Package ‘knnflex’

April 17, 2009

Type Package

Title A more flexible KNN

Version 1.1.1

Depends R (>= 2.4.0)

Date 2007-02-01

Author Atina Dunlap Brooks

Maintainer ORPHANED

Description A KNN implementaion which allows continuous responses, the specification of the distance used to calculate nearest neighbors (euclidean, binary, etc.), the aggregation method used to summarize repsonse (majority class, mean, etc.) and the method of handling ties (all, random selection, etc.).

License GPL

Repository CRAN

Date/Publication 2009-02-23 11:00:57

R topics documented:

knnflex-package 2
classprob 3
knn.dist 4
knn.predict 6
knn.probability 9
majority 11

Index **12**

knnflex-package *A more flexible KNN*

Description

A K-Nearest Neighbor (KNN) implementation which allows the specification of the distance used to calculate nearest neighbors (euclidean, binary, etc.), the aggregation method used to summarize response (majority class, mean, etc.) and the method of handling ties (all, random selection, etc.). Additionally, responses are not limited to classes, but can be continuous.

Note

This work was funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant 1 P20 HG003900-01. Information on the Molecular Libraries Roadmap Initiative can be obtained from `http://nihroadmap.nih.gov/molecularlibraries/`.

Author(s)

Atina Dunlap Brooks

See Also

`knn`

Examples

```
# a quick classification example
x1 <- c(rnorm(20,mean=1),rnorm(20,mean=5))
x2 <- c(rnorm(20,mean=5),rnorm(20,mean=1))
x  <- cbind(x1,x2)
y <- c(rep(1,20),rep(0,20))
train <- sample(1:40,30)
# plot the training cases
plot(x1[train],x2[train],col=y[train]+1,xlab="x1",ylab="x2")
# predict the other cases
test <- (1:40)[-train]
kdist <- knn.dist(x)
preds <- knn.predict(train,test,y,kdist,k=3,agg.meth="majority")
# add the predictions to the plot
points(x1[test],x2[test],col=as.integer(preds)+1,pch="+")
# display the confusion matrix
table(y[test],preds)

# the iris example used by knn(class)
library(class)
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)))
```

```
# how to get predictions from knn(class)
pred<-knn(train, test, cl, k = 3)
# display the confusion matrix
table(pred,cl)

# how to get predictions with knn.dist and knn.predict
x <- rbind(train,test)
kdist <- knn.dist(x)
pred <- knn.predict(1:75, 76:150, cl, kdist, k=3)
# display the confusion matrix
table(pred,cl)

# note any small differences are a result of both methods
# breaking ties in majority class randomly

# 5-fold cross-validation to select k for above example
fold <- sample(1:5,75,replace=TRUE)
cvpred <- matrix(NA,nrow=75,ncol=10)
for (k in 1:10)
  for (i in 1:5)
    cvpred[which(fold==i),k] <- knn.predict(train=which(fold!=i),test=which(fold==i),cl,kdis
# display misclassification rates for k=1:10
apply(cvpred,2,function(x) sum(cl!=x))
```

---

classprob *Determines the prevalence of each class*

Description

A function which determines the prevalence or probability for each class of a vector (treated as factor).

Usage

```
classprob(x)
```

Arguments

x	a one dimensional vector

Details

This function treats the input vector as a factor and determines the probability for each level (class) of the factor. The order of the returned probabilities is the order of the `levels` command, which defaults to numeric or alphabetic order.

Value

A vector whose length is equal to the number of levels in the input. The order is numerically or alphabetically increasing. Note the factors may have levels which are not present in the vector, see examples for details.

Author(s)

Atina Dunlap Brooks

See Also

`majority` `factor`

Examples

```
#calculate probbilities
x <- sample( c("a","b","c","d","e"), 10, replace=TRUE )
classprob(x)
#label the probabilities
levels(as.factor(x))

#to see levels which aren't represnted in the vector
x<-as.factor(c('a','a','a','b','b','c'))
levels(x)
#now remove the 'c'
x<-x[1:5]
#but 'c' is still a level
levels(x)
#and the probability is calculated for it
classprob(x)
```

`knn.dist` *Calculates the distances to be used for KNN predictions*

Description

The distances to be used for K-Nearest Neighbor (KNN) predictions are calculated and returned as a symmetric matrix. Distances are calculated by `dist`.

Usage

```
knn.dist(x, dist.meth = "euclidean", p = 2)
```

Arguments

`x`	the entire dataset, the rows (cases) to be used for training and testing.
`dist.meth`	the distance to be used in calculating the neighbors. Any method valid in function `dist` is valid.
`p`	the power of the Minkowski distance.

Details

This function calculates the distances to be used by `knn.predict`. Distances are calculated between all cases. In the traditional scenario (a fixed n training cases, m disjoint test cases) this method will calculate more distances than required for prediction. For example, distances between training cases are not needed, but are calculated anyway. However, performance testing has shown that in most cases it is still faster to simply calculate all distances, even when many will not be used.

The advantage to calculating distances in a separate step prior to prediction, is that these calculations only need to be performed once. So, for example, cross-validation to select k can be performed on many values of k, with different cross-validation splits, all using a single run of knn.dist.

The default method for calculating distances is the "euclidean" distance, which is the method used by the `knn` function from the `class` package. Alternative methods may be used here. Any method valid for the the function `dist` is valid here. The parameter p may be specified with the Minkowski distance to use the *p* norm as the distance method.

Value

a square symmetric matrix whose dimensions are the number of rows in the original data. The diagonal contains zeros, the off diagonal entries will be >= 0.

Note

For the traditional scenario, classification using the Euclidean distance on a fixed set of training cases and a fixed set of test cases, the method `knn` is ideal. The functions `knn.dist` and `knn.predict` are intend to be used when something beyond the traditional case is desired. For example, prediction on a continuous y (non-classification), cross-validation for the selection of k, or the use of an alternate distance method are all possible with this package.

Author(s)

Atina Dunlap Brooks

See Also

`knn.predict`, `dist`, `knn`

Examples

```
#a quick classification example
# a quick classification example
x1 <- c(rnorm(20,mean=1),rnorm(20,mean=5))
x2 <- c(rnorm(20,mean=5),rnorm(20,mean=1))
x  <- cbind(x1,x2)
y <- c(rep(1,20),rep(0,20))
train <- sample(1:40,30)
# plot the training cases
plot(x1[train],x2[train],col=y[train]+1,xlab="x1",ylab="x2")
# predict the other cases
test <- (1:40)[-train]
kdist <- knn.dist(x)
```

```
preds <- knn.predict(train,test,y,kdist,k=3,agg.meth="majority")
# add the predictions to the plot
points(x1[test],x2[test],col=as.integer(preds)+1,pch="+")
# display the confusion matrix
table(y[test],preds)

# the iris example used by knn(class)
library(class)
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)))
# how to get predictions from knn(class)
pred<-knn(train, test, cl, k = 3)
# display the confusion matrix
table(pred,cl)

# how to get predictions with knn.dist and knn.predict
x <- rbind(train,test)
kdist <- knn.dist(x)
pred <- knn.predict(1:75, 76:150, cl, kdist, k=3)
# display the confusion matrix
table(pred,cl)

# note any small differences are a result of both methods
# breaking ties in majority class randomly

# 5-fold cross-validation to select k for above example
fold <- sample(1:5,75,replace=TRUE)
cvpred <- matrix(NA,nrow=75,ncol=10)
for (k in 1:10)
  for (i in 1:5)
    cvpred[which(fold==i),k] <- knn.predict(train=which(fold!=i),test=which(fold==i),cl,kdis
# display misclassification rates for k=1:10
apply(cvpred,2,function(x) sum(cl!=x))
```

knn.predict *KNN prediction routine using pre-calculated distances*

Description

K-Nearest Neighbor prediction method which uses the distances calculated by knn.dist.

Usage

```
knn.predict(train, test, y, dist.matrix, k = 1, agg.meth = if (is.factor(y)) "major
```

Arguments

`train`	indexes which specify the rows of *x* provided to `knn.dist` to be used in making the predictions
`test`	indexes which specify the rows of *x* provided to `knn.dist` to make predictions for
`y`	responses, see details below
`dist.matrix`	the output from a call to `knn.dist`
`k`	the number of nearest neighbors to consider
`agg.meth`	method to combine responses of the nearest neighbors, defaults to "majority" for classification and "mean" for continuous responses
`ties.meth`	method to handle ties for the kth neighbor, the default is "min" which uses all ties, alternatives include "max" which uses none if there are ties for the k-th nearest neighbor, "random" which selects among the ties randomly and "first" which uses the ties in their order in the data

Details

Predictions are calculated for each test case by aggregating the responses of the k-nearest neighbors among the training cases. `k` may be specified to be any positive integer less than the number of training cases, but is generally between 1 and 10.

The indexes for the training and test cases are in reference to the order of the entire data set as it was passed to `knn.dist`.

Only responses for the training cases are used. The responses provided in y may be those for the entire data set (test and training cases), or just for the training cases.

The aggregation may be any named function. By default, classification (factored responses) will use the "majority" class function and non-factored responses will use "mean". Other options to consider include "min", "max" and "median".

The ties are handled using the `rank` function. Further information may be found by examining the `ties.method` there.

Value

a vector of predictions whose length is the number of test cases.

Note

For the traditional scenario, classification using the Euclidean distance on a fixed set of training cases and a fixed set of test cases, the method `knn` is ideal. The functions `knn.dist` and `knn.predict` are intend to be used when something beyond the traditional case is desired. For example, prediction on a continuous y (non-classification), cross-validation for the selection of k, or the use of an alternate distance method are well handled.

Author(s)

Atina Dunlap Brooks

See Also

knn.dist, dist, knn

Examples

```
# a quick classification example
x1 <- c(rnorm(20,mean=1),rnorm(20,mean=5))
x2 <- c(rnorm(20,mean=5),rnorm(20,mean=1))
x  <- cbind(x1,x2)
y <- c(rep(1,20),rep(0,20))
train <- sample(1:40,30)
# plot the training cases
plot(x1[train],x2[train],col=y[train]+1,xlab="x1",ylab="x2")
# predict the other cases
test <- (1:40)[-train]
kdist <- knn.dist(x)
preds <- knn.predict(train,test,y,kdist,k=3,agg.meth="majority")
# add the predictions to the plot
points(x1[test],x2[test],col=as.integer(preds)+1,pch="+")
# display the confusion matrix
table(y[test],preds)

# the iris example used by knn(class)
library(class)
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)))
# how to get predictions from knn(class)
pred<-knn(train, test, cl, k = 3)
# display the confusion matrix
table(pred,cl)

# how to get predictions with knn.dist and knn.predict
x <- rbind(train,test)
kdist <- knn.dist(x)
pred <- knn.predict(1:75, 76:150, cl, kdist, k=3)
# display the confusion matrix
table(pred,cl)

# note any small differences are a result of both methods
# breaking ties in majority class randomly

# 5-fold cross-validation to select k for above example
fold <- sample(1:5,75,replace=TRUE)
cvpred <- matrix(NA,nrow=75,ncol=10)
for (k in 1:10)
  for (i in 1:5)
    cvpred[which(fold==i),k] <- knn.predict(train=which(fold!=i),test=which(fold==i),cl,kdis
# display misclassification rates for k=1:10
apply(cvpred,2,function(x) sum(cl!=x))
```

knn.probability *KNN prediction probability routine using pre-calculated distances*

Description

K-Nearest Neighbor prediction probability method which uses the distances calculated by `knn.dist`. For predictions (not probabilities) see `knn.predict`.

Usage

```
knn.probability(train, test, y, dist.matrix, k = 1, ties.meth = "min")
```

Arguments

`train`	indexes which specify the rows of *x* provided to `knn.dist` to be used in making the predictions
`test`	indexes which specify the rows of *x* provided to `knn.dist` to make predictions for
`y`	responses, see details below
`dist.matrix`	the output from a call to `knn.dist`
`k`	the number of nearest neighbors to consider
`ties.meth`	method to handle ties for the kth neighbor, the default is "min" which uses all ties, alternatives include "max" which uses none if there are ties for the k-th nearest neighbor, "random" which selects among the ties randomly and "first" which uses the ties in their order in the data

Details

Prediction probabilities are calculated for each test case by aggregating the responses of the k-nearest neighbors among the training cases and using the `classprob`. `k` may be specified to be any positive integer less than the number of training cases, but is generally between 1 and 10.

The indexes for the training and test cases are in reference to the order of the entire data set as it was passed to `knn.dist`.

Only responses for the training cases are used. The responses provided in y may be those for the entire data set (test and training cases), or just for the training cases.

The ties are handled using the `rank` function. Further information may be found by examining the `ties.method` there.

Value

a matirx of prediction probabilities whose number of columns is the number of test cases and the number of rows is the number of levels in the responses.

Note

For the traditional scenario, classification using the Euclidean distance on a fixed set of training cases and a fixed set of test cases, the method knn is ideal. The functions knn.dist and knn.predict are intend to be used when something beyond the traditional case is desired. For example, prediction on a continuous y (non-classification), cross-validation for the selection of k, or the use of an alternate distance method are well handled.

Author(s)

Atina Dunlap Brooks

See Also

knn.dist, knn.predict, knn

Examples

```
# the iris example used by knn(class)
library(class)
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)))
# how to get predictions from knn(class)
pred <- knn(train, test, cl, k = 3, prob=TRUE)
# display the confusion matrix
table(pred,cl)
# view probabilities (only the highest probability is returned)
attr(pred,"prob")

# how to get predictions with knn.dist and knn.predict
x <- rbind(train,test)
kdist <- knn.dist(x)
pred <- knn.predict(1:75, 76:150, cl, kdist, k=3)
# display the confusion matrix
table(pred,cl)
# view probabilities (all class probabilities are returned)
knn.probability(1:75, 76:150, cl, kdist, k=3)

# to compare probabilites, rounding done for display purposes
p1 <- knn(train, test, cl, k = 3, prob=TRUE)
p2 <- round(knn.probability(1:75, 76:150, cl, kdist, k=3), digits=2)
table( round(attr(p1,"prob"), digits=2), apply(p2,2,max) )

# note any small differences in predictions are a result of
# both methods breaking ties in majority class randomly
```

`majority` *Determines majority class*

Description

A function which determines the majority class of a vector (treated as factor).

Usage

```
majority(x)
```

Arguments

`x`	a one dimensional vector

Details

This function treats the input vector as a factor and determines which level (class) of the factor is present most often. If two or more levels tie for majority then a random selection is made among the ties.

Value

The factor level which occurs most often in x.

Author(s)

Atina Dunlap Brooks

See Also

`factor`

Examples

```
x <- sample( c("a","b","c","d","e"), 10, replace=TRUE )
majority(x)
```

Index

∗Topic **methods**
  knn.dist, 4
  knn.predict, 6
  knn.probability, 8
∗Topic **package**
  knnflex-package, 1
∗Topic **utilities**
  classprob, 3
  majority, 10

classprob, 3, *9*

dist, *4, 5*, *7*

factor, *3*, *11*

knn, *2*, *5*, *7*, *9*
knn.dist, 4, *5–9*
knn.predict, *4, 5*, 6, *7–9*
knn.probability, 8
knnflex-package, 1

levels, *3*

majority, *3*, 10

rank, *7*, *9*