CV-for-KNN-Parameter-Tuning.R

patriciamaya

2020-12-07

```
#KNN CLASSIFIER
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 4.0.2
```

```
data("Carseats")
Data<- Carseats
attach(Data)
head(Data)
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 1  9.50       138     73          11        276   120       Bad  42        17
## 2 11.22       111     48          16        260    83      Good  65        10
## 3 10.06       113     35          10        269    80    Medium  59        12
## 4  7.40       117    100           4        466    97    Medium  55        14
## 5  4.15       141     64           3        340   128       Bad  38        13
## 6 10.81       124    113          13        501    72       Bad  78        16
##   Urban  US
## 1   Yes Yes
## 2   Yes Yes
## 3   Yes Yes
## 4   Yes Yes
## 5   Yes  No
## 6    No Yes
```

```
#create categorical variable for Sales (High, not high)
Data$Sales <- as.factor(ifelse(Sales>=8, "High", "Low"))

#When we use KNN data should be balanced, normalized, and all categorical variables
#should be converted to numerical variables using dummy variables

#checking if data is balanced
summary(Data$Sales)
```

```
## High  Low
##  164  236
```

```
mean(Data$Sales == "High") #41% is ok--> data is balanced, unbalanced if around 20%
```

```
## [1] 0.41
```

```
#*** DATAFRAME 1
#normalizing numerical variables -- min max transformation
#if we don't  normalize, the variables with large values dominate the other variables in
#the distance function
```

```r
num_vars <- unlist(lapply(Data, is.numeric))
num_vars
```

```
##       Sales   CompPrice      Income Advertising  Population       Price
##       FALSE        TRUE        TRUE        TRUE        TRUE        TRUE
##   ShelveLoc         Age   Education       Urban          US
##       FALSE        TRUE        TRUE       FALSE       FALSE
```

```r
DataNum<- Data[ , num_vars]  #new df of num vars only
head(DataNum)
```

```
##   CompPrice Income Advertising Population Price Age Education
## 1       138     73          11        276   120  42        17
## 2       111     48          16        260    83  65        10
## 3       113     35          10        269    80  59        12
## 4       117    100           4        466    97  55        14
## 5       141     64           3        340   128  38        13
## 6       124    113          13        501    72  78        16
```

```r
#min max normalization using scale function
mins <- apply(DataNum, 2 ,min)  #Gets min value of each column -- 1 rows, 2 cols
maxs <- apply(DataNum, 2, max)  #Gets max value of each column
Data.scaled <- scale(DataNum, center = mins, scale = maxs-mins)
summary(Data.scaled)
```

```
##    CompPrice          Income        Advertising       Population
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.3878   1st Qu.:0.2197   1st Qu.:0.0000   1st Qu.:0.2585
##  Median :0.4898   Median :0.4848   Median :0.1724   Median :0.5251
##  Mean   :0.4895   Mean   :0.4814   Mean   :0.2288   Mean   :0.5107
##  3rd Qu.:0.5918   3rd Qu.:0.7071   3rd Qu.:0.4138   3rd Qu.:0.7786
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##      Price             Age           Education
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.4551   1st Qu.:0.2682   1st Qu.:0.2500
##  Median :0.5569   Median :0.5364   Median :0.5000
##  Mean   :0.5497   Mean   :0.5150   Mean   :0.4875
##  3rd Qu.:0.6407   3rd Qu.:0.7455   3rd Qu.:0.7500
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

```r
#*** DATAFRAME 2
#Convert categorical(factor) variables into numerical dummy variables
factor <- !num_vars
factor
```

```
##       Sales   CompPrice      Income Advertising  Population       Price
##        TRUE       FALSE       FALSE       FALSE       FALSE       FALSE
##   ShelveLoc         Age   Education       Urban          US
##        TRUE       FALSE       FALSE        TRUE        TRUE
```

```r
factor[1] <- "FALSE"    #this is the target variable, we want to keep it as factor
factor <- as.logical(factor)
factor
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE
```

```r
library(psych)
```

```
DataFactor <- as.data.frame(sapply(Data[, factor], dummy.code) )
head(DataFactor)      #df of dummy variables
```

```
##   ShelveLoc.Medium ShelveLoc.Bad ShelveLoc.Good Urban.Yes Urban.No US.Yes US.No
## 1                0             1              0         1        0      1     0
## 2                0             0              1         1        0      1     0
## 3                1             0              0         1        0      1     0
## 4                1             0              0         1        0      1     0
## 5                0             1              0         1        0      0     1
## 6                0             1              0         0        1      1     0
```

```
#*** COMBINING 2 DATAFRAMES & target variables
Data_norm <- data.frame(Data.scaled, DataFactor, Data$Sales)
head(Data_norm)
```

```
##   CompPrice    Income Advertising Population     Price       Age Education
## 1 0.6224490 0.5252525   0.3793103  0.5330661 0.5748503 0.3090909     0.875
## 2 0.3469388 0.2727273   0.5517241  0.5010020 0.3532934 0.7272727     0.000
## 3 0.3673469 0.1414141   0.3448276  0.5190381 0.3353293 0.6181818     0.250
## 4 0.4081633 0.7979798   0.1379310  0.9138277 0.4371257 0.5454545     0.500
## 5 0.6530612 0.4343434   0.1034483  0.6613226 0.6227545 0.2363636     0.375
## 6 0.4795918 0.9292929   0.4482759  0.9839679 0.2874251 0.9636364     0.750
##   ShelveLoc.Medium ShelveLoc.Bad ShelveLoc.Good Urban.Yes Urban.No US.Yes US.No
## 1                0             1              0         1        0      1     0
## 2                0             0              1         1        0      1     0
## 3                1             0              0         1        0      1     0
## 4                1             0              0         1        0      1     0
## 5                0             1              0         1        0      0     1
## 6                0             1              0         0        1      1     0
##   Data.Sales
## 1       High
## 2       High
## 3       High
## 4        Low
## 5        Low
## 6       High
```

```
#Split train/test
set.seed(2)
indx  <- sample(2,nrow(Data_norm), replace=TRUE, prob = c(0.7, 0.3))
#we have to separate target variable from input variables (-15)
train <- Data_norm[indx==1, -15]  #removing target variable-Sales
test  <- Data_norm[indx==2, -15]  #removing target variable-Sales
ncol(train)
```

```
## [1] 14
```

```
#vector of target variable
trainLabels <- Data_norm[indx==1, 15]
testLabels <- Data_norm[indx==2, 15]

#*** KNN MODEL
library(FNN)
```

```
## Warning: package 'FNN' was built under R version 4.0.2
```

```r
pred_class <- knn(train= train, test= test, cl=trainLabels, k=3, prob=T)

indices <- attr(pred_class, "nn.index")
print(indices[20,])  #returns 3 NN training examples of test point 20
```

```
## [1] 139  71 173
```

```r
probs <- attr(pred_class, "prob")
probs
```

```
##   [1] 1.0000000 0.6666667 1.0000000 1.0000000 1.0000000 0.6666667 0.6666667
##   [8] 0.6666667 1.0000000 1.0000000 0.6666667 0.6666667 1.0000000 0.6666667
##  [15] 1.0000000 1.0000000 0.6666667 0.6666667 1.0000000 0.6666667 1.0000000
##  [22] 1.0000000 0.6666667 0.6666667 0.6666667 1.0000000 0.6666667 0.6666667
##  [29] 1.0000000 0.6666667 1.0000000 0.6666667 1.0000000 0.6666667 0.6666667
##  [36] 0.6666667 0.6666667 1.0000000 0.6666667 0.6666667 0.6666667 0.6666667
##  [43] 1.0000000 0.6666667 1.0000000 0.6666667 0.6666667 1.0000000 0.6666667
##  [50] 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667 1.0000000 0.6666667
##  [57] 1.0000000 0.6666667 1.0000000 0.6666667 1.0000000 1.0000000 1.0000000
##  [64] 0.6666667 0.6666667 1.0000000 0.6666667 0.6666667 1.0000000 0.6666667
##  [71] 0.6666667 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667 0.6666667
##  [78] 1.0000000 1.0000000 0.6666667 1.0000000 0.6666667 0.6666667 0.6666667
##  [85] 1.0000000 0.6666667 0.6666667 0.6666667 1.0000000 1.0000000 0.6666667
##  [92] 0.6666667 0.6666667 1.0000000 1.0000000 0.6666667 1.0000000 0.6666667
##  [99] 0.6666667 1.0000000 0.6666667 0.6666667 0.6666667 0.6666667 1.0000000
## [106] 1.0000000 0.6666667 0.6666667 0.6666667 0.6666667 0.6666667 1.0000000
## [113] 1.0000000 0.6666667 1.0000000 1.0000000 0.6666667 1.0000000 0.6666667
## [120] 1.0000000
```

```r
dist <- attr(pred_class, "nn.dist")
print(dist[20, ])
```

```
## [1] 0.3771954 0.4019596 0.4310047
```

```r
table(pred_class, testLabels)
```

```
##           testLabels
## pred_class High Low
##       High   33  20
##       Low    13  54
```

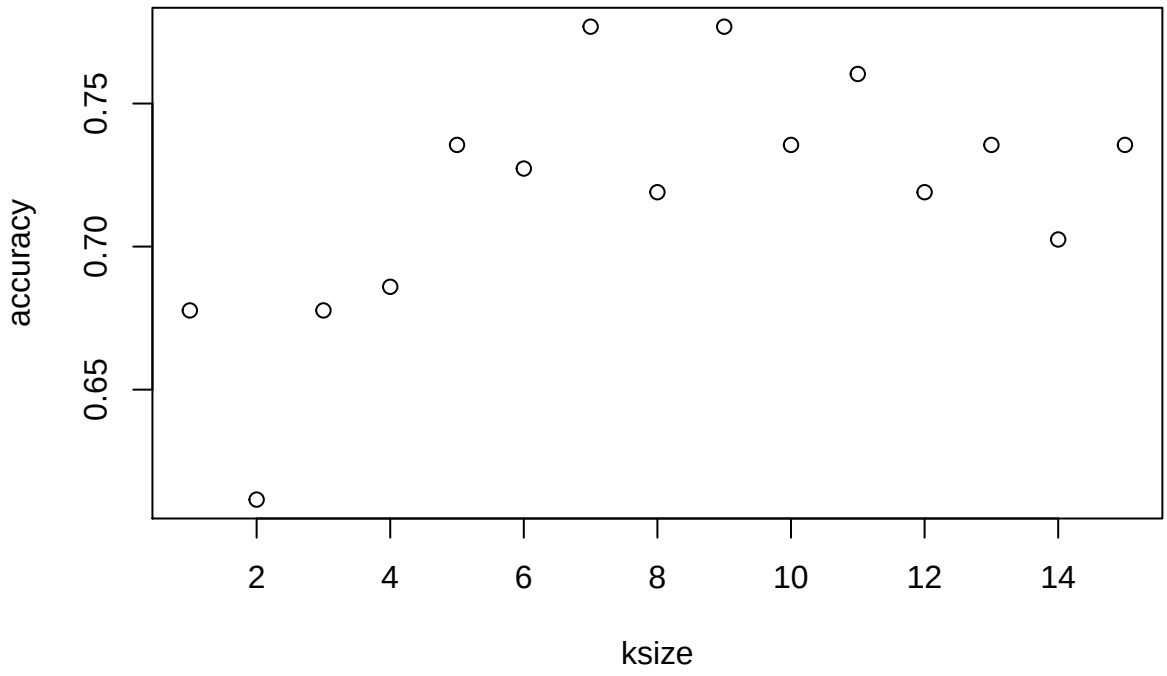
```r
#not the best performance, change K-value using cross validation.
```

```r
#USING CROSS VALIDATION TO CHOOSE BEST K VALUE
#partitioning taining data
indx<- sample(2, nrow(train), replace=TRUE, prob=c(0.6, 0.4))
trainD <- train[indx==1,]
validD<- train[indx==2,]

train_Labels<- trainLabels[indx==1]
valid_Labels<- trainLabels[indx==2]

ksize <- c()          #empty vector
accuracy <- c()
for (i in 1:15)
{
  library(FNN)
```

```
  prc_test_pred <- knn(train=trainD, test=validD, cl=train_Labels, k=i)
  ksize <- c(ksize, i)
  confusion_matrix <- table(valid_Labels, prc_test_pred)
  accuracy <- c(accuracy, sum(diag(confusion_matrix))/sum(confusion_matrix))
}
result <- data.frame(ksize, accuracy)
result
```

```
##    ksize  accuracy
## 1      1 0.6776860
## 2      2 0.6115702
## 3      3 0.6776860
## 4      4 0.6859504
## 5      5 0.7355372
## 6      6 0.7272727
## 7      7 0.7768595
## 8      8 0.7190083
## 9      9 0.7768595
## 10    10 0.7355372
## 11    11 0.7603306
## 12    12 0.7190083
## 13    13 0.7355372
## 14    14 0.7024793
## 15    15 0.7355372
```

```
plot(result)
```




```
#we can see best result here is k=7

#****RE-RUN KNN model with best value of k found on CV using the entire dataset.
pred_class_bestk <- knn(train= train, test= test, cl=trainLabels, k=7, prob=T)
table(pred_class_bestk, testLabels)
```

```
##                 testLabels
```

```
## pred_class_bestk High Low
##             High   30  13
##             Low    16  61
```

```
#accuracy of model improved.
```