DEMO 2 - KNN CLASSIFICATION AND KMEANS CLUSTERING

QUESTION 1 - KNN WITH TEST-TRAIN SPLIT AND CROSS-VALIDATION

Data Story ~

The cross-validation results are consistent with the manual sampling and with the simple test-train KNN models with an improved accuracy of around 2.3%. The 80-20 and 70-30 data split don't vary significantly but the predictors used in the model affect the accuracy and error. The accuracy when using all 10 predictors was around 85% while the accuracy while using attributes - 3,5,7,9 was found to be 88.32%. This difference suggests that some factors don't play a significant role in determining whether a customer is eligible for credit or no.

As for the kfold value in cross-validation, 8-12 fold cross-validation proved to produce better accuracy, in contrast to lower or higher values that gave substantially extreme neighbour values, with the upper limit going all the way to k = 50 and lower limit going to down to k = 5, all with deteriorating accuracy values.

In [ ]:
```
#Code

library(ISLR)
library(ggplot2)
library(reshape2)
library(plyr)
library(dplyr)
library(class)
library(combinat)

set.seed(400)
#create an index to split the data: 80% training and 20% test
index = round(nrow(credit_card_data)*0.2,digits=0)
#sample randomly throughout the dataset and keep the total number equal to the value of index
test.indices = sample(1:nrow(credit_card_data), index)
#80% training set
credit_card_data.train=credit_card_data[-test.indices,]
#20% test set
credit_card_data.test=credit_card_data[test.indices,]
#Select the training set except the DV
YTrain = credit_card_data.train$V11
XTrain = credit_card_data.train %>% select(-11)
# Select the test set except the DV
YTest = credit_card_data.test$V11
XTest = credit_card_data.test %>% select(-11)

#define an error rate function and apply it to obtain test/training errors
calc_error_rate <- function(predicted.value, true.value){
 return(mean(true.value!=predicted.value))
}



nfold = 10
set.seed(400)
# cut() divides the range into several intervals
folds = seq.int(nrow(credit_card_data.train)) %>%
      cut(breaks = nfold, labels=FALSE) %>%
      sample

do.chunk <- function(chunkid, folddef, Xdat, Ydat, k){
  train = (folddef!=chunkid)# training index

Xtr = Xdat[train,] # training set by the index
Ytr = Ydat[train] # true label in training set
Xvl = Xdat[!train,] # test set
Yvl = Ydat[!train] # true label in test set
predYtr = knn(train = Xtr, test = Xtr, cl = Ytr, k = k) # predict training labels
predYvl = knn(train = Xtr, test = Xvl, cl = Ytr, k = k) # predict test labels
data.frame(fold =chunkid, # k folds
train.error = calc_error_rate(predYtr, Ytr),#training error per fold
 val.error = calc_error_rate(predYvl, Yvl)) # test error per fold
 }

# set error.folds to save validation errors
error.folds=NULL

# create a sequence of data with an interval of 10
kvec = c(1, seq(10, 50, length.out=5))
set.seed(1)
for (j in kvec){
 tmp = ldply(1:nfold, do.chunk, # apply do.function to each fold
 folddef=folds, Xdat=XTrain, Ydat=YTrain, k=j) # required arguments
 tmp$neighbors = j # track each value of neighbors
 error.folds = rbind(error.folds, tmp) # combine the results
 }

#melt() in the package reshape2 melts wide-format data into long-format data
errors = melt(error.folds, id.vars=c('fold','neighbors'), value.name= 'error')



val.error.means = errors %>%
    #select all rows of validation errors
    filter(variable== 'val.error' ) %>%
    #group the selected data by neighbors
    group_by(neighbors, variable) %>%
    #calcluate CV error for each k
    summarise_each(funs(mean), error) %>%
    #remove existing grouping
    ungroup() %>%
    filter(error==min(error))
# Best number of neighbors
# if there is a tie, pick larger number of neighbors for simpler model
numneighbor = max(val.error.means$neighbors)
numneighbor
```

Code Results

The best accuracy and performance results are from the values -

k = 12

k-fold cross validation = 10

Test - Train Split = 30 - 70 (in percentage)

Accuracy = 88.32487

Attribute Set = 3,5,7,9

QUESTION 2 - REALISTIC CLUSTERING EXAMPLE

Consider a commercial project where we give cluster millions of students' code submissions into groups based on certain criteria with the intention of giving feedback to these clusters in order to help these students get better at the code they write. The following would be the predictors that would come most in use when creating these clusters. Now please note that code submissions are in the form of unstructured data, and if encoded in a language like XML, they can also be parsed and evaluated for information that may prove to be a useful predictor. Hence, here five examples of predictors to if a specific concept is learnt or not -

1. Correctness - Numerical Predictor

- Does the code pass the unit test or not? If yes, then the value is 1, else it is 0.

1. Time Complexity - Categorical Predictor

- After running the code multiple times on different data, an average of the time lapse is taken to get the code's time complexity. The largest degree of the code is extracted and converted to a string and used as a categorical variable. This could include - n2, n3, n4, logn, nlogn, and so on.

1. Space Complexity - Categorical Predictor

- Similar to time complexity, the program heap can be inspected and the space complexity can be calculated and simplified into categorical variables.

1. Imported Library Function Usage - Numerical Predictor

- The number of built-in library functions they used to get the task done.

1. Code Style - Numerical Predictor

- This is the number of non-assignment lines they used to create their code.

There are other predictors I would use. With XML you can find out looping structures, conditionals, and so on in the code and evaluate them based on count and parameters.

ABOUT THE CLUSTERS - The final clusters in which the students submission go will be given post-clustering feedback on code performance and quality. Ideally, for an evaluator to get a good perspective of how his students are performing as a population, the count of clusters shouldn't be below 5 and shouldn't exceed 15. Between this range if new predictors are derived, or some fine-tuned or removed, to improve the relevance and accuracy of group feedback to the students in the group, that should be done.

QUESTION 3 - KMEANS CLUSTERING

Data Story ~

There are four predictors here - 1) SepalLength

2) SepalWidth

3) PetalLength

4) PetalWidth

The following are the possible combinations -

1) 4C4 = 1 Combination (Base Case)

- 1234

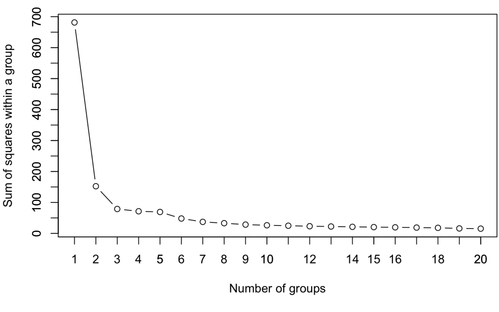
2) 4C3 = 4 Combinations

- 123, 124, 134, 234

3) 4C2 - 6 Combinations

- 12, 13, 14, 23, 24, 34

The following is a graph for the base case depicting the number of clusters versus the sum of squares within the group. The sum of squares within the group shows the compactness and similarity measure of the group. The higher it is the better. Of course, we should note that the first observation - 1 cluster - should be ignored.



Now observe how the steepest difference in compactness occurs from 3 groups to 2 groups where the measure differs by 75. Let us make note of the differences and group that comes after this dip. And see which of the combinations of attributes produces the highest compactness for the steepest dip.

When we do this we observe that the best cluster results are almost always at k = 3, hence now we can normalize our data, cluster it with different predictor sets and check the accuracy value.

Upon doing so, we get results for each combination in this form -

```
  iris.class
  setosa versicolor virginica
    1     50          1         0
    2      0          3        38
    3      0         46        12
```

And the accuracy can be calculated this way - Classified correct = 50 + 38 + 46 = 134 Classified incorrect = 3 + 0 = 3

Therefore, accuracy = 134/(134+3) = 97.8% And this was achieved with the SepalLength and PetalWidth attributes (1 and 4). For the base case, the results re as follows for k = 3 -

```
  iris.class
     setosa    versicolor    virginica
1      50          0          0
2       0         47         14
3       0          3         36
```

And the accuracy is 88.67%.

In [ ]:
```
#Code for Compactness Check to get Cluster Number and Attribute Set

input <- iris[,1:4]

wssplot <- function(data, nc=15, seed=123){
               wss <- (nrow(data)-1)*sum(apply(data,2,var))
               for (i in 2:nc){
                set.seed(seed)
                wss[i] <- sum(kmeans(data, centers=i)$withinss)}
               plot(1:nc, wss, type="b", xlab="Number of groups",ylab="Sum of squares within a group")
               axis(1, at = seq(0, 20, by = 1))
               axis(2, at = seq(0, 700, by = 50))}

w <- wssplot(input, nc = 20)
w
```

In [ ]:
```
#Code for Cluster number and Attribute Confirmation and Accuracy Check

iris.new<- iris[,c(1,2,3,4)]
iris.class<- iris[,"Species"]
normalize <- function(x){
  return ((x-min(x))/(max(x)-min(x)))
}

iris.new$Sepal.Length<- normalize(iris.new$Sepal.Length)
iris.new$Sepal.Width<- normalize(iris.new$Sepal.Width)
iris.new$Petal.Length<- normalize(iris.new$Petal.Length)
iris.new$Petal.Width<- normalize(iris.new$Petal.Width)

result<- kmeans(iris.new,3) #apply k-means algorithm with no. of centroids(k)=3
result$size # gives no. of records in each cluster

i = 1
j = 4

result<- kmeans(iris.new[c(i,j)],3)
table(result$cluster,iris.class)
```

Code Results

The best accuracy and performance results are from the values -

k = 3 clusters

Attribute Set = SepalLength and PetalWidth

Accuracy = 97.8%