Package 'KernelKnn'

January 16, 2018

Type Package

Title Kernel k Nearest Neighbors

Version 1.0.8

Date 2018-01-16

Author Lampros Mouselimis <mouselimislampros@gmail.com>

Maintainer Lampros Mouselimis <mouselimislampros@gmail.com>

BugReports https://github.com/mlampros/KernelKnn/issues

URL https://github.com/mlampros/KernelKnn

Description Extends the simple k-nearest neighbors algorithm by incorporating numerous kernel functions and a variety of distance metrics. The package takes advantage of 'RcppArmadillo' to speed up the calculation of distances between observations.

License MIT + file LICENSE

LazyData TRUE

Depends R(>= 2.10.0)

Imports Rcpp (>= 0.12.5)

LinkingTo Rcpp, RcppArmadillo

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-01-16 18:24:51 UTC

R topics documented:

`Boston` *Boston Housing Data (Regression)*

Description

housing values in suburbs of Boston

Usage

```
data(Boston)
```

Format

A data frame with 506 Instances and 14 attributes (including the class attribute, "medv")

`crim` per capita crime rate by town

`zn` proportion of residential land zoned for lots over 25,000 sq.ft.

`indus` proportion of non-retail business acres per town

`chas` Charles River dummy variable (= 1 if tract bounds)

`nox` nitric oxides concentration (parts per 10 million)

`rm` average number of rooms per dwelling

`age` proportion of owner-occupied units built prior to 1940

`dis` weighted distances to five Boston employment centres

`rad` index of accessibility to radial highways

`tax` full-value property-tax rate per $10,000

`ptratio` pupil-teacher ratio by town

`black` 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town

`lstat` percentage of lower status of the population

`medv` Median value of owner-occupied homes in $1000's

Source

This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

Creator: Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978.

References

https://archive.ics.uci.edu/ml/datasets/Housing

Examples

```
data(Boston)

X = Boston[, -ncol(Boston)]

y = Boston[, ncol(Boston)]
```

`distMat.KernelKnn` *kernel k-nearest-neighbors using a distance matrix*

Description

kernel k-nearest-neighbors using a distance matrix

Usage

```
distMat.KernelKnn(DIST_mat, TEST_indices = NULL, y, k = 5, h = 1,
  weights_function = NULL, regression = F, threads = 1, extrema = F,
  Levels = NULL, minimize = T)
```

Arguments

`DIST_mat`	a distance matrix (square matrix) having a *diagonal* filled with either zero's (*0*) or NA's (*missing values*)
`TEST_indices`	a numeric vector specifying the indices of the test data in the distance matrix (row-wise or column-wise). If the parameter equals NULL then no test data is included in the distance matrix
`y`	a numeric vector (in classification the labels must be numeric from 1:Inf). It is assumed that if the *TEST_indices* is not NULL then the length of *y* equals to the rows of the train data *(nrow(DIST_mat) - length(TEST_indices))*, otherwise *length(y) == nrow(DIST_mat)*.
`k`	an integer specifying the k-nearest-neighbors
`h`	the bandwidth (applicable if the weights_function is not NULL, defaults to 1.0)
`weights_function`	there are various ways of specifying the kernel function. See the details section.
`regression`	a boolean (TRUE,FALSE) specifying if regression or classification should be performed
`threads`	the number of cores to be used in parallel (openmp will be employed)
`extrema`	if TRUE then the minimum and maximum values from the k-nearest-neighbors will be removed (can be thought as outlier removal)

Levels	a numeric vector. In case of classification the unique levels of the response variable are necessary
minimize	either TRUE or FALSE. If TRUE then lower values will be considered as relevant for the k-nearest search, otherwise higher values.

Details

This function takes a distance matrix (square matrix where the diagonal is filled with *0* or *NA*) as input. If the *TEST_indices* parameter is NULL then the predictions for the train data will be returned, whereas if the *TEST_indices* parameter is not NULL then the predictions for the test data will be returned. There are three possible ways to specify the weights function, 1st option : if the weights_function is NULL then a simple k-nearest-neighbor is performed. 2nd option : the weights_function is one of 'uniform', 'triangular', 'epanechnikov', 'biweight', 'triweight', 'tricube', 'gaussian', 'cosine', 'logistic', 'gaussianSimple', 'silverman', 'inverse', 'exponential'. The 2nd option can be extended by combining kernels from the existing ones (adding or multiplying). For instance, I can multiply the tricube with the gaussian kernel by giving 'tricube_gaussian_MULT' or I can add the previously mentioned kernels by giving 'tricube_gaussian_ADD'. 3rd option : a user defined kernel function

Value

a vector (if regression is TRUE), or a data frame with class probabilities (if regression is FALSE)

Author(s)

Lampros Mouselimis

Examples

```
data(Boston)

X = Boston[, -ncol(Boston)]
y = Boston[, ncol(Boston)]

dist_obj = dist(X)

dist_mat = as.matrix(dist_obj)

out = distMat.KernelKnn(dist_mat, TEST_indices = NULL, y, k = 5, regression = TRUE)
```

---

distMat.KernelKnnCV *cross validated kernel-k-nearest-neighbors using a distance matrix*

Description

cross validated kernel-k-nearest-neighbors using a distance matrix

Usage

```
distMat.KernelKnnCV(DIST_mat, y, k = 5, folds = 5, h = 1,
  weights_function = NULL, regression = F, threads = 1, extrema = F,
  Levels = NULL, minimize = T, seed_num = 1)
```

Arguments

DIST_mat	a distance matrix (square matrix) having a diagonal filled with either zero's (*0*) or NA's (*missing values*)
y	a numeric vector (in classification the labels must be numeric from 1:Inf)
k	an integer specifying the k-nearest-neighbors
folds	the number of cross validation folds (must be greater than 1)
h	the bandwidth (applicable if the weights_function is not NULL, defaults to 1.0)
weights_function	there are various ways of specifying the kernel function. See the details section.
regression	a boolean (TRUE,FALSE) specifying if regression or classification should be performed
threads	the number of cores to be used in parallel (openmp will be employed)
extrema	if TRUE then the minimum and maximum values from the k-nearest-neighbors will be removed (can be thought as outlier removal)
Levels	a numeric vector. In case of classification the unique levels of the response variable are necessary
minimize	either TRUE or FALSE. If TRUE then lower values will be considered as relevant for the k-nearest search, otherwise higher values.
seed_num	a numeric value specifying the seed of the random number generator

Details

This function takes a number of arguments (including the number of cross-validation-folds) and it returns predicted values and indices for each fold. There are three possible ways to specify the weights function, 1st option : if the weights_function is NULL then a simple k-nearest-neighbor is performed. 2nd option : the weights_function is one of 'uniform', 'triangular', 'epanechnikov', 'biweight', 'triweight', 'tricube', 'gaussian', 'cosine', 'logistic', 'gaussianSimple', 'silverman', 'inverse', 'exponential'. The 2nd option can be extended by combining kernels from the existing ones (adding or multiplying). For instance, I can multiply the tricube with the gaussian kernel by giving 'tricube_gaussian_MULT' or I can add the previously mentioned kernels by giving 'tricube_gaussian_ADD'. 3rd option : a user defined kernel function

Value

a list of length 2. The first sublist is a list of predictions (the length of the list equals the number of the folds). The second sublist is a list with the indices for each fold.

Author(s)

Lampros Mouselimis

Examples

```
## Not run:
data(ionosphere)

X = ionosphere[, -c(2, ncol(ionosphere))]
y = as.numeric(ionosphere[, ncol(ionosphere)])

dist_obj = dist(X)

dist_mat = as.matrix(dist_obj)

out = distMat.KernelKnnCV(dist_mat, y, k = 5, folds = 3, Levels = unique(y))

## End(Not run)
```

`distMat.knn.index.dist` *indices and distances of k-nearest-neighbors using a distance matrix*

Description

indices and distances of k-nearest-neighbors using a distance matrix

Usage

```
distMat.knn.index.dist(DIST_mat, TEST_indices = NULL, k = 5, threads = 1,
  minimize = T)
```

Arguments

`DIST_mat`	a distance matrix (square matrix) having a diagonal filled with either zero's (*0*) or NA's (*missing values*)
`TEST_indices`	a numeric vector specifying the indices of the test data in the distance matrix (row-wise or column-wise). If the parameter equals NULL then no test data is included in the distance matrix
`k`	an integer specifying the k-nearest-neighbors
`threads`	the number of cores to be used in parallel (openmp will be employed)
`minimize`	either TRUE or FALSE. If TRUE then lower values will be considered as relevant for the k-nearest search, otherwise higher values.

Details

This function takes a number of arguments and it returns the indices and distances of the k-nearest-neighbors for each observation. If TEST_indices is NULL then the indices-distances for the DIST_mat be returned, whereas if TEST_indices is not NULL then the indices-distances for the test data only will be returned.

Value

a list of length 2. The first sublist returns the indices and the second the distances of the k nearest neighbors for each observation. If TEST_indices is NULL the number of rows of each sublist equals the number of rows in the DIST_mat data. If TEST_indices is not NULL the number of rows of each sublist equals the length of the input TEST_indices.

Author(s)

Lampros Mouselimis

Examples

```
data(Boston)

X = Boston[, -ncol(Boston)]

dist_obj = dist(X)

dist_mat = as.matrix(dist_obj)

out = distMat.knn.index.dist(dist_mat, TEST_indices = NULL, k = 5)
```

---

ionosphere *Johns Hopkins University Ionosphere database (binary classification)*

Description

This radar data was collected by a system in Goose Bay, Labrador. This radar data was collected by a system in Goose Bay, Labrador. This system consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. See the paper for more details. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this databse are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal.

Usage

```
data(ionosphere)
```

Format

A data frame with 351 Instances and 35 attributes (including the class attribute, "class")

Details

Sigillito, V. G., Wing, S. P., Hutton, L. V., \& Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. Johns Hopkins APL Technical Digest, 10, 262-266.

They investigated using backprop and the perceptron training algorithm on this database. Using the first 200 instances for training, which were carefully split almost 50 percent positive and 50 percent negative, they found that a "linear" perceptron attained 90.7 percent, a "non-linear" perceptron attained 92 percent, and backprop an average of over 96 percent accuracy on the remaining 150 test instances, consisting of 123 "good" and only 24 "bad" instances. (There was a counting error or some mistake somewhere; there are a total of 351 rather than 350 instances in this domain.) Accuracy on "good" instances was much higher than for "bad" instances. Backprop was tested with several different numbers of hidden units (in [0,15]) and incremental results were also reported (corresponding to how well the different variants of backprop did after a periodic number of epochs). David Aha (aha@ics.uci.edu) briefly investigated this database. He found that nearest neighbor attains an accuracy of 92.1 percent, that Ross Quinlan's C4 algorithm attains 94.0 percent (no windowing), and that IB3 (Aha \& Kibler, IJCAI-1989) attained 96.7 percent (parameter settings: 70 percent and 80 percent for acceptance and dropping respectively).

Source

Donor: Vince Sigillito (vgs@aplcen.apl.jhu.edu)

Date: 1989

Source: Space Physics Group

Applied Physics Laboratory

Johns Hopkins University

Johns Hopkins Road

Laurel, MD 20723

References

https://archive.ics.uci.edu/ml/datasets/Ionosphere

Examples

```
data(ionosphere)

X = ionosphere[, -ncol(ionosphere)]

y = ionosphere[, ncol(ionosphere)]
```

KernelKnn *kernel k-nearest-neighbors*

Description

This function utilizes kernel k nearest neighbors to predict new observations

Usage

```
KernelKnn(data, TEST_data = NULL, y, k = 5, h = 1, method = "euclidean",
  weights_function = NULL, regression = F, transf_categ_cols = F,
  threads = 1, extrema = F, Levels = NULL)
```

Arguments

data	a data frame or matrix
TEST_data	a data frame or matrix (it can be also NULL)
y	a numeric vector (in classification the labels must be numeric from 1:Inf)
k	an integer specifying the k-nearest-neighbors
h	the bandwidth (applicable if the weights_function is not NULL, defaults to 1.0)
method	a string specifying the method. Valid methods are 'euclidean', 'manhattan', 'chebyshev', 'canberra', 'braycurtis', 'pearson_correlation', 'simple_matching_coefficient', 'minkowski' (by default the order 'p' of the minkowski parameter equals k), 'hamming', 'mahalanobis', 'jaccard_coefficient', 'Rao_coefficient'
weights_function	there are various ways of specifying the kernel function. See the details section.
regression	a boolean (TRUE,FALSE) specifying if regression or classification should be performed
transf_categ_cols	a boolean (TRUE, FALSE) specifying if the categorical columns should be converted to numeric or to dummy variables
threads	the number of cores to be used in parallel (openmp will be employed)
extrema	if TRUE then the minimum and maximum values from the k-nearest-neighbors will be removed (can be thought as outlier removal)
Levels	a numeric vector. In case of classification the unique levels of the response variable are necessary

Details

This function takes a number of arguments and it returns the predicted values. If TEST_data is NULL then the predictions for the train data will be returned, whereas if TEST_data is not NULL then the predictions for the TEST_data will be returned. There are three possible ways to specify the weights function, 1st option : if the weights_function is NULL then a simple k-nearest-neighbor is performed. 2nd option : the weights_function is one of 'uniform', 'triangular', 'epanechnikov',

'biweight', 'triweight', 'tricube', 'gaussian', 'cosine', 'logistic', 'gaussianSimple', 'silverman', 'inverse', 'exponential'. The 2nd option can be extended by combining kernels from the existing ones (adding or multiplying). For instance, I can multiply the tricube with the gaussian kernel by giving 'tricube_gaussian_MULT' or I can add the previously mentioned kernels by giving 'tricube_gaussian_ADD'. 3rd option : a user defined kernel function

Value

a vector (if regression is TRUE), or a data frame with class probabilities (if regression is FALSE)

Author(s)

Lampros Mouselimis

Examples

```
data(Boston)

X = Boston[, -ncol(Boston)]
y = Boston[, ncol(Boston)]

out = KernelKnn(X, TEST_data = NULL, y, k = 5, method = 'euclidean', regression = TRUE)
```

KernelKnnCV *kernel-k-nearest-neighbors using cross-validation*

Description

This function performs kernel k nearest neighbors regression and classification using cross validation

Usage

```
KernelKnnCV(data, y, k = 5, folds = 5, h = 1, method = "euclidean",
  weights_function = NULL, regression = F, transf_categ_cols = F,
  threads = 1, extrema = F, Levels = NULL, seed_num = 1)
```

Arguments

data	a data frame or matrix
y	a numeric vector (in classification the labels must be numeric from 1:Inf)
k	an integer specifying the k-nearest-neighbors
folds	the number of cross validation folds (must be greater than 1)
h	the bandwidth (applicable if the weights_function is not NULL, defaults to 1.0)

`method` a string specifying the method. Valid methods are 'euclidean', 'manhattan', 'chebyshev', 'canberra', 'braycurtis', 'pearson_correlation', 'simple_matching_coefficient', 'minkowski' (by default the order 'p' of the minkowski parameter equals k), 'hamming', 'mahalanobis', 'jaccard_coefficient', 'Rao_coefficient'

`weights_function` there are various ways of specifying the kernel function. See the details section.

`regression` a boolean (TRUE,FALSE) specifying if regression or classification should be performed

`transf_categ_cols` a boolean (TRUE, FALSE) specifying if the categorical columns should be converted to numeric or to dummy variables

`threads` the number of cores to be used in parallel (openmp will be employed)

`extrema` if TRUE then the minimum and maximum values from the k-nearest-neighbors will be removed (can be thought as outlier removal)

`Levels` a numeric vector. In case of classification the unique levels of the response variable are necessary

`seed_num` a numeric value specifying the seed of the random number generator

Details

This function takes a number of arguments (including the number of cross-validation-folds) and it returns predicted values and indices for each fold. There are three possible ways to specify the weights function, 1st option : if the weights_function is NULL then a simple k-nearest-neighbor is performed. 2nd option : the weights_function is one of 'uniform', 'triangular', 'epanechnikov', 'biweight', 'triweight', 'tricube', 'gaussian', 'cosine', 'logistic', 'gaussianSimple', 'silverman', 'inverse', 'exponential'. The 2nd option can be extended by combining kernels from the existing ones (adding or multiplying). For instance, I can multiply the tricube with the gaussian kernel by giving 'tricube_gaussian_MULT' or I can add the previously mentioned kernels by giving 'tricube_gaussian_ADD'. 3rd option : a user defined kernel function

Value

a list of length 2. The first sublist is a list of predictions (the length of the list equals the number of the folds). The second sublist is a list with the indices for each fold.

Author(s)

Lampros Mouselimis

Examples

```
## Not run:
data(ionosphere)

X = ionosphere[, -c(2, ncol(ionosphere))]
y = as.numeric(ionosphere[, ncol(ionosphere)])
```

```
out = KernelKnnCV(X, y, k = 5, folds = 3, regression = FALSE, Levels = unique(y))

## End(Not run)
```

---

knn.index.dist *indices and distances of k-nearest-neighbors*

Description

This function returns the k nearest indices and distances of each observation

Usage

```
knn.index.dist(data, TEST_data = NULL, k = 5, method = "euclidean",
  transf_categ_cols = F, threads = 1)
```

Arguments

data	a data.frame or matrix
TEST_data	a data.frame or matrix (it can be also NULL)
k	an integer specifying the k-nearest-neighbors
method	a string specifying the method. Valid methods are 'euclidean', 'manhattan', 'chebyshev', 'canberra', 'braycurtis', 'pearson_correlation', 'simple_matching_coefficient', 'minkowski' (by default the order 'p' of the minkowski parameter equals k), 'hamming', 'mahalanobis', 'jaccard_coefficient', 'Rao_coefficient'
transf_categ_cols	a boolean (TRUE, FALSE) specifying if the categorical columns should be converted to numeric or to dummy variables
threads	the number of cores to be used in parallel (openmp will be employed)

Details

This function takes a number of arguments and it returns the indices and distances of the k-nearest-neighbors for each observation. If TEST_data is NULL then the indices-distances for the train data will be returned, whereas if TEST_data is not NULL then the indices-distances for the TEST_data will be returned.

Value

a list of length 2. The first sublist returns the indices and the second the distances of the k nearest neighbors for each observation. If TEST_data is NULL the number of rows of each sublist equals the number of rows in the train data. If TEST_data is not NULL the number of rows of each sublist equals the number of rows in the TEST data.

Author(s)

Lampros Mouselimis

Examples

```
data(Boston)

X = Boston[, -ncol(Boston)]

out = knn.index.dist(X, TEST_data = NULL, k = 4, method = 'euclidean', threads = 1)
```

Index