Package 'knnGarden'

February 20, 2015

Type Package

Title Multi-distance based k-Nearest Neighbors

Version 1.0.1

Date 2012-05-16

Author Boxian Wei & Fan Yang & Xinmiao Wang & Yanni Ge

Special Acknowledgement Zhihong Huang & Weixian Jing

Maintainer Boxian Wei<boxianwei@gmail.com>

Description Multi-distance based k-Nearest Neighbors Classification with K Threshold Value Check and Same K_i Problem Dealing, Missing Observations Filling

License GPL (>= 2)

Depends cluster

Imports stats

Repository CRAN

Date/Publication 2012-07-13 12:42:02

NeedsCompilation no

R topics documented:

`dataFiller` *Missing Observations Filling Function*

Description

fill in the missing observations in a dataset by exploring similarities between cases

Usage

```
dataFiller(data, NAstring = NA)
```

Arguments

`data`	a dataset that contains missing observations in some cases
`NAstring`	a character or string that denotes missing values in the input dataset

Details

fill the cases with missing observations by finding the median of 10 most similar cases with the current one. Of course, the missing in the same column of the 10 cases will be removed when calculating the median. The criterion we define "similar" is based on euclidian distance between standardized cases

Value

A complete data set with missing observations filled will be returned.

Note

The cases with missing values in the input dataset will be printed on the screen instead of being returned. The return will be only the complete data set with missing observations filled.

Author(s)

Boxian Wei(The ideas are inspired by Luis Torgo, and thanks)

References

Luis Torgo (2003) Data Mining with R:learning by case studies. LIACC-FEP, University of Porto

See Also

knnMCN, knnVCN

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## Define Data
library(knnGarden)
data(iris)
v1=c(iris[1:4,3],NA,iris[6:10,3])
v2=iris[101:110,4]
v3=iris[101:110,1]
v4=c(iris[11:18,3],NA,iris[20,3])
data1=data.frame(v1,v2,v3,v4)

## Call Function
data2=dataFiller(data1)

## The function is currently defined as
function (data, NAstring = NA)
{
    central.value <- function(x) {
        if (is.numeric(x))
            median(x, na.rm = T)
        else if (is.factor(x))
            levels(x)[which.max(table(x))]
        else {
            f <- as.factor(x)
            levels(f)[which.max(table(f))]
        }
    }
    dist.mtx <- as.matrix(daisy(data, stand = T))
    ShowMissing = NULL
    ShowMissing = data[which(!complete.cases(data)), ]
    for (r in which(!complete.cases(data))) data[r, which(is.na(data[r,
        ]))] <- apply(data.frame(data[c(as.integer(names(sort(dist.mtx[r,
        ])[2:11]))), which(is.na(data[r, ]))]), 2, central.value)
    cat("the missing case(s) in the orignal dataset ", "\n\n")
    print(ShowMissing)
    cat("\n\n")
    return(data)
  }
```

---

knnMCN *k-Nearest Neighbour Classification of Mahalanobis Distance Version*

Description

k-nearest neighbour classification of Mahalanobis Distance version for test set from training set. This function allows you measure the distance bewteen vectors by Mahalanobis Distance. K Threshold Value Check and Same K_i Problem Dealing are also been considered.

Usage

```
knnMCN(TrnX, OrigTrnG, TstX = NULL, K = 1, ShowObs = F)
```

Arguments

TrnX	matrix or data frame of training set cases.
OrigTrnG	matrix or data frame of true classifications of training set.
TstX	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
K	number of top K nearest neighbours considered.
ShowObs	logical, when it's ture, the funtion will output the imformation of training set cases.

Details

The knnMCN function determines which class a undetermined case should belong to by following steps. First, calculate the Mahalanobis Distance between all the cases in training dataset. Then, select top K cases with nearest distances. Finally, these selected cases represent their classes and vote for the undetermined case under the principle of the minority is subordinate to the majority.

When calculating the Mahalanobis Distance, we use samples' covariance matrix (CM) and the Mahalanobis Distance is defined as follows:

MD=sqrt((X-Y)*inverse(CM)*transpose(X-Y)) , where X,Y are 1*n vectors and CM is n*n matrix.

Sometimes a case may get same "ballot" from class A and class B (even C, D, ...), this time a weighted voting process will be activated. The weight is based on the actual distance calculated between the test case and K cases in neighbor A and B. The test case belongs to the class with less total distance.

Also, to avoid unfair voting for undetermined case, K Threshold Value is stipulated to be less than the minimum size of the class in training dataset, or a warning will be shown.

Value

result of classifications of test set will be returned. (When TstX is NULL, the function will automatically consider the user is trying to test the knn algorithm. Hence, a test result table and accuracy report will be shown on the R-console.)

Note

Sometimes, singular covariance matrix may appear due to the input data and the size of the classes. This time the funtion will return a warning and hint the user try knnVCN to retry the classification.

Author(s)

Boxian Wei

References

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

See Also

knnVCN, dataFiller

Examples

```
library(knnGarden)
data(iris)

## Define data
TrnX=iris[c(1:20,80:100,140:150),1:4]
OrigTrnG=iris[c(1:20,80:100,140:150),5]
#
TstX<-iris[c(1:20,50:70,120:140),1:4]
#or
TstX<-NULL
## Call function
knnMCN(TrnX=TrnX,OrigTrnG=OrigTrnG,TstX=TstX,ShowObs=FALSE,K=5)


## The function is currently defined as
function (TrnX, OrigTrnG, TstX = NULL, K = 1, ShowObs = F)
{
    OrigTrnG = as.factor(OrigTrnG)
    TrnG = as.numeric(OrigTrnG)
    CodeMeaning = data.frame(TrnG, OrigTrnG)
    TK = sort(as.matrix(table(TrnG)), decreasing = F)
    if (K > TK[1]) {
        stop(c("\nNOTES: \nsorry, the value of K ", "(K=", K,
          ") ", "you have selected is bigger than the capacity of one class in your training data set",
           "(", "the capacity is ", TK[1], ")", ",", "please choose a less value for K"))
    }
    if (is.null(TstX) == T) {
        IsTst = 1
        TstX <- as.matrix(TrnX)
    }
    else {
        IsTst = 0
    }
    if (is.matrix(TstX) == F) {
        TstX <- as.matrix(TstX)
    }
    TrnX <- as.matrix(TrnX)
    ElmTrnG = union(TrnG, TrnG)
    LevTrnG = length(ElmTrnG)
    TrnTotal = cbind(TrnG, TrnX)
    if (abs(det(cov(TrnX[which(TrnTotal[, 1] == ElmTrnG[1]),
        ]))) < 1e-07) {
      stop("\nWarnings:\nsample variance-covariance matrix is singular,\nand larger class sample capacity is req
    }
    else {
        MaDisList = list(solve(cov(TrnX[which(TrnTotal[, 1] ==
            ElmTrnG[1]), ]), LINPACK = T))
```

```
}
if (LevTrnG > 1) {
    for (i in (1 + 1):LevTrnG) {
        if (abs(det(cov(TrnX[which(TrnTotal[, 1] == ElmTrnG[i]),
            ]))) < 1e-07) {
         stop("\nWarnings:\nsample variance-covariance matrix is singular,\nand larger class sample capacity
        }
        else {
            MaDisNode = list(solve(cov(TrnX[which(TrnTotal[,
              1] == ElmTrnG[i]), ]), LINPACK = T))
            MaDisList = c(MaDisList, MaDisNode)
        }
    }
}
NTstX = nrow(TstX)
NTrnTotal = nrow(TrnTotal)
VoteResult = NULL
VoteResultList = NULL
for (i in 1:nrow(TstX)) {
    RankBoardI <- NULL
    RankBoardIJ <- NULL
    for (j in 1:LevTrnG) {
        TempTrnXI = TrnX[which(TrnTotal[, 1] == ElmTrnG[j]),
            ]
        TempCovJ = as.matrix(MaDisList[[j]])
        TempTstXI = NULL
        for (k in 1:nrow(TempTrnXI)) {
            TempTstXI = rbind(TempTstXI, TstX[i, ])
        }
        TempMadisBoardI <- sqrt(diag((TempTstXI - TempTrnXI) %*%
            TempCovJ %*% t(TempTstXI - TempTrnXI)))
        MadisBoardI <- as.matrix(TempMadisBoardI)
        GBoardI <- as.matrix(rep(ElmTrnG[j], nrow(TempTrnXI)))
        RankBoardI <- cbind(GBoardI, MadisBoardI)
        RankBoardIJ <- rbind(RankBoardIJ, RankBoardI)
    }
    VoteAndWeight = RankBoardIJ[sort(RankBoardIJ[, 2], index.return = T)$ix[1:k],
        1:2]
    TempVote4TstXI = RankBoardIJ[sort(RankBoardIJ[, 2], index.return = T)$ix[1:k],
        1]
    ElmVote = union(TempVote4TstXI, TempVote4TstXI)
    CountVote = as.matrix(sort(table(TempVote4TstXI), decreasing = T))
    TempWinner = as.numeric(rownames(CountVote))
    if (length(CountVote) == 1 | K == 1) {
        Winner = TempWinner[1]
        TstXIBelong = union(CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
            Winner)], CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
            Winner)])
        VoteResultNode = data.frame(TstXIBelong)
        VoteResultList = rbind(VoteResultList, VoteResultNode)
    }
    else {
        NumOfTie = CountVote[1]
```

```
            FinalList = NULL
            j = 1
            TempWeight = sum(VoteAndWeight[which(VoteAndWeight[,
                1] == TempWinner[j]), 2])
            FinalList = data.frame(TempWinner[j], TempWeight)
            while (CountVote[j] == CountVote[j + 1] & j < length(CountVote)) {
                TempWeight = sum(VoteAndWeight[which(VoteAndWeight[,
                  1] == TempWinner[j + 1]), 2])
                FinalListNode = c(TempWinner[j + 1], TempWeight)
                FinalList = rbind(FinalList, FinalListNode)
                j = j + 1
            }
            FinalList = FinalList[sort(FinalList$TempWeight,
                index.return = T)$ix[1], ]
            TstXIBelong = union(CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
                FinalList[1, 1])], CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
                FinalList[1, 1])])
            VoteResultNode = data.frame(TstXIBelong)
            VoteResultList = rbind(VoteResultList, VoteResultNode)
        }
    }
    if (IsTst == 1) {
        CheckT = as.matrix(table(data.frame(VoteResultList, OrigTrnG)))
        AccuStat = 1 - sum(CheckT - diag(diag(CheckT)))/length(TrnG)
        cat("test results", "\n")
        print(CheckT)
       cat("the classification accuracy of this algorithm on this training dataset is: ",
            AccuStat * 100, "%", "\n\n\n")
    }
    if (IsTst == 1 & ShowObs == F) {
        result = data.frame(VoteResultList, OrigTrnG)
    }
    else {
        if (IsTst == 1 & ShowObs == T) {
            result = data.frame(TstX, VoteResultList, OrigTrnG)
        }
        else {
            if (ShowObs == F) {
                result = data.frame(VoteResultList)
            }
            else {
                result = data.frame(TstX, VoteResultList)
            }
        }
    }
    return(result)
}
```

Description

k-nearest neighbour classification of versatile Distance version for test set from training set. For each row of the test set, the k nearest (in multiple distances) training set vectors are found, and the classification is decided by majority vote. This function allows you measure the distance bewteen vectors by six different means. K Threshold Value Check and Same K_i Problem Dealing are also been considered.

Usage

```
knnVCN(TrnX, OrigTrnG, TstX, K = 1, ShowObs=F,method = "euclidean",p =2)
```

Arguments

TrnX	matrix or data frame of training set cases.
OrigTrnG	matrix or data frame of true classifications of training set.
TstX	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
K	number of top K nearest neighbours considered.
ShowObs	logical, when it's ture, the funtion will output the imformation of training set cases.
method	the distance measure to be used. This must be one of "euclidean", "maximum", "manhattan","canberra", "binary" or "minkowski". Any unambiguous substring can be given.
p	The power of the Minkowski distance.

Details

K Threshold Value is stipulated to be less than the minimum size of the class in training set, or a warning will be shown.

Sometimes a case may get same "ballot" from class A and class B (even C, D, ...), this time a weighted voting process will be activated. The weight is based on the actual distance calculated between the test case and K cases in neighbor A and B. The test case belongs to the class with less total distance.

The multiple distances are implemented by transfering the function dist(). For the convenience of users, we quote the details of function "dist()" and show them here.

Available distance measures are :

euclidean: Usual square distance between the two vectors (2 norm).

maximum: Maximum distance between two components of x and y (supremum norm)

manhattan: Absolute distance between the two vectors (1 norm).

canberra: sum(abs(Xi-Yi)/abs(Xi+Yi)) Terms with zero numerator and denominator are omitted from the sum and treated as if the values were missing.

This is intended for non-negative values (e.g. counts): taking the absolute value of the denominator is a 1998 R modification to avoid negative distances.

binary: (aka asymmetric binary): The vectors are regarded as binary bits, so non-zero elements are "on" and zero elements are "off". The distance is the proportion of bits in which only one is on amongst those in which at least one is on.

minkowski: The p norm, the pth root of the sum of the pth powers of the differences of the components.

Missing values are allowed, and are excluded from all computations involving the rows within which they occur. Further, when Inf values are involved, all pairs of values are excluded when their contribution to the distance gave NaN or NA. If some columns are excluded in calculating a Euclidean, Manhattan, Canberra or Minkowski distance, the sum is scaled up proportionally to the number of columns used. If all pairs are excluded when calculating a particular distance, the value is NA.

Value

result of classifications of test set will be returned. (When TstX is NULL, the function will automatically consider the user is trying to test the knn algorithm. Hence, a test result table and accuracy report will be shown on the R-console.)

Note

If you want to use the distance measure "binary", the vectors must be binary bits, non-zero elements are "on" and zero elements are "off".

Author(s)

Xinmiao Wang

References

Ripley, B. D. (1996) Pattern Recognition and Neural Networks. Cambridge.

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

See Also

knnMCN, dataFiller

Examples

```
library(knnGarden)
data(iris)
## Define data
TrnX<-iris[,1:4]
OrigTrnG<-iris[,5]
#
TstX<-iris[c(1:20,50:70,120:140),1:4]
#or
TstX<-NULL
## Call function
knnVCN(TrnX=TrnX,OrigTrnG=OrigTrnG,TstX=TstX,ShowObs=FALSE,K=5,method="euclidean",p = 2)
```

```
## The function is currently defined as
function (TrnX, OrigTrnG, TstX = NULL, K = 1, ShowObs = F, method = "euclidean",
    p = 2)
{
    OrigTrnG = as.factor(OrigTrnG)
    TrnG = as.numeric(OrigTrnG)
    CodeMeaning = data.frame(TrnG, OrigTrnG)
    TK = sort(as.matrix(table(TrnG)), decreasing = F)
    if (K > TK[1]) {
        stop(c("\nNOTES: \nsorry, the value of K ", "(K=", K,
          ") ", "you have selected is bigger than the capacity of one class in your training data set",
           "(", "the capacity is ", TK[1], ")", ",", "please choose a less value for K"))
    }
    if (is.null(TstX) == T) {
        IsTst = 1
        TstX <- as.matrix(TrnX)
    }
    else {
        IsTst = 0
    }
    if (is.matrix(TstX) == F) {
        TstX <- as.matrix(TstX)
    }
    TrnX <- as.matrix(TrnX)
    ElmTrnG = union(TrnG, TrnG)
    LevTrnG = length(ElmTrnG)
    TrnTotal = cbind(TrnG, TrnX)
    NTstX = nrow(TstX)
    NTrnTotal = nrow(TrnTotal)
    VoteResult = NULL
    VoteResultList = NULL
    for (i in 1:nrow(TstX)) {
        RankBoardI <- NULL
        RankBoardIJ <- NULL
        Total = rbind(TstX[i, ], TrnX)
        RankBoardI = as.matrix(dist(Total, method = method, p = p)[1:nrow(TrnX)])
        RankBoardIJ = cbind(TrnG, RankBoardI)
        VoteAndWeight = RankBoardIJ[sort(RankBoardIJ[, 2], index.return = T)$ix[1:K],
            1:2]
        TempVote4TstXI = RankBoardIJ[sort(RankBoardIJ[, 2], index.return = T)$ix[1:K],
            1]
        ElmVote = union(TempVote4TstXI, TempVote4TstXI)
        CountVote = as.matrix(sort(table(TempVote4TstXI), decreasing = T))
        TempWinner = as.numeric(rownames(CountVote))
        if (length(CountVote) == 1 | K == 1) {
            Winner = TempWinner[1]
            TstXIBelong = union(CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
                Winner)], CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
                Winner)])
            VoteResultNode = data.frame(TstXIBelong)
            VoteResultList = rbind(VoteResultList, VoteResultNode)
        }
```

```
        else {
            NumOfTie = CountVote[1]
            FinalList = NULL
            j = 1
            TempWeight = sum(VoteAndWeight[which(VoteAndWeight[,
                1] == TempWinner[j]), 2])
            FinalList = data.frame(TempWinner[j], TempWeight)
            while (CountVote[j] == CountVote[j + 1] & j < length(CountVote)) {
                TempWeight = sum(VoteAndWeight[which(VoteAndWeight[,
                  1] == TempWinner[j + 1]), 2])
                FinalListNode = c(TempWinner[j + 1], TempWeight)
                FinalList = rbind(FinalList, FinalListNode)
                j = j + 1
            }
            FinalList = FinalList[sort(FinalList$TempWeight,
                index.return = T)$ix[1], ]
            TstXIBelong = union(CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
                FinalList[1, 1])], CodeMeaning$OrigTrnG[which(CodeMeaning$TrnG ==
                FinalList[1, 1])])
            VoteResultNode = data.frame(TstXIBelong)
            VoteResultList = rbind(VoteResultList, VoteResultNode)
        }
    }
    if (IsTst == 1) {
        CheckT = as.matrix(table(data.frame(VoteResultList, OrigTrnG)))
        AccuStat = 1 - sum(CheckT - diag(diag(CheckT)))/length(TrnG)
        print(CheckT)
       cat("the classification accuracy of this algorithm on this training dataset is: ",
            AccuStat * 100, "%", "\n\n\n")
    }
    if (IsTst == 1 & ShowObs == F) {
        result = data.frame(VoteResultList, OrigTrnG)
    }
    else {
        if (IsTst == 1 & ShowObs == T) {
            result = data.frame(TstX, VoteResultList, OrigTrnG)
        }
        else {
            if (ShowObs == F) {
                result = data.frame(VoteResultList)
            }
            else {
                result = data.frame(TstX, VoteResultList)
            }
        }
    }
    return(result)
}
```

Index