503: HW #1

Diana Liang

1/30/2020

## Part 1: Flexible Method: Better or Worse

For K-nearest neighbors (KNN) as a classification model, the decision of how many neighbors to consider (K) is important. Choosing a lower number (ex. K = 5) can make the model more flexible while choosing a higher number (ex. K = 100) can make the model more inflexible.

Different circumstances will determine whether a flexible method will be better or worse than an inflexible method.

# A) Large Sample Size, Few Predictors

In this case, a flexible method would be better since there is more information on the actual distribution of classes, and the flexible method would yield lower train error.

# B) Many Predictors, Few Observations

In this case, a flexible method would be worse since there is not much information on the actual distribution of the classes, and a more flexible method would overfit to the train data. And adding more dimensions would require more observations in order to find the K nearest neighbors that are actually representative of that specific space, which is not available here.

# C) Non-linear Relationship between Predictors and Response

In this case, a flexible method would be better since it would allow the model to fit in a more irregular shape to accomodate the non-linear relationship.

# D) High Variance of Error

In this case, a flexible method would be worse since overall variance increases with increased flexibility. There is already greater variance due to the high variance of error terms, so decreasing variance through decreasing flexibility would be preferable.

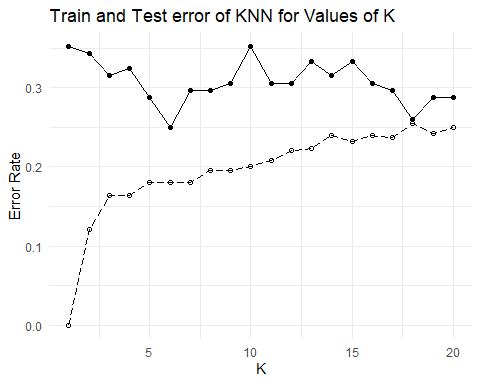
## Part 2: KNN for diabetes Dataset

The following code uses KNN to model the Outcome of the diabetes data based on its other variables. The train data was cleaned by getting rid of any observations were zero for Glucose, BloodPressure, and BMI since those values should be greater than zero for a human to be alive. The variables Insulin and SkinThickness were removed from both the train and test data entirely since so too many observations were removed since most of the Insulin and SkinThickness values were zero.

# Load packages  
library(dplyr); library(tidyr); library(ggplot2); library(class)  
  
# Load the data  
train\_og <- readr::read\_csv("diabetes\_train.csv")

test\_og <- readr::read\_csv("diabetes\_test.csv")

# Clean up data  
train\_og <- train\_og %>%  
 select(-c(Insulin, SkinThickness)) %>%  
 filter(Glucose > 0, BloodPressure > 0, BMI > 0)  
  
test\_og <- test\_og %>%  
 select(-c(Insulin, SkinThickness))  
  
# Get rid of Outcome Column  
train <- train\_og %>%  
 select(-Outcome)  
  
test <- test\_og %>%  
 select(-Outcome)   
  
# Calculate mean and std to standardize variables  
means <- colMeans(train)  
stds = sqrt(diag(var(train)))  
  
train\_x <- scale(train, center = means, scale = stds)  
test\_x <- scale(test, center = means, scale = stds)  
  
# Run KNN for k=1:20  
## Set up k and empty error vectors  
k\_val = c(1:20)  
train\_err = c()  
test\_err = c()  
  
## Run KNN for each value of k  
for(i in 1:20){  
 ## Calc error in predicting Outcome for train  
 pred\_train <- knn(train\_x, train\_x,   
 train\_og$Outcome,   
 k = k\_val[i])  
 train\_err[i] = mean(pred\_train != train\_og$Outcome)  
 ## Calc error in predicting Outcome for test  
 pred\_test = knn(train\_x,test\_x,   
 train\_og$Outcome,   
 k = k\_val[i])  
 test\_err[i] = mean(pred\_test != test\_og$Outcome)  
}  
  
# Plot the errors against K  
errors = data.frame(train\_err, test\_err, k\_val)  
ggplot(errors, aes(x = k\_val)) +   
 geom\_line(aes(y = train\_err), linetype = "longdash") +   
 geom\_point(aes(y = train\_err), shape = 1) +  
 geom\_line(aes(y = test\_err), linetype = "solid") +   
 geom\_point(aes(y = test\_err), shape = 16) +  
 ylab("Error Rate") + xlab("K") +   
 ggtitle("Train and Test error of KNN for Values of K") +  
 theme\_minimal()



Here is the train error for K = 1 to K = 20 in order of increasing K values:

train\_err

## [1] 0.0000000 0.1212871 0.1633663 0.1633663 0.1806931 0.1806931 0.1806931  
## [8] 0.1955446 0.1955446 0.2004950 0.2079208 0.2202970 0.2227723 0.2400990  
## [15] 0.2326733 0.2400990 0.2376238 0.2549505 0.2425743 0.2500000

Here is the test error for K = 1 to K = 20 in order of increasing K values:

test\_err

## [1] 0.3518519 0.3425926 0.3148148 0.3240741 0.2870370 0.2500000 0.2962963  
## [8] 0.2962963 0.3055556 0.3518519 0.3055556 0.3055556 0.3333333 0.3148148  
## [15] 0.3333333 0.3055556 0.2962963 0.2592593 0.2870370 0.2870370

While both train and test error were plotted and displayed, minimizing test error is more important than minimizing train error since the main objective is to find the model that best fits new data. The K with the lowest test error is shown below:

# Returns K with the lowest test error  
k\_val[which.min(test\_err)]

## [1] 6

Interestingly, the best K value changes by iteration. This could be due to the K values being very close together which leads to many of the models having similar train and test errors. Usually the K values tests would be at different magnitudes so that a greater range of models can be tested. In this case, any of the models with K = 5 to K = 12 could possibly be the model with the lowest test error and, thus, the best fitting model.