

`ccd`, `rng`, `gg`, `dist`, `get.knn` `dominate`

Examples

```
set.seed(456330)
z <- matrix(runif(1000),ncol=2)
ind <- which(z[,1]<.5 & z[,2]<.5)
x <- z[ind,]
y <- z[-ind,]
g <- cccd(x,y)
C <- cccd.classifier(x,y)
z2 <- matrix(runif(1000),ncol=2)
ind <- which(z2[,1]<.5 & z2[,2]<.5)
cls <- rep(0,nrow(z2))
cls[ind] <- 1
out <- cccd.classify(z2,C)
sum(out != cls)/nrow(z2)
## Not run:
plot(g,plot.circles=TRUE,dominate.only=TRUE)
points(z2,col=2*(1-cls)+1,pch=20)

## End(Not run)
```

`ccd` *Cluster Catch Digraphs*

Description

construct the cluster catch digraph from a data matrix.

Usage

```
ccd(data, m = 1, alpha = 0.05, sequential = TRUE, method = NULL)
## S3 method for class 'ccd'
plot(x,...)
```

Arguments

`data`	a matrix of observations.
`m`	slope of the null hypothesis curve.
`alpha`	alpha for the K-S test if `sequential=T`.
`sequential`	use the sequential or non-sequential version.
`method`	the method used for the distance. See `dist`.
`x`	an object of class ccd.
`...`	arguments passed to `plot.cccd`.

Details

cluster cover digraph. `plot.ccd` is just a call to `plot.cccd`.

Value

an object of class igraph. In addition, this contains the attributes:

`R`	the radii.
`stats`	the K-S statistics.
`layout`	the data vectors.
`walks`	the y-values of the random walks.
`fs`	the null hypothesis curve.
`A`	the adjacency matrix.
`m`, `alpha`	arguments passed to `ccd`.

Author(s)

David J. Marchette david.marchette@navy.mil

References

D.J. Marchette, Random Graphs for Statistical Pattern Recognition, John Wiley & Sons, 2004.

See Also

cccd

Examples

```
x <- matrix(rnorm(100),ncol=2)
G <- ccd(x)
## Not run:
plot(G)

## End(Not run)
```

dominate *Dominating Sets*

Description

find maximum dominating sets in (di)graphs.

Usage

```
dominate(g, method = "greedy",proportion=1.0)
```

Arguments

`g`	an adjacency matrix.
`method`	one of "greedy","random","byRadius", "greedyProportion".
`proportion`	proportion of points to cover.

Details

`dominate` is the main program which calls the others, as indicated by `method`. Greedy is the greedy dominating algorithm. In the greedy method ties are broken by first index (a la `which.max`). The byRadius method uses the radii to break ties while the random routine breaks ties randomly. If `proportion` is given, the algorithm stops after `proportion` points are covered.

Value

a vector of vertices corresponding to the dominating set.

Author(s)

David J. Marchette david.marchette@navy.mil

References

T.W. Haynes, S.T. Hedetniemi and P.J. Slater, Fundamentals of Domination in Graphs, Marcel Dekker, 1998,

Examples

```
x <- matrix(runif(100),ncol=2)
y <- matrix(runif(100,-2,2),ncol=2)
G <- cccd(x,y)
D <- dominate(G)
## Not run:
plot(G,balls=TRUE,D=D)

## End(Not run)
```

---

gg *Gabriel Graph*

Description

A Gabriel graph is one where the vertices are points and there is an edge between two points if the maximal ball between the points contains no other points.

Usage

```
gg(x, r = 1, method = NULL, usedeldir = TRUE, open = TRUE,
   k = NA, algorithm = 'cover_tree')
```

Arguments

`x`	a matrix of observations.
`r`	a multiplier on the ball radius.
`method`	the method used for the distance. See `dist`
`usedeldir`	logical. Whether to use the deldir package or not.
`open`	logical. If TRUE, open balls are used in the definition.
`k`	If given, `get.knn` is used from FNN to approximate the Gabriel graph. Only the `k` nearest neighbors to the points are used to determine whether an edge should be made or not. This will be much faster and use less memory for large data sets, but is an approximation unless `k` is sufficiently large.
`algorithm`	See `get.knn`.

Details

places an edge between two points i, j if the ball centered between the points with radius $rd(i, j)/2$ contains no other points.

Value

an object of class igraph. In addition it contains the attributes:

`layout`	the data.
`r, p`	arguments passed to gg

Author(s)

David J. Marchette

References

K.R. Gabriel and R.R. Sokal, A New Statistical Approach to Geographic Variation Analysis, Systemic Zoology, 18, 259-278, 1969

D.J. Marchette, Random Graphs for Statistical Pattern Recognition, John Wiley & Sons, 2004.

See Also

`rng`, `dist`, `get.knn`

Examples

```
x <- matrix(runif(100),ncol=2)

g <- gg(x)
## Not run:
plot(g)

## End(Not run)
```

`juggling` *Juggling*

Description

a resampled version of the CCCD classifier.

Usage

```
juggle(data, classes, sampled = TRUE, sample.dim = FALSE,
       num = 100, sample.proportion = 0.1, k = 2, method = NULL)
juggle.classify(data,J,tdata,indices)
```

Arguments

`data, tdata`	training data from which to build the classifier. In the case of `juggle.classify`, `tdata` is the training data and `data` is the test data.
`classes`	class labels.
`sampled`	whether the data are subsampled.
`sample.dim`	if TRUE, the dimensions (variates) are also sampled.
`num`	number of juggles (resamples).
`sample.proportion`	proportion of the data to sample. If 1 or greater, the data are sampled with replacement.
`k`	number of variates to sample when `sample.dim` is TRUE.
`J`	the juggled classifier.
`indices`	the indices of the juggles to use.
`method`	the method used for the distance. See `dist`

Details

The idea of juggling is to sample the data, compute a CCCD classifier, then repeat. The resampling is controled by the two sampling variables, which basically determine whether the data are sampled with replacement, or whether a subsample is used. If `sample.dim` is TRUE, the variates are also sampled, with `k` indicating how many are sampled.

Value

`juggle.classify` returns a matrix holding the classification probabilities for each observation in `data`. a list consisting of:

`S`	the dominating sets.
`R`	the radii.
`dimension`	the dimension of the data.
`vars`	in the case of `sample.dim`=TRUE, the variables sampled each time.

Only the indicies into the training data are stored in `J`, which is why the classifier requires the original training data in `tdata`.

Author(s)

David J. Marchette, david.marchette@navy.mil

See Also

`cccd`, `dist`

nng *Nearest Neighbor Graphs*

Description

nearest neighbor, k-nearest neighbor, and mutual k-nearest neighbor (di)graphs.

Usage

```
nng(x = NULL, dx = NULL, k = 1, mutual = FALSE, method = NULL,
    use.fnn = FALSE, algorithm = 'cover_tree')
```

Arguments

`x`	a data matrix. Either x or dx is required
`dx`	interpoint distance matrix
`k`	number of neighbors
`mutual`	logical. if true the neighbors must be mutual. See details.
`method`	the method used for the distance. See `dist`
`use.fnn`	logical. If TRUE, `get.knn` from the FNN package is used to obtain the neighbors.
`algorithm`	see `get.knn`.

Details

a k-nearest neighbor graph is a digraph where each vertex is associated with an observation and there is a directed edge between the vertex and it's k nearest neighbors. A mutual k-nearest neighbor graph is a graph where there is an edge between x and y if x is one of the k nearest neighbors of y AND y is one of the k nearest neighbors of x.

Value

an object of class igraph with the extra attributes

`layout`	the x vectors.
`k`, `mutual`, `p`	arguments given to `nn`.

Author(s)

David J. Marchette david.marchette@navy.mil

References

D.J. Marchette, Random Graphs for Statistical Pattern Recognition, John Wiley & Sons, 2004.

See Also

dist get.knn

Examples

```
x <- matrix(runif(100),ncol=2)

G1 <- nng(x,k=1)
## Not run:
par(mfrow=c(2,2))
plot(G1)

## End(Not run)

G2 <- nng(x,k=2)
## Not run:
plot(G2)

## End(Not run)

G5 <- nng(x,k=5)
## Not run:
plot(G5)

## End(Not run)

G5m <- nng(x,k=5,mutual=TRUE)
## Not run:
plot(G5m)
par(mfrow=c(1,1))

## End(Not run)
```

---

prune *Prune Points*

Description

a nearest neighbor pruning using neighborhood graphs.

Usage

```
prune(x, classes, prox = "Gabriel", ignore.ties = TRUE, ...)
```

Arguments

x	a data matrix.
classes	a vector of class labels.

`prox`	type of proximity graph.
`ignore.ties`	do not prune if there is a tie vote.
`...`	arguments passed to the proximity graph.

Details

First a proximity graph is computed on the data. Then points are marked if their neighbors have a different class than they do: if the most common class among the neighbors is different than the point. Then all marked points are removed.

Value

A list with attributes:

`x`	the pruned data.
`v`	the indices of the retained data.
`g`	the proximity graph.

Author(s)

David J. Marchette, david.marchette@navy.mil

References

http://www.bic.mni.mcgill.ca/users/crisco/pgedit/

`rng` *Relative Neighborhood Graph.*

Description

the relative neighborhood graph defined by a set of points.

Usage

```
rng(x=NULL, dx=NULL, r = 1, method = NULL, usedeldir = TRUE, open = TRUE, k = NA,
    algorithm = 'cover_tree')
```

Arguments

`x`	a data matrix. Either `x` or `dx` must be provided.
`dx`	an interpoint distance matrix.
`r`	a multiplier to grow the balls.
`method`	the method used for the distance. See `dist`

`usedeldir`	a logical. If true and the data are two dimensional and the deldir package is installed, the Delaunay triangularization is first computed, and this is used to compute the relative neighborhood graph.
`open`	logical. If TRUE, open balls are used in the definition.
`k`	If given, `get.knn` is used from FNN to approximate the relative neighborhood graph. Only the `k` nearest neighbors to the points are used to determine whether an edge should be made or not. This will be much faster and use less memory for large data sets, but is an approximation unless `k` is sufficiently large.
`algorithm`	See `get.knn`.

Details

the relative neighborhood graph is defined in terms of balls centered at observations. For two observations, the balls are set to have radius equal to the distance between the observations (or `r` times this distance if `r` is not 1). There is an edge between the vertices associated with the observations if and only if there are no vertices in the lune defined by the intersection of the balls.

The flag `open` should make no difference for most applications, but there are very specific cases (see the example section below) where setting it to be TRUE will give the wrong answer (thanks to Luke Mathieson for pointing this out to me).

Value

an object of class igraph, with the additional attributes

`layout`	the x matrix.
`r, p`	arguments given to `rng`.

Author(s)

David J. Marchette david.marchette@navy.mil

References

J.W. Jaromczyk and G.T. Toussaint, "Relative neighborhood graphs and their relatives", Proceedings of the IEEE, 80, 1502-1517, 1992.

G.T. Toussaint, "A Graph-Theoretic Primal Sketch", Computational Morphology, 229-260, 1988.

D.J. Marchette, Random Graphs for Statistical Pattern Recognition, John Wiley & Sons, 2004.

See Also

`gg`,`cccd`,`ccd`, `dist` `get.knn`

Examples

```
x <- matrix(runif(100),ncol=2)

g <- rng(x)
## Not run:
```

```
plot(g)

## End(Not run)

## Example using 'open':
g <- graph.full(5,directed=FALSE)

g1 <- rng(x=get.adjacency(g,sparse=FALSE),open=TRUE)
ecount(g1)
g2 <- rng(x=get.adjacency(g,sparse=FALSE),open=FALSE)
graph.isomorphic(g2,g)
```

Index