Package ‘FNN’

December 10, 2018

Version 1.1.2.2
Date 2018-12-10
Title Fast Nearest Neighbor Search Algorithms and Applications
Author Alina Beygelzimer, Sham Kakadet and John Langford (cover tree library),
        Sunil Arya and David Mount (ANN library 1.1.2 for the kd-tree approach), Shengqiao Li
Copyright ANN Copyright (c) 1997-2010 University of Maryland and Sunil
        Arya and David Mount. All Rights Reserved.
Maintainer ORPHANED
Depends R (>= 3.0.0)
Suggests chemometrics, mvtnorm
Description Cover-tree and kd-tree fast k-nearest neighbor search algorithms and related applications
        including KNN classification, regression and information measures are implemented.
License GPL (>= 2)
NeedsCompilation yes
Repository CRAN
Date/Publication 2018-12-10 17:53:53 UTC
X-CRAN-Original-Maintainer Shengqiao Li <lishengqiao@yahoo.com>
X-CRAN-Comment Orphaned on 2018-08-10 as C++ issues were not corrected
        since 2017 despite reminders.

R topics documented:
crossentropy .................................................. 2
entropy .......................................................... 3
get.knn .......................................................... 4
KL.dist .......................................................... 5
KL.divergence ................................................. 6
knn .............................................................. 8
knn.cv .......................................................... 9
knn.dist ........................................................ 10
knn.index ...................................................... 11
crossentropy

Description
KNN Cross Entropy Estimators.

Usage

crossentropy(X, Y, k=10, algorithm=c("kd_tree", "cover_tree", "brute"))

Arguments

X an input data matrix.
Y an input data matrix.
k the maximum number of nearest neighbors to search. The default value is set to 10.
algorithm nearest neighbor search algorithm.

Details

If \( p(x) \) and \( q(x) \) are two continuous probability density functions, then the cross-entropy of \( p \) and \( q \) is defined as \( H(p, q) = E_p[- \log q(x)] \).

Value

a vector of length \( k \) for crossentropy estimates using 1:k nearest neighbors, respectively.

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com.> 

References

entropy

Shannon Entropy

Description

KNN Shannon Entropy Estimators.

Usage

entropy(X, k = 10, algorithm = c("kd_tree", "brute"))

Arguments

x an input data matrix.

k the maximum number of nearest neighbors to search. The default value is set to 10.

algorithm nearest neighbor search algorithm.

Value

a vector of length k for entropy estimates using 1:k nearest neighbors, respectively.

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com.>

References


get.knn

Search Nearest Neighbors

Description

Fast k-nearest neighbor searching algorithms including a kd-tree, cover-tree and the algorithm implemented in class package.

Usage

get.knn(data, k=10, algorithm=c("kd_tree", "cover_tree", "CR", "brute"))
get.knnx(data, query, k=10, algorithm=c("kd_tree", "cover_tree", "CR", "brute"))

Arguments

data an input data matrix.
query a query data matrix.
algorithm nearest neighbor searching algorithm.
k the maximum number of nearest neighbors to search. The default value is set to 10.

Details

The cover tree is O(n) space data structure which allows us to answer queries in the same O(log(n)) time as kd tree given a fixed intrinsic dimensionality. Templated code from http://hunch.net/~jl/projects/cover_tree/cover_tree.html is used.

The kd tree algorithm is implemented in the Approximate Near Neighbor (ANN) C++ library (see http://www.cs.umd.edu/~mount/ANN/). The exact nearest neighbors are searched in this package.

The CR algorithm is the VR using distance $1-x'y$ assuming $x$ and $y$ are unit vectors. The brute algorithm searches linearly. It is a naive method.

Value

a list contains:

nn.index an n x k matrix for the nearest neighbor indice.

nn.dist an n x k matrix for the nearest neighbor Euclidean distances.

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com>
References


See Also

`nnR` in `RANN`, `ann` in `yaImpute` and `knn` in `class`.

Examples

```r
data<- query<- cbind(1:10, 1:10)

get.knn(data, k=5)
get.knnx(data, query, k=5)
get.knnx(data, query, k=5, algo="kd_tree")

th<-- runif(10, min=0, max=2*pi)
data2<- cbind(cos(th), sin(th))
get.knn(data2, k=5, algo="CR")
```

---

**KL.dist**

*Kullback-Leibler Divergence*

**Description**

Compute Kullback-Leibler symmetric distance.

**Usage**

```r
KL.dist(X, Y, k = 10, algorithm=c("kd_tree", "cover_tree", "brute"))
KLx.dist(X, Y, k = 10, algorithm="kd_tree")
```

**Arguments**

- **X** An input data matrix.
- **Y** An input data matrix.
- **k** The maximum number of nearest neighbors to search. The default value is set to 10.
- **algorithm** nearest neighbor search algorithm.
Details

Kullback-Leibler distance is the sum of divergence \( q(x) \) from \( p(x) \) and \( p(x) \) from \( q(x) \).

\( KL.* \) versions return distances from \( C \) code to \( \mathbb{R} \) but \( KL.x.* \) do not.

Value

Return the Kullback-Leibler distance between \( X \) and \( Y \).

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com.>

References


See Also

\( KL.divergence \).

Examples

```r
set.seed(1)
x<- rexp(10000, rate=0.2)
y<- rexp(10000, rate=0.4)

KL.dist(x, y, k=5)
KLx.dist(x, y, k=5)
#theoretical distance = (0.2-0.4)^2/(0.2*0.4) = 0.5
```

---

**KL.divergence:**

Kullback-Leibler Divergence

Description

Compute Kullback-Leibler divergence.

Usage

```r
KL.divergence(X, Y, k = 10, algorithm=c("kd_tree", "cover_tree", "brute"))
KLx.divergence(X, Y, k = 10, algorithm="kd_tree")
```
**KL.divergence**

**Arguments**

- **X** An input data matrix.
- **Y** An input data matrix.
- **k** The maximum number of nearest neighbors to search. The default value is set to 10.
- **algorithm** nearest neighbor search algorithm.

**Details**

If \( p(x) \) and \( q(x) \) are two continuous probability density functions, then the Kullback-Leibler divergence of \( q \) from \( p \) is defined as \( E_p[\log \frac{p(x)}{q(x)}] \).

\( \text{KL.} \ast \) versions return divergences from C code to R but \( \text{KLx.} \ast \) do not.

**Value**

Return the Kullback-Leibler divergence from \( X \) to \( Y \).

**Author(s)**

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com.>

**References**


**See Also**

- KL.dist

**Examples**

```r
set.seed(1000)
X <- rexp(10000, rate=0.2)
Y <- rexp(10000, rate=0.4)

KL.divergence(X, Y, k=5)
#theoretical divergence = log(0.2/0.4)+(0.4-0.2)-1 = 1-log(2) = 0.307
```
**knn**

### k-Nearest Neighbour Classification

**Description**

k-nearest neighbour classification for test set from training set. For each row of the test set, the k nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

**Usage**

```r
knn(train, test, cl, k = 1, prob = FALSE, algorithm=c("kd_tree", "cover_tree", "brute"))
```

**Arguments**

- `train`: matrix or data frame of training set cases.
- `test`: matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
- `cl`: factor of true classifications of training set.
- `k`: number of neighbours considered.
- `prob`: if this is true, the proportion of the votes for the winning class are returned as attribute `prob`.
- `algorithm`: nearest neighbor search algorithm.

**Value**

factor of classifications of test set. `doubt` will be returned as `NA`.

**Author(s)**

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com.>

**References**


**See Also**

`ownn`, `knn.cv` and `knn` in `class`.
Examples

```r
data(iris3)
train <- rbind(iris3[1:25,1], iris3[1:25,2], iris3[1:25,3])
test <- rbind(iris3[26:50,1], iris3[26:50,2], iris3[26:50,3])
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
knn(train, test, cl, k = 3, prob=TRUE)
attributes(.Last.value)
```

---

**knn.cv**

**k-Nearest Neighbour Classification Cross-Validation**

**Description**

k-nearest neighbour classification cross-validation from training set.

**Usage**

```r
knn.cv(train, cl, k = 1, prob = FALSE, algorithm=c("kd_tree", 
"cover_tree", "brute"))
```

**Arguments**

- `train` matrix or data frame of training set cases.
- `cl` factor of true classifications of training set
- `k` number of neighbours considered.
- `prob` if this is true, the proportion of the votes for the winning class are returned as attribute prob.
- `algorithm` nearest neighbor search algorithm.

**Details**

This uses leave-one-out cross validation. For each row of the training set `train`, the k nearest (in Euclidean distance) other training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the kth nearest vector, all candidates are included in the vote.

**Value**

factor of classifications of training set. doubt will be returned as NA. distances and indice of k nearest neighbors are also returned as attributes.

**Author(s)**

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com>
References


See Also

`knn` and `knn.cv` in `class`.

Examples

```r
data(iris3)
train <- rbind(iris3[,1], iris3[,2], iris3[,3])
c1 <- factor(c(rep("s",50), rep("c",50), rep("v",50)))
knn.cv(train, cl, k = 3, prob = TRUE)
attributes(.Last.value)
```

Description

Fast k-nearest neighbor distance searching algorithms.

Usage

```r
knn.dist(data, k=10, algorithm=c("kd_tree", "cover_tree", "CR", "brute"))
knnx.dist(data, query, k=10, algorithm=c("kd_tree", "cover_tree", "CR", "brute"))
```

Arguments

- `data` : an input data matrix.
- `query` : a query data matrix.
- `algorithm` : nearest neighbor searching algorithm.
- `k` : the maximum number of nearest neighbors to search. The default value is set to 10.

Value

return the Euclidean distances of k nearest neighbors.

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com>
References


See Also

get.knn and knn.index.

Examples

```r
if(require(mvtnorm))
{
  sigma<- function(v, r, p)
  {
    V<- matrix(r^2, ncol=p, nrow=p)
    diag(V)<- 1
    V*V
  }

  X<- rmvnorm(1000, mean=rep(0, 20), sigma(1, .5, 20))
  print(system.time(knn.dist(X)))
  print(system.time(knn.dist(X, algorithm = "kd_tree")))
}
```

---

**knn.index**  
**Search Nearest Neighbors**

**Description**

Fast k-nearest neighbor searching algorithms including a kd-tree, cover-tree and the algorithm implemented in class package.

**Usage**

```r
knn.index(data, k=10, algorithm=c("kd_tree", "cover_tree", "CR", "brute"))
knnx.index(data, query, k=10, algorithm=c("kd_tree", "cover_tree", "CR", "brute"))
```
Arguments

data: an input data matrix.
query: a query data matrix.
algorithm: nearest neighbor searching algorithm.
k: the maximum number of nearest neighbors to search. The default value is set to 10.

Value

return the indice of k nearest neighbors.

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com>.

References


See Also

knn.dist and get.knn.

Examples

data<- query<- cbind(1:10, 1:10)

knn.index(data, k=5)
knnx.index(data, query, k=5)
knnx.index(data, query, k=5, algo="kd_tree")
**knn.reg**

---

**k Nearest Neighbor Regression**

### Description

k-nearest neighbor regression

### Usage

```r
knn.reg(train, test = NULL, y, k = 3, algorithm=c("kd_tree", "cover_tree", "brute"))
```

### Arguments

- **train**: matrix or data frame of training set cases.
- **test**: matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case. If not supplied, cross-validation will be done.
- **y**: response of each observation in the training set.
- **k**: number of neighbours considered.
- **algorithm**: nearest neighbor search algorithm.

### Details

If test is not supplied, Leave one out cross-validation is performed and \textit{R-square} is the predicted R-square.

### Value

\texttt{knn.reg} returns an object of \texttt{class} "knnReg" or "knnRegCV" if test data is not supplied. The returned object is a list containing at least the following components:

- **call**: the match call.
- **k**: number of neighbours considered.
- **n**: number of predicted values, either equals test size or train size.
- **pred**: a vector of predicted values.
- **residuals**: predicted residuals. \texttt{NULL} if test is supplied.
- **PRESS**: the sums of squares of the predicted residuals. \texttt{NULL} if test is supplied.
- **R2Pred**: predicted R-square. \texttt{NULL} if test is supplied.

### Note

The code for “VR” nearest neighbor searching is taken from \texttt{class} source.
Author(s)
Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com.>

See Also
knn.

Examples
if(require(chemometrics)){
data(PAC);
pac.knn<- knn.reg(PAC$x, y=PAC$y, k=3);

plot(PAC$y, pac.knn$pred, xlab="y", ylab=expression(hat(y)))
}

mutinfo

Description
KNN Mutual Information Estimators.

Usage
mutinfo(X, Y, k=10, direct=TRUE)

Arguments
X an input data matrix.
Y an input data matrix.
k the maximum number of nearest neighbors to search. The default value is set to 10.
direct Directly compute or via entropies.

Details
The direct computation is based on the first estimator of A. Kraskov, H. Stogbauer and P.Grassberger (2004) and the indirect computation is done via entropy estimates, i.e., I(X, Y) = H(X) + H(Y) - H(X, Y). The direct method has smaller bias and variance but the indirect method is faster, see Evans (2008).

Value
For direct method, one mutual information estimate; For indirect method, a vector of length k for mutual information estimates using 1:k nearest neighbors, respectively.
Optimal Weighted Nearest Neighbor Classification

Description

This function implements Samworth’s optimal weighting scheme for $k$ nearest neighbor classification. The performance improvement is greatest when the dimension is 4 as reported in the reference.

Usage

```r
ownn(train, test, cl, testcl=NULL, k = NULL, prob = FALSE, algorithm=c("kd_tree", "cover_tree", "brute"))
```

Arguments

- `train`: matrix or data frame of training set cases.
- `test`: matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
- `cl`: factor of true classifications of training set.
- `testcl`: factor of true classifications of testing set for error rate calculation.
- `k`: number of neighbours considered, chosen by 5-fold cross-validation if not supplied.
- `prob`: if this is true, the proportion of the weights for the winning class are returned as attribute `prob`.
- `algorithm`: nearest neighbor search algorithm.

Value

A list includes `$k$`, predictions by ordinary knn, optimal weighted knn and bagged knn, and accuracies if class labels of test data set are given.

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com.>
References


See Also

`knn` and `knn` in `class`.

Examples

data(irisS)
train <- rbind(irisS[1:25,1], irisS[1:25,2], irisS[1:25,3])
test <- rbind(irisS[26:50,1], irisS[26:50,2], irisS[26:50,3])
c1 <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
testcl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
out <- ownn(train, test, cl, testcl)
out

print.knnReg

*Print Method for KNN Regression*

Description

Print method for KNN regression.

Usage

```r
## S3 method for class 'knnReg'
print(x, ...)
## S3 method for class 'knnRegCV'
print(x, ...)
```

Arguments

- `x` a knnReg or knnRegCV object.
- `...` Additional print arguments.

Author(s)

Shengqiao Li. To report any bugs or suggestions please email: <lishengqiao@yahoo.com>
Index

*Topic **classif**
  knn, 8
  knn.cv, 9
  ownn, 15

*Topic **manip**
  crossentropy, 2
  entropy, 3
  get.knn, 4
  KL.dist, 5
  KL.divergence, 6
  knn.dist, 10
  knn.index, 11
  mutinfo, 14

*Topic **nonparametric**
  knn, 8
  knn.cv, 9
  knn.reg, 13
  ownn, 15

*Topic **print**
  print.knnReg, 16

*Topic **regression**
  knn.reg, 13

crossentropy, 2

entropy, 3

get.knn, 4, 11, 12
get.knnx (get.knn), 4

KL.dist, 5, 7
KL.divergence, 6, 6
KLx.dist (KL.dist), 5
KLx.divergence (KL.divergence), 6
knn, 5, 8, 8, 10, 14, 16
knn.cv, 8, 9, 10
knn.dist, 10, 12
knn.index, 11, 11
knn.reg, 13
knnx.dist (knn.dist), 10

knnx.index (knn.index), 11
mutinfo, 14
ownn, 8, 15
print.knnReg, 16
print.knnRegCV (print.knnReg), 16