Package ‘snn’

October 14, 2022

Type Package
Title Stabilized Nearest Neighbor Classifier
Version 1.1
Date 2015-08-22
Author Wei Sun, Xingye Qiao, and Guang Cheng
Maintainer Wei Sun <sunweisurrey8@gmail.com>
Description Implement K-nearest neighbor classifier, weighted nearest neighbor classifier, bagged nearest neighbor classifier, optimal weighted nearest neighbor classifier and stabilized nearest neighbor classifier, and perform model selection via 5 fold cross-validation for them. This package also provides functions for computing the classification error and classification instability of a classification procedure.
License GPL-3
Depends R (>= 3.0.0), stats
NeedsCompilation no
Repository CRAN
Date/Publication 2015-08-23 10:22:09

R topics documented:

snn-package .................................................. 2
cv.tune .......................................................... 2
mybnn ............................................................ 4
mycis .............................................................. 5
mydata ............................................................ 6
myerror ........................................................... 7
myknn ............................................................. 8
myownn .......................................................... 10
mysnn ............................................................ 11
mywnn ............................................................ 12

Index 14
snn-package

Package for Stabilized Nearest Neighbor Classifier

Description

A package for implementations of various nearest neighbor classifiers, including K-nearest neighbor classifier, weighted nearest neighbor classifier, bagged nearest neighbor classifier, optimal weighted nearest neighbor classifier, and a new stabilized nearest neighbor classifier. This package also provides functions for computing the classification error and classification instability of a classification procedure.

Details

| Package: | snn |
| Type:    | Package |
| Version: | 1.0 |
| Date:    | 2015-07-31 |
| License: | GPL-3 |

The package "snn" provides 8 main functions: (1) the classification error. (2) the classification instability. (3) the K-nearest neighbor classifier. (4) the weighted neighbor classifier. (5) the bagged nearest neighbor classifier. (6) the optimal nearest neighbor classifier. (7) the stabilized nearest neighbor classifier. (8) the model selection via cross-validation for K-nearest neighbor classifier, bagged nearest neighbor classifier, optimal nearest neighbor classifier, and stabilized nearest neighbor classifier.

Author(s)

Wei Sun, Xingye Qiao, and Guang Cheng

Maintainer: Wei Sun <sunweisurrey8@gmail.com>

References


cv.tune

Tuning via 5 fold Cross-Validation.

Description

Implement the tuning procedure for K-nearest neighbor classifier, bagged nearest neighbor classifier, optimal weighted nearest neighbor classifier, and stabilized nearest neighbor classifier.
Usage

cv.tune(train, numgrid = 20, classifier = "snn")

Arguments

- **train**: Matrix of training data sets. An \(n\) by \((d+1)\) matrix, where \(n\) is the sample size and \(d\) is the dimension. The last column is the class label.

- **numgrid**: Number of grids for search

- **classifier**: The classifier for tuning. Possible choices are knn, bnn, ownn, snn.

Details

For the K-nearest neighbor classifier (knn), the grids for search are equal spaced integers in \([1, n/2]\).

Given the best \(k\) for the K-nearest neighbor classifier, the best parameter for the bagged nearest neighbor classifier (bnn) is computed via (3.5) in Samworth (2012).

Given the best \(k\) for the K-nearest neighbor classifier, the best parameter for Samworth’s optimal weighted nearest neighbor classifier (ownn) is computed via (2.9) in Samworth (2012).

For the stabilized nearest neighbor classifier (snn), we first identify a set of lambda’s whose corresponding risks are among the lower 10th percentiles, and then choose from them an optimal one which has the minimal estimated classification instability. The grids of lambda’s are chosen such that each one is corresponding to an evenly spaced grid of \(k\) in \([1, n/2]\). See Sun et al. (2015) for details.

Value

The returned list contains:

- **parameter.opt**: The best tuning parameter for the chosen classifier. For example, the best \(K\) for knn and ownn, the best ratio for bnn, and the best lambda for snn.

- **parameter.list**: The list of parameters in the grid search for the chosen classifier.

Author(s)

Wei Sun, Xingye Qiao, and Guang Cheng

References


Examples

```r
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)

## Tuning procedure
out.tune = cv.tune(DATA, classifier = "knn")
out.tune
```

mybnn

### Bagged Nearest Neighbor Classifier

**Description**

Implement the bagged nearest neighbor classification algorithm to predict the label of a new input using a training data set.

**Usage**

```r
mybnn(train, test, ratio)
```

**Arguments**

- `train`: Matrix of training data sets. An n by (d+1) matrix, where n is the sample size and d is the dimension. The last column is the class label.
- `test`: Vector of a test point. It also admits a matrix input with each row representing a new test point.
- `ratio`: Resampling ratio.

**Details**

The bagged nearest neighbor classifier is asymptotically equivalent to a weighted nearest neighbor classifier with the i-th weight a function of the resampling ratio, the sample size n, and i. See Hall and Samworth (2005) for details. The tuning parameter ratio can be tuned via cross-validation, see `cv.tune` function for the tuning procedure.

**Value**

It returns the predicted class label of the new test point. If input is a matrix, it returns a vector which contains the predicted class labels of all the new test points.

**Author(s)**

Wei Sun, Xingye Qiao, and Guang Cheng
References


Examples

# Training data
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)

# Testing data
set.seed(2015)
ntest = 100
TEST = mydata(n, d)
TEST.x = TEST[,1:d]

# bagged nearest neighbor classifier
mybnn(DATA, TEST.x, ratio = 0.5)

---

mycis  Classification Instability

Description

Compute the classification instability of a classification procedure.

Usage

mycis(predict1, predict2)

Arguments

predict1  The list of predicted labels based on one training data set.
predict2  The list of predicted labels based on another training data set.

Details

CIS of a classification procedure is defined as the probability that the same object is classified to two different classes by this classification procedure trained from two i.i.d. data sets. Therefore, the arguments predict1 and predict2 are generated on the same test data from the same classification procedure trained on two i.i.d. training data sets. CIS is among [0,1] and a smaller CIS represents a more stable classification procedure. See Section 2 of Sun et al. (2015) for details.
Author(s)

Wei Sun, Xingye Qiao, and Guang Cheng

References


Examples

```r
# Training data
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)

# Testing data
set.seed(2015)
ntest = 100
TEST = mydata(ntest, d)
TEST.x = TEST[,1:d]

## Compute classification instability for knn, bnn, ownn, and snn with given parameters
nn=floor(n/2)
permIndex = sample(n)
predict1.knn = myknn(DATA[permIndex[1:nn],], TEST.x, K = 5)
predict2.knn = myknn(DATA[permIndex[-(1:nn)],], TEST.x, K = 5)
predict1.bnn = mybnn(DATA[permIndex[1:nn],], TEST.x, ratio = 0.5)
predict2.bnn = mybnn(DATA[permIndex[-(1:nn)],], TEST.x, ratio = 0.5)
predict1.ownn = myownn(DATA[permIndex[1:nn],], TEST.x, K = 5)
predict2.ownn = myownn(DATA[permIndex[-(1:nn)],], TEST.x, K = 5)
predict1.snn = mysnn(DATA[permIndex[1:nn],], TEST.x, lambda = 10)
predict2.snn = mysnn(DATA[permIndex[-(1:nn)],], TEST.x, lambda = 10)

mycis(predict1.knn, predict2.knn)
mycis(predict1.bnn, predict2.bnn)
mycis(predict1.ownn, predict2.ownn)
mycis(predict1.snn, predict2.snn)
```

mydata

**Data Generator**

**Description**

Generate random data from mixture Gaussian distribution.
Usage

mydata(n, d, mu = 0.8, portion = 1/2)

Arguments

n
The number of observations (sample size).

d
The number of variables (dimension).

mu
In the Gaussian mixture model, the first Gaussian is generated with zero mean and identity covariance matrix. The second Gaussian is generated with mean a d-dimensional vector with all mu and identity covariance matrix.

portion
The prior probability for the first Gaussian component.

Value

Return the data matrix with n rows and d + 1 columns. Each row represents a sample generated from the mixture Gaussian distribution. The first d columns are features and the last column is the class label of the corresponding sample.

Author(s)

Wei Sun, Xingye Qiao, and Guang Cheng

Examples

set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)
DATA.x = DATA[,1:d]
DATA.y = DATA[,d+1]

myerror

Classification Error

Description

Compute the error of the predict list given the true list.

Usage

myerror(predict, true)
Arguments

- **predict**: The list of predicted labels
- **true**: The list of true labels

Value

It returns the errors of the predicted labels from a classification algorithm.

Author(s)

Wei Sun, Xingye Qiao, and Guang Cheng

Examples

```r
# Training data
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)

# Testing data
set.seed(2015)
ntest = 100
TEST = mydata(ntest, d)
TEST.x = TEST[,1:d]
TEST.y = TEST[,d+1]

## Compute the errors for knn, bnn, ownn, and snn with given parameters.
predict.knn = myknn(DATA, TEST.x, K = 5)
predict.bnn = mybnn(DATA, TEST.x, ratio = 0.5)
predict.ownn = myownn(DATA, TEST.x, K = 5)
predict.snn = mysnn(DATA, TEST.x, lambda = 10)

myerror(predict.knn, TEST.y)
myerror(predict.bnn, TEST.y)
myerror(predict.ownn, TEST.y)
myerror(predict.snn, TEST.y)
```

myknn — **K Nearest Neighbor Classifier**

Description

Implement the K nearest neighbor classification algorithm to predict the label of a new input using a training data set.
myknn

Usage

myknn(train, test, K)

Arguments

train  Matrix of training data sets. An n by (d+1) matrix, where n is the sample size and d is the dimension. The last column is the class label.

test  Vector of a test point. It also admits a matrix input with each row representing a new test point.

K  Number of nearest neighbors considered.

Details

The tuning parameter K can be tuned via cross-validation, see cv.tune function for the tuning procedure.

Value

It returns the predicted class label of the new test point. If input is a matrix, it returns a vector which contains the predicted class labels of all the new test points.

Author(s)

Wei Sun, Xingye Qiao, and Guang Cheng

References


Examples

# Training data
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)

# Testing data
set.seed(2015)
ntest = 100
TEST = mydata(ntest, d)
TEST.x = TEST[,1:d]

# K nearest neighbor classifier
myknn(DATA, TEST.x, K = 5)
**Description**

Implement Samworth’s optimal weighted nearest neighbor classification algorithm to predict the label of a new input using a training data set.

**Usage**

myownn(train, test, K)

**Arguments**

- **train**: Matrix of training data sets. An n by (d+1) matrix, where n is the sample size and d is the dimension. The last column is the class label.
- **test**: Vector of a test point. It also admits a matrix input with each row representing a new test point.
- **K**: Number of nearest neighbors considered.

**Details**

The tuning parameter K can be tuned via cross-validation, see cv.tune function for the tuning procedure.

**Value**

It returns the predicted class label of the new test point. If input is a matrix, it returns a vector which contains the predicted class labels of all the new test points.

**Author(s)**

Wei Sun, Xingye Qiao, and Guang Cheng

**References**


**Examples**

```r
# Training data
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)
```
# Testing data
set.seed(2015)
ntest = 100
TEST = mydata(ntest, d)
TEST.x = TEST[,1:d]

# optimal weighted nearest neighbor classifier
myownn(DATA, TEST.x, K = 5)

## mysnn

### Stabilized Nearest Neighbor Classifier

#### Description
Implement the stabilized nearest neighbor classification algorithm to predict the label of a new input using a training data set. The stabilized nearest neighbor classifier contains the K-nearest neighbor classifier and the optimal weighted nearest neighbor classifier as two special cases.

#### Usage
mysnn(train, test, lambda)

#### Arguments
- **train**: Matrix of training data sets. An \( n \) by \((d+1)\) matrix, where \( n \) is the sample size and \( d \) is the dimension. The last column is the class label.
- **test**: Vector of a test point. It also admits a matrix input with each row representing a new test point.
- **lambda**: Tuning parameter controlling the degree of stabilization of the nearest neighbor classification procedure. The larger \( \lambda \), the more stable the procedure is.

#### Details
The tuning parameter \( \lambda \) can be tuned via cross-validation, see cv.tune for the tuning procedure.

#### Value
It returns the predicted class label of the new test point. If input is a matrix, it returns a vector which contains the predicted class labels of all the new test points.

#### Author(s)
Wei Sun, Xingye Qiao, and Guang Cheng

#### References
Examples

```r
# Training data
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)

# Testing data
set.seed(2015)
ntest = 100
TEST = mydata(ntest, d)
TEST.x = TEST[,1:d]

# stabilized nearest neighbor classifier
mysnn(DATA, TEST.x, lambda = 10)
```

---

**mywnn**

**Weighted Nearest Neighbor Classifier**

**Description**

Implement the weighted nearest neighbor classification algorithm to predict the label of a new input using a training data set.

**Usage**

```r
mywnn(train, test, weight)
```

**Arguments**

- **train**: Matrix of training data sets. An n by (d+1) matrix, where n is the sample size and d is the dimension. The last column is the class label.
- **test**: Vector of a test point.
- **weight**: The weight vector for all n nearest neighbors.

**Value**

It returns the predicted class label of the new test point.

**Author(s)**

Wei Sun, Xingye Qiao, and Guang Cheng
Examples

```r
set.seed(1)
n = 100
d = 10
DATA = mydata(n, d)

## weighted nearest neighbor classifier
weight.vec = c(rep(0.02, 50), rep(0, 50))
mywnn(DATA, rep(-5, d), weight = weight.vec)
```
Index

cv.tune, 2
mybnn, 4
mycis, 5
mydata, 6
myerror, 7
myknn, 8
myownn, 10
mysnn, 11
mywnn, 12

snn (snn-package), 2
snn-package, 2