kNN practice

Practicing k-Nearest Neighbors

- Training/Test split
- Try LOOCV to select the best number of neighbors by knn.cv()
- Try k-fold CV to select the best number of neighbors by do.chunk()
- Compute Training and Test error rates

```r
# install.packages("ISLR")
# install.packages("ggplot2") # install.packages("plyr")
# install.packages("dplyr")
# install.packages("class")
# Load libraries
library(ISLR)
library(ggplot2)
library(reshape2)
library(plyr)
library(dplyr)
library(class)
```

Carseats is a simulated data set containing sales of child car seats at 400 different stores. There are 11 variables (3 categorical and 8 numerical).

```r
data(Carseats)

str(Carseats)
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

```r
colnames(Carseats)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

High is a new binary feature as the response variable where High = No if Sales <= median(Sales) or High = Yes if Sales > median(Sales). I will drop Sales and the 3 discrete independent variables(ShelveLoc, Urban and US). The goal is to investigate the relationship between High and all the continuous explanatory variables.

```r
seats <- Carseats %>% mutate(High=as.factor(ifelse(Sales<=median(Sales), "Low", "High"))) %>% select(-Sa
str(seats)
```

```
## 'data.frame':    400 obs. of  8 variables:
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ High       : Factor w/ 2 levels "High","Low": 1 1 1 2 2 1 2 1 2 2 ...
```

a. Training/Testing Split

First, I will check the test error where I sample 50% of the observations as a training set, and the other 50% as a test set.

```r
# Set random seed
set.seed(333)

# Sample 50% observations as training data
train = sample(1:nrow(seats), 200)
# seats.train = entire rows where the row is specified by the values of train
seats.train = seats[train,]

# The rest 50% as test data
seats.test = seats[-train,]
```

```r
# YTrain is the observed labels for High on the training set, XTrain is the design matrix
YTrain = seats.train$High
XTrain = seats.train %>% select(-High) %>% scale(center=TRUE, scale=TRUE)
```

```r
# Test set should be centered and scaled based on the means and variances of the training set
meanvec <- attr(XTrain,'scaled:center')
sdvec <- attr(XTrain,'scaled:scale')

# YTest is the observed labels for High on the test set, Xtest is the design matrix
YTest = seats.test$High
XTest = seats.test %>% select(-High) %>% scale(center = meanvec, scale = sdvec)
```

b. Train a kNN classifier and calculate error rates

Now I apply knn() function to train the kNN classifier on the training set and make predictions on training and test sets.

To get the training error, I have to train the kNN classifier on the training set and predict High on the same training set, then I can construct the 2x2 confusion matrix to get the training error rate. I have train=XTrain, test=XTrain, and cl=YTrain in knn(). For now I use k=2.

```r
set.seed(444)

# knn - train the classifier and make predictions on the TRAINING set
pred.YTtrain = knn(train=XTrain, test=XTrain, cl=YTrain, k=2)
```

```r
# Calculate confusion matrix
conf.train = table(predicted=pred.YTtrain, observed=YTrain)
conf.train
```

```
##          observed
## predicted High Low
##      High   84  10
##      Low    16  90
```

```r
# Train accuracy rate
sum(diag(conf.train)/sum(conf.train))
```

```
## [1] 0.87
```

```r
# Train error rate
1 - sum(diag(conf.train)/sum(conf.train))
```

```
## [1] 0.13
```

To get the test error, I train kNN on the training set and predict High on the test set, then construct the 2x2 confusion matrix to get the test error rate, where train=XTrain, test=XTest, and cl=YTrain in knn().

```r
set.seed(555)

# knn - train the classifier on TRAINING set and make predictions on TEST set
pred.YTest = knn(train=XTrain, test=XTest, cl=YTrain, k=2)

# Get confusion matrix
conf.test = table(predicted=pred.YTest, observed=YTest)
conf.test
```

```
##          observed
## predicted High Low
##      High   58  44
##      Low    41  57
```

```r
# Test accuracy rate
sum(diag(conf.test)/sum(conf.test))
```

```
## [1] 0.575
```

```r
# Test error rate
1 - sum(diag(conf.test)/sum(conf.test))
```

```
## [1] 0.425
```

Test error rate obtained by 2-NN classifier is not good, since 42.5% of the test observations are incorrectly predicted. I should change k, number of neighbors, to improve the error rates. To find optimal value, I will use cross validation.

c. k-fold and Leave-One-Out Cross-validation for selecting best k

- Leave-One-Out Cross-validation (LOOCV)

LOOCV is a special case of k-fold CV in which k is set to equal n. LOOCV has the potential to be expensive to implement, since the model has to be fit n times. This can be very time consuming if n is large, and if each individual model is slow to fit."

```r
# validation.error(a vector) to save validation errors in future
validation.error = NULL

# Give possible number of nearest neighbours to be considered
allK = 1:50

# Set random seed to make the results reproducible
set.seed(66)
```

```r
# For each number in allK, use LOOCV to find a validation error
for (i in allK){ # Loop through different number of neighbors
  pred.Yval = knn.cv(train=XTrain, cl=YTrain, k=i) # Predict on the left out validation set
  validation.error = c(validation.error, mean(pred.Yval!=YTrain)) # Combine all validation errors
}

# Validation error for 1-NN, 2-NN, ..., 50-NN
validation.error
```

```
##  [1] 0.350 0.375 0.340 0.340 0.325 0.305 0.330 0.310 0.315 0.295 0.290 0.325
## [13] 0.300 0.315 0.295 0.340 0.310 0.320 0.305 0.290 0.320 0.300 0.305 0.320
## [25] 0.310 0.315 0.285 0.275 0.280 0.300 0.295 0.300 0.295 0.310 0.310 0.330
## [37] 0.305 0.320 0.305 0.310 0.300 0.300 0.305 0.300 0.310 0.305 0.295 0.335
## [49] 0.325 0.320
```

```r
# Best number of neighbors. In the case of a tie, pick larger number of neighbors for simpler model
numneighbor = max(allK[validation.error == min(validation.error)])
numneighbor
```

```
## [1] 28
```

```r
set.seed(67)

# Best k used
pred.YTest = knn(train=XTrain, test=XTest, cl=YTrain, k=numneighbor)

# Confusion matrix
conf.matrix = table(predicted=pred.YTest, true=YTest)
conf.matrix
```

```
##          true
## predicted High Low
##      High   73  36
##      Low    26  65
```

```r
# Test accuracy rate
sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.69
```

```r
# Test error rate
1 - sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.31
```

- k-fold Cross-Validation

This approach involves randomly dividing the set of observations into k groups, or folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining k - 1 folds.

This procedure is repeated k times; each time, a different group of observations is treated as a validation set and the rest k - 1 folds as a training set.

For regression problems, the mean squared error(MSE) is computed on the observations in the kth hold-out fold. This process results in k estimates of the test error: MSE1, MSE2, . . . , MSEk. The k-fold CV error is computed by averaging these values,

For classification problems, the error rate is computed on the observations in the kth hold-out fold. Similarly as in regresssion cases, the CV process yield Err1, Err2, . . . , Errk. The k-fold CV error is similarly computed by averaging these values.

Create function do.chunk() to select the best number of neighbors using a k-fold CV and to calculate the test error rate afterwards.

```
# do.chunk() returns a data frame consisting of all possible values of folds, each training error and v
do.chunk <- function(chunkid, folddef, Xdat, Ydat, ...){
  train = (folddef!=chunkid) # Get training index
  Xtr = Xdat[train,] # Get training set by the above index
  Ytr = Ydat[train] # Get true labels in training set

  Xvl = Xdat[!train,] # Get validation set
  Yvl = Ydat[!train] # Get true labels in validation set

  predYtr = knn(train=Xtr, test=Xtr, cl=Ytr, ...) # Predict training labels
  predYvl = knn(train=Xtr, test=Xvl, cl=Ytr, ...) # Predict validation labels

  data.frame(fold = chunkid, # k folds
             train.error = mean(predYtr != Ytr), # Training error for each fold
             val.error = mean(predYvl != Yvl)) # Validation error for each fold
}
```

Firstly, I specify k = 3 in k-fold CV, and use cut() and sample() to assign an interval index (1, 2, or 3) to each observation in the training set.

Divide all observations from the training set into 3 intervals, namely, interval 1, 2 and 3. To make the division more random, I sample from all the interval indices without replacement. Call the resulting vector folds.

```
# Specify a 3-fold CV
nfold = 3

# cut: divides all training observations into 3 intervals;
#      labels = FALSE instructs R to use integers to code different intervals
set.seed(66)
folds = cut(1:nrow(seats.train), breaks=nfold, labels=FALSE) %>% sample()
folds
```

```
##   [1] 2 2 3 2 1 3 3 2 2 1 1 2 1 3 1 1 2 2 3 3 2 1 1 2 3 2 3 2 3 1 1 3 2 1 3 2 3
##  [38] 3 2 1 1 3 1 3 3 2 3 2 2 1 2 3 3 2 3 1 3 1 1 2 1 2 1 1 2 1 2 3 2 1 2 2 3 3
##  [75] 3 3 1 1 1 2 1 3 3 1 1 1 3 3 1 2 1 3 2 3 1 1 2 2 3 3 1 1 3 3 3 2 3 3 3 2 2
## [112] 3 2 2 3 1 2 2 2 3 3 3 2 1 2 2 2 3 1 2 1 1 1 2 3 3 2 1 2 1 1 1 3 3 2 1 1 2
## [149] 3 1 3 2 3 3 1 1 1 3 1 2 1 2 2 1 3 3 2 3 2 1 1 2 1 2 3 1 1 3 3 2 1 2 2 1 2
## [186] 2 1 2 3 3 2 3 1 1 2 3 1 3 3 1
```

Secondly, use do.chunk() to perform a 3-fold CV for selecting the best number of neighbors. The idea is similar to LOOCV, but 3-fold CV instead of doing a 200-fold CV.

```
# Set error.folds (a vector) to save validation errors in future
error.folds = NULL
```

```r
# Give possible number of nearest neighbours to be considered
allK = 1:50

set.seed(888)

# Loop through different number of neighbors
for (j in allK){
  tmp = ldply(1:nfold, do.chunk, # Apply do.chunk() function to each fold
              folddef=folds, Xdat=XTrain, Ydat=YTrain, k=j)

  tmp$neighbors = j # Keep track of each value of neighors

  error.folds = rbind(error.folds, tmp) # combine results
}
```

```r
# dim(error.folds)
head(error.folds)
```

```
##   fold train.error val.error neighbors
## 1    1   0.0000000 0.3731343         1
## 2    2   0.0000000 0.3484848         1
## 3    3   0.0000000 0.3134328         1
## 4    1   0.1578947 0.3582090         2
## 5    2   0.2238806 0.3484848         2
## 6    3   0.1654135 0.3731343         2
```

Thirdly, decide the optimal k for kNN based on error.folds.

```r
# Transform the format of error.folds for further convenience
errors = melt(error.folds, id.vars=c('fold', 'neighbors'), value.name='error')

# Choose the number of neighbors which minimizes validation error
val.error.means = errors %>%
  # Select all rows of validation errors
  filter(variable=='val.error') %>%
  # Group the selected data frame by neighbors
  group_by(neighbors, variable) %>%
  # Calculate CV error rate for each k
  summarise_each(funs(mean), error) %>%
  # Remove existing group
  ungroup() %>%
  filter(error==min(error))
```

```r
# Best number of neighbors
# if there is a tie, pick larger number of neighbors for simpler model
numneighbor = max(val.error.means$neighbors)
numneighbor
```

```
## [1] 31
```

Fourthly, train a 31-NN classifier, and calculate the test error rate.

```r
set.seed(99)
pred.YTest = knn(train=XTrain, test=XTest, cl=YTrain, k=numneighbor)

# Confusion matrix
conf.matrix = table(predicted=pred.YTest, true=YTest)
```

```r
# Test accuracy rate
sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.68
```

```r
# Test error rate
1 - sum(diag(conf.matrix)/sum(conf.matrix))
```

```
## [1] 0.32
```