Package 'KPC'

October 12, 2022

Type Package

Title Kernel Partial Correlation Coefficient

Version 0.1.2

Maintainer Zhen Huang <zh2395@columbia.edu>

Description Implementations of two empirical versions the kernel partial correlation (KPC) coefficient and the associated variable selection algorithms. KPC is a measure of the strength of conditional association between Y and Z given X, with X, Y, Z being random variables taking values in general topological spaces. As the name suggests, KPC is defined in terms of kernels on reproducing kernel Hilbert spaces (RKHSs). The population KPC is a deterministic number between 0 and 1; it is 0 if and only if Y is conditionally independent of Z given X, and it is 1 if and only if Y is a measurable function of Z and X. One empirical KPC estimator is based on geometric graphs, such as K-nearest neighbor graphs and minimum spanning trees, and is consistent under very weak conditions. The other empirical estimator, defined using conditional mean embeddings (CMEs) as used in the RKHS literature, is also consistent under suitable conditions. Using KPC, a stepwise forward variable selection algorithm KFOCI (using the graph based estimator of KPC) is provided, as well as a similar stepwise forward selection algorithm based on the RKHS based estimator. For more details on KPC, its empirical estimators and its application on variable selection, see Huang, Z., N. Deb, and B. Sen (2022). "Kernel partial correlation coefficient – a measure of conditional dependence" (URL listed below). When X is empty, KPC measures the unconditional dependence between Y and Z, which has been described in Deb, N., P. Ghosal, and B. Sen (2020), "Measuring association on topological spaces using kernels and geometric graphs" <arXiv:2010.01768>, and it is implemented in the functions KMAc() and Klin() in this package. The latter can be computed in near linear time.

License GPL-3

Encoding UTF-8

URL https://www.jmlr.org/papers/v23/21-493.html,
https://arxiv.org/abs/2012.14804

LazyData true

RoxygenNote 7.1.2

Depends R (>= 4.0.0), data.table, kernlab

Imports RANN, proxy, parallel, mlpack

NeedsCompilation no

Author Zhen Huang [aut, cre],
Nabarun Deb [ctb],
Bodhisattva Sen [ctb]

Repository CRAN

Date/Publication 2022-10-05 20:20:03 UTC

R topics documented:

`ElecData` *2017 Korea presidential election data*

Description

A dataset containing 9 variables, consists of the voting results earned by the top five candidates from 250 electoral districts in Korea.

Usage

```
ElecData
```

Format

A data frame with 1250 rows and 9 variables:

PrecinctCode 250 precinct codes designated by the election committee (4 digits)

CityCode 250 city codes of administrative standard code management system (5 digits)

CandidateName Symbols 1-5, corresponding to Moon Jae-in, Hong Jun-pyo, Ahn Cheol-soo, Yoo Seung-min, Shim Sang-jung

AveAge Average age of voters in 17 years: statistics on resident registration population of the Ministry of Government Administration and Home Affairs

AveYearEdu Average number of years of education for voters

AveHousePrice Average price per square meter in 17 years

AveInsurance The average insurance premium for each city, county, district

VoteRate Vote rate by candidate

NumVote Number of votes by candidate

Source

https://github.com/OhmyNews/2017-Election

`KFOCI` *Kernel Feature Ordering by Conditional Independence*

Description

Variable selection with KPC using directed K-NN graph or minimum spanning tree (MST)

Usage

```
KFOCI(
  Y,
  X,
  k = kernlab::rbfdot(1/(2 * stats::median(stats::dist(Y))^2)),
  Knn = min(ceiling(NROW(Y)/20), 20),
  num_features = NULL,
  stop = TRUE,
  numCores = parallel::detectCores(),
  verbose = FALSE
)
```

Arguments

`Y`	a matrix of responses (n by dy)
`X`	a matrix of predictors (n by dx)
`k`	a function $k(y, y')$ of class `kernel`. It can be the kernel implemented in `kernlab` e.g., Gaussian kernel: `rbfdot(sigma = 1)`, linear kernel: `vanilladot()`.
`Knn`	a positive integer indicating the number of nearest neighbor; or "MST". The suggested choice of Knn is 0.05n for samples up to a few hundred observations. For large n, the suggested Knn is sublinear in n. That is, it may grow slower than any linear function of n. The computing time is approximately linear in Knn. A smaller Knn takes less time.
`num_features`	the number of variables to be selected, cannot be larger than dx. The default value is NULL and in that case it will be set equal to dx. If `stop == TRUE` (see below), then num_features is the maximal number of variables to be selected.
`stop`	If `stop == TRUE`, then the automatic stopping criterion (stops at the first instance of negative Tn, as mentioned in the paper) will be implemented and continued till `num_features` many variables are selected. If `stop == FALSE` then exactly `num_features` many variables are selected.
`numCores`	number of cores that are going to be used for parallelizing the process.
`verbose`	whether to print each selected variables during the forward stepwise algorithm

Details

A stepwise forward selection of variables using KPC. At each step it selects the X_j that maximizes $\hat{\rho^2}(Y, X_j|\text{selected } X_i)$. It is suggested to normalize the predictors before applying KFOCI. Euclidean distance is used for computing the K-NN graph and the MST.

Value

The algorithm returns a vector of the indices from 1,...,dx of the selected variables in the same order that they were selected. The variables at the front are expected to be more informative in predicting Y.

See Also

KPCgraph, KPCRKHS, KPCRKHS_VS

Examples

```
n = 200
p = 10
X = matrix(rnorm(n * p), ncol = p)
Y = X[, 1] * X[, 2] + sin(X[, 1] * X[, 3])
KFOCI(Y, X, kernlab::rbfdot(1), Knn=1, numCores=1)
# 1 2 3
```

Klin *A near linear time analogue of KMAc*

Description

Calculate $\hat{\eta}_n^{\text{lin}}$ (the unconditional version of graph-based KPC) using directed K-NN graph or minimum spanning tree (MST). The computational complexity is O(nlog(n))

Usage

```
Klin(
  Y,
  X,
  k = kernlab::rbfdot(1/(2 * stats::median(stats::dist(Y))^2)),
  Knn = 1
)
```

Arguments

Y	a matrix of response (n by dy)
X	a matrix of predictors (n by dx)
k	a function $k(y, y')$ of class `kernel`. It can be the kernel implemented in `kernlab` e.g. `rbfdot(sigma = 1)`, `vanilladot()`
Knn	the number of K-nearest neighbor to use; or "MST". A small Knn (e.g., Knn=1) is recommended.

Details

$\hat{\eta}_n$ is an estimate of the population kernel measure of association, based on data $\{(X_i, Y_i)\}_{i=1}^n$ from μ. For K-NN graph, $\hat{\eta}_n$ can be computed in near linear time (in n). In particular,

$$\hat{\eta}_n^{\text{lin}} := \frac{n^{-1}\sum_{i=1}^n d_i^{-1}\sum_{j:(i,j)\in\mathcal{E}(G_n)} k(Y_i, Y_j) - (n-1)^{-1}\sum_{i=1}^{n-1} k(Y_i, Y_{i+1})}{n^{-1}\sum_{i=1}^n k(Y_i, Y_i) - (n-1)^{-1}\sum_{i=1}^{n-1} k(Y_i, Y_{i+1})}$$

, where all symbols have their usual meanings as in the definition of $\hat{\eta}_n$. Euclidean distance is used for computing the K-NN graph and the MST.

Value

The algorithm returns a real number 'Klin': an empirical kernel measure of association which can be computed in near linear time when K-NN graphs are used.

References

Deb, N., P. Ghosal, and B. Sen (2020), "Measuring association on topological spaces using kernels and geometric graphs" <arXiv:2010.01768>.

See Also

KPCgraph, KMAc

Examples

```
library(kernlab)
Klin(Y = rnorm(100), X = rnorm(100), k = rbfdot(1), Knn = 1)
```

`KMAc` *KMAc (the unconditional version of graph-based KPC) with geometric graphs.*

Description

Calculate $\hat{\eta}_n$ (the unconditional version of graph-based KPC) using directed K-NN graph or minimum spanning tree (MST).

Usage

```
KMAc(
  Y,
  X,
  k = kernlab::rbfdot(1/(2 * stats::median(stats::dist(Y))^2)),
  Knn = 1
)
```

Arguments

`Y`	a matrix of response (n by dy)
`X`	a matrix of predictors (n by dx)
`k`	a function $k(y, y')$ of class `kernel`. It can be the kernel implemented in `kernlab` e.g., Gaussian kernel: `rbfdot(sigma = 1)`, linear kernel: `vanilladot()`
`Knn`	the number of K-nearest neighbor to use; or "MST". A small Knn (e.g., Knn=1) is recommended for an accurate estimate of the population KMAc.

Details

$\hat{\eta}_n$ is an estimate of the population kernel measure of association, based on data $\{(X_i, Y_i)\}_{i=1}^n$ from μ. For K-NN graph, ties will be broken at random. MST is found using package `emstreeR`. In particular,

$$\hat{\eta}_n := \frac{n^{-1}\sum_{i=1}^n d_i^{-1}\sum_{j:(i,j)\in\mathcal{E}(G_n)} k(Y_i, Y_j) - (n(n-1))^{-1}\sum_{i\neq j} k(Y_i, Y_j)}{n^{-1}\sum_{i=1}^n k(Y_i, Y_i) - (n(n-1))^{-1}\sum_{i\neq j} k(Y_i, Y_j)},$$

where G_n denotes a MST or K-NN graph on $X_1, \ldots, X_n$, $\mathcal{E}(G_n)$ denotes the set of edges of G_n and $(i, j) \in \mathcal{E}(G_n)$ implies that there is an edge from X_i to X_j in G_n. Euclidean distance is used for computing the K-NN graph and the MST.

Value

The algorithm returns a real number ‘KMAc’, the empirical kernel measure of association

References

Deb, N., P. Ghosal, and B. Sen (2020), “Measuring association on topological spaces using kernels and geometric graphs” <arXiv:2010.01768>.

See Also

`KPCgraph`, `Klin`

Examples

```
library(kernlab)
KMAc(Y = rnorm(100), X = rnorm(100), k = rbfdot(1), Knn = 1)
```

KPCgraph — *Kernel partial correlation with geometric graphs*

Description

Calculate the kernel partial correlation (KPC) coefficient with directed K-nearest neighbor (K-NN) graph or minimum spanning tree (MST).

Usage

```
KPCgraph(
  Y,
  X,
  Z,
  k = kernlab::rbfdot(1/(2 * stats::median(stats::dist(Y))^2)),
  Knn = 1,
  trans_inv = FALSE
)
```

Arguments

`Y`	a matrix (n by dy)
`X`	a matrix (n by dx) or `NULL` if X is empty
`Z`	a matrix (n by dz)
`k`	a function $k(y, y')$ of class `kernel`. It can be the kernel implemented in `kernlab` e.g., Gaussian kernel: `rbfdot(sigma = 1)`, linear kernel: `vanilladot()`.
`Knn`	a positive integer indicating the number of nearest neighbor to use; or "MST". A small Knn (e.g., Knn=1) is recommended for an accurate estimate of the population KPC.
`trans_inv`	TRUE or FALSE. Is $k(y, y)$ free of y?

Details

The kernel partial correlation squared (KPC) measures the conditional dependence between Y and Z given X, based on an i.i.d. sample of (Y, Z, X). It converges to the population quantity (depending on the kernel) which is between 0 and 1. A small value indicates low conditional dependence between Y and Z given X, and a large value indicates stronger conditional dependence. If `X == NULL`, it returns the `KMAc(Y,Z,k,Knn)`, which measures the unconditional dependence between Y and Z. Euclidean distance is used for computing the K-NN graph and the MST. MST in practice often achieves similar performance as the 2-NN graph. A small K is recommended for the K-NN graph for an accurate estimate of the population KPC, while if KPC is used as a test statistic for conditional independence, a larger K can be beneficial.

Value

The algorithm returns a real number which is the estimated KPC.

See Also

KPCRKHS, KMAc, Klin

Examples

```
library(kernlab)
n = 2000
x = rnorm(n)
z = rnorm(n)
y = x + z + rnorm(n,1,1)
KPCgraph(y,x,z,vanilladot(),Knn=1,trans_inv=FALSE)

n = 1000
x = runif(n)
z = runif(n)
y = (x + z) %% 1
KPCgraph(y,x,z,rbfdot(5),Knn="MST",trans_inv=TRUE)

discrete_ker = function(y1,y2) {
    if (y1 == y2) return(1)
    return(0)
}
class(discrete_ker) <- "kernel"
set.seed(1)
n = 2000
x = rnorm(n)
z = rnorm(n)
y = rep(0,n)
for (i in 1:n) y[i] = sample(c(1,0),1,prob = c(exp(-z[i]^2/2),1-exp(-z[i]^2/2)))
KPCgraph(y,x,z,discrete_ker,1)
##0.330413
```

KPCRKHS *Kernel partial correlation with RKHS method*

Description

Compute estimate of Kernel partial correlation (KPC) coefficient using conditional mean embeddings in the reproducing kernel Hilbert spaces (RKHS).

Usage

```
KPCRKHS(
  Y,
  X = NULL,
  Z,
  ky = kernlab::rbfdot(1/(2 * stats::median(stats::dist(Y))^2)),
  kx = kernlab::rbfdot(1/(2 * stats::median(stats::dist(X))^2)),
  kxz = kernlab::rbfdot(1/(2 * stats::median(stats::dist(cbind(X, Z)))^2)),
  eps = 0.001,
  appro = FALSE,
  tol = 1e-05
)
```

Arguments

`Y`	a matrix (n by dy)
`X`	a matrix (n by dx) or `NULL` if X is empty
`Z`	a matrix (n by dz)
`ky`	a function $k(y, y')$ of class `kernel`. It can be the kernel implemented in `kernlab` e.g., Gaussian kernel: `rbfdot(sigma = 1)`, linear kernel: `vanilladot()`.
`kx`	the kernel function for X
`kxz`	the kernel function for (X, Z) or for Z if X is empty
`eps`	a small positive regularization parameter for inverting the empirical cross-covariance operator
`appro`	whether to use incomplete Cholesky decomposition for approximation
`tol`	tolerance used for incomplete Cholesky decomposition (implemented by the function `inchol` in the package `kernlab`)

Details

The kernel partial correlation (KPC) coefficient measures the conditional dependence between Y and Z given X, based on an i.i.d. sample of (Y, Z, X). It converges to the population quantity (depending on the kernel) which is between 0 and 1. A small value indicates low conditional dependence between Y and Z given X, and a large value indicates stronger conditional dependence. If `X = NULL`, it measures the unconditional dependence between Y and Z.

Value

The algorithm returns a real number which is the estimated KPC.

See Also

`KPCgraph`

Examples

```
n = 500
set.seed(1)
x = rnorm(n)
z = rnorm(n)
y = x + z + rnorm(n,1,1)
library(kernlab)
k = vanilladot()
KPCRKHS(y, x, z, k, k, k, 1e-3/n^(0.4), appro = FALSE)
# 0.4854383 (Population quantity = 0.5)
KPCRKHS(y, x, z, k, k, k, 1e-3/n^(0.4), appro = TRUE, tol = 1e-5)
# 0.4854383 (Population quantity = 0.5)
```

KPCRKHS_VS *Variable selection with RKHS estimator*

Description

The algorithm performs a forward stepwise variable selection using RKHS estimators.

Usage

```
KPCRKHS_VS(
  Y,
  X,
  num_features,
  ky = kernlab::rbfdot(1/(2 * stats::median(stats::dist(Y))^2)),
  kS = NULL,
  eps = 0.001,
  appro = FALSE,
  tol = 1e-05,
  numCores = parallel::detectCores(),
  verbose = FALSE
)
```

Arguments

`Y`	a matrix of responses (n by dy)
`X`	a matrix of predictors (n by dx)
`num_features`	the number of variables to be selected, cannot be larger than dx.
`ky`	a function $k(y, y')$ of class `kernel`. It can be the kernel implemented in `kernlab` e.g., Gaussian kernel: `rbfdot(sigma = 1)`, linear kernel: `vanilladot()`
`kS`	a function that takes X and a subset of indices S as inputs, and then outputs the kernel for X_S. The first argument of kS is X, and the second argument is a vector of positive integer. If `kS == NULL`, Gaussian kernel with empitical bandwidth `kernlab::rbfdot(1/(2*stats::median(stats::dist(X[,S]))^2))` will be used.
`eps`	a positive number; the regularization parameter for the RKHS estimator
`appro`	whether to use incomplete Cholesky decomposition for approximation
`tol`	tolerance used for incomplete Cholesky decomposition (`inchol` in package `kernlab`)
`numCores`	number of cores that are going to be used for parallelizing the process.
`verbose`	whether to print each selected variables during the forward stepwise algorithm

Details

A stepwise forward selection of variables using KPC. At each step it selects the X_j that maximizes $\tilde{\rho^2}(Y, X_j|\text{selected } X_i)$. It is suggested to normalize the features before applying the algorithm.

Value

The algorithm returns a vector of the indices from `1,...,dx` of the selected variables in the same order that they were selected. The variables at the front are expected to be more informative in predicting Y.

See Also

`KPCgraph`, `KPCRKHS`, `KFOCI`

Examples

```
n = 200
p = 10
X = matrix(rnorm(n * p), ncol = p)
Y = X[, 1] * X[, 2] + sin(X[, 1] * X[, 3])
library(kernlab)
kS = function(X,S) return(rbfdot(1/length(S)))
KPCRKHS_VS(Y, X, num_features = 3, rbfdot(1), kS, eps = 1e-3, appro = FALSE, numCores = 1)
kS = function(X,S) return(rbfdot(1/(2*stats::median(stats::dist(X[,S]))^2)))
KPCRKHS_VS(Y, X, num_features = 3, rbfdot(1), kS, eps = 1e-3, appro = FALSE, numCores = 1)
```

`med` *Medical data from 35 patients*

Description

A dataset containing three variables (creatinine clearance C; digoxin clearance D; urine flow U) from 35 patients.

Usage

```
med
```

Format

A data frame with 35 rows and 3 variables:

C creatinine clearance, in ml/min/1.73m^2

D digoxin clearance, in ml/min/1.73m^2

U urine flow, in ml/min

Source

Edwards, D. (2012). Introduction to graphical modelling, Section 3.1.4, Springer Science & Business Media.

TnKnn *Tn with geometric graphs*

Description

Calculate T_n using directed K-NN graph or minimum spanning tree (MST).

Usage

```
TnKnn(Y, X, k, Knn = 1)
```

Arguments

`Y`	a matrix of response (n by dy)
`X`	a matrix of predictors (n by dx)
`k`	a function $k(y, y')$ of class `kernel`. It can be the kernel implemented in `kernlab` e.g. Gaussian kernel: `rbfdot(sigma = 1)`, linear kernel: `vanilladot()`.
`Knn`	the number of K-nearest neighbor to use; or "MST".

Details

T_n is an estimate of $E[E[k(Y_1, Y_1')|X]]$, with Y_1, Y_1' drawn iid from $Y|X$, given X. For K-NN graph, ties will be broken at random. Algorithm finding the MST is implemented the package `emstreeR`.

Value

The algorithm returns a real number which is the value of Tn.

Index