Simple k-NN Example

Jon Kinsey

Sun Nov 30 13:12:37 2014

```
#
# see http://blog.webagesolutions.com/archives/1164
# http://www.inside-r.org/r-doc/class/knn
# 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:
# knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
# 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 sin
# cl :  factor of true classifications of training set
# k : number of neighbours considered.
# l : minimum vote for definite decision, otherwise doubt.
#    (More precisely, less than k-l dissenting votes are allowed, even if k
#    is increased by ties.)
# prob : If this is true, the proportion of the votes for the winning class
#        are returned as attribute prob.
# use.all : controls handling of ties. If true, all distances equal to the kth largest
# are included. If false, a random selection of distances equal to the kth is chosen
# to use exactly k neighbors.
#
# Load the class package that holds the knn() function
library(class)

# Class A cases
A1=c(0,0)
A2=c(1,1)
A3=c(2,2)

# Class B cases
B1=c(6,6)
B2=c(5.5,7)
B3=c(6.5,5)

# Build the classification matrix
train=rbind(A1,A2,A3, B1,B2,B3)

# Class labels vector (attached to each class instance)
cl=factor(c(rep("A",3),rep("B",3)))

# The object to be classified
test=c(4, 4)

# call knn() and get its summary
summary(knn(train, test, cl, k = 1))
```

```
## A B
## 0 1
```

```
# This result indicates that the test case has been classified as
# belonging to class B
test=c(3.5, 3.5)
# Visually, this test case point looks to be closer to the cluster of
# the A class cases (points). Let's verify our assumption.
summary(knn(train, test, cl, k = 1))
```

```
## A B
## 1 0
```

```
# The point has been classified as belonging to class A.

# Let's increase the number of closest neighbors that are involved in
# voting during the classification step.
summary(knn(train, test, cl, k = 3))
```

```
## A B
## 1 0
```

```
# Now, the positions of class B points make them closer as a whole
# (according to the Euclidean distance metric) to the test point, so
# the (3.5, 3.5) case is classified as belonging to class B this time.
#
# Here is another example using the iris built in dataset:
data(iris3)
train <- rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
cl <- factor(c(rep("s",25), rep("c",25), rep("v",25)))
#
# test is the dataset you are trying to classify. Given an existing (trained)
# knn structure, test observations are analyzed row by row, and a prediction
# is generated. The same dataset is used to construct test data. Of course,
# we know the true classification here, but we pretend that we do not.
# True classification is the same as before; it cannot be used by the knn: for
# knn this information is not available. We store this data in order to estimate
# our predictions.
test <- rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
cl.test <- cl
# Finally, we are ready to proceed. Here's a vector of predictions for the
# test dataset. If prob=TRUE, we additionally see how "confident" the algorithm
# is about each case:
pr.test <- knn(train, test, cl, k = 3, prob=TRUE)
knn(train, test, cl, k = 3, prob=TRUE)
```

```
##  [1] s s s s s s s s s s s s s s s s s s s s s s s s s c c v c c c c c v c
## [36] c c c c c c c c c c c c c c c v c c v v v v v v c v v v v v c v v v v v v
## [71] v v v v v
## attr(,"prob")
##  [1] 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
## [11] 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
```

```
## [21] 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 0.6667 1.0000 1.0000
## [31] 1.0000 1.0000 1.0000 0.6667 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
## [41] 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
## [51] 1.0000 0.6667 0.7500 1.0000 1.0000 1.0000 1.0000 1.0000 0.5000 1.0000
## [61] 1.0000 1.0000 1.0000 0.6667 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
## [71] 1.0000 0.6667 1.0000 1.0000 0.6667
## Levels: c s v
```

```
# We now may estimate how correct our model is.
sum(pr.test==cl.test)/length(cl.test)
```

```
## [1] 0.92
```

```
# We get 70 out of 75, or 93% correct.
#
# lets try again
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)
```

```
##      cl
## pred  c  s  v
##    c 23  0  3
##    s  0 25  0
##    v  2  0 22
```

```
#
# Here is another simple example
# http://jakeporway.com/teaching/code/lecture12.R
example <- read.csv('http://jakeporway.com/teaching/data/example_data.csv', h=T, as.is=T)
n <- nrow(example)
set.seed(1) # This sets the random number generator

# Let's create training and testing sets (something R's kNN needs).
# To do this, let's sample a random set of rows from our data
# to be training and testing
indices <- sort(sample(1:n, n * (1 / 2)))

# Get our data points
training.x <- example[indices, 1:2]
test.x <- example[-indices, 1:2]

# Get their labels
training.y <- example[indices, 3]
test.y <- example[-indices, 3]

# Predict!
predicted.y <- knn(training.x, test.x, training.y, k = 5)
```

```r
# How'd we do?
sum(predicted.y != test.y)
```

```
## [1] 7
```

```r
#[1] 7
# Whadda ya know, only 7/100 wrong.

length(test.y)
```

```
## [1] 50
```

```r
#[1] 50
```