Homework\_2

Alan Lo, Arjun Goyal, James Trawick, Richard Hardis

9/5/2019

## Question 3.1

Using the same data set (credit\_card\_data.txt or credit\_card\_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier: (a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional); and (b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

To answer parts a) and b) for question 3.1, we created several functions to allow for repeatability in our testing of multiple k-values and model parameters for KNN and SVM. These functions are copied below The ksvm\_accuracy and kknn\_accuracy functions return the accuracy as a 0-1 value of the how well the model performed on the training set with the provided model parameters. The get\_kfolds\_expected\_accuracy function returns the mean of the kfolds cross validation results. THe train\_test\_split function returns two dataframes; the first dataframe is the training data with test data removed and the second is the test dataframe taken from the overall provided dataset. train\_test\_split can be run by reserving a percentage of data for test or by specifying the number of datapoints to reserve for test.

##   
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':  
##   
## alpha

### Question 3.1-a)

knn\_accs = c()  
svm\_accs = c()  
  
for (i in 1:10){  
 knn\_accs = c(knn\_accs, get\_kfolds\_expected\_accuracy(data\_df, 10, 'kknn', i))  
 svm\_accs = c(svm\_accs, get\_kfolds\_expected\_accuracy(data\_df, 10, 'ksvm', i))  
}  
  
print(knn\_accs)

## [1] 0.8098812 0.8061472 0.8253520 0.8452231 0.8448789 0.8508316 0.8440347  
## [8] 0.8406402 0.8369028 0.8375807

print(svm\_accs)

## [1] 0.8547380 0.8467781 0.8454181 0.8414270 0.8349164 0.8379041 0.8409810  
## [8] 0.8306577 0.8268673 0.8196210

Using the functions defined above, we ran the get\_kfolds\_expected\_accuracy function over a range of 1:10 for the k parameter for KNN and C parameter for SVM and found that the best KNN model had k=6 with an accuracy of 0.8508316 and the SVM model did not change accuracy based on C values between 1-10. This confirms the result found last week. The cross validation exercise also did not show any folds with outlier accuracies, allowing us to conclude that our k and C values provided good models.

### Question 3.1-b)

To run train-validate-test operations on our dataset for SVM and KNN, we start by partitioning the entire dataset into three components using the following R code:

split\_data = train\_test\_split(data\_df, 20)  
test\_data = split\_data[[2]]  
train\_val\_data = split\_data[[1]]  
split\_data\_tv = train\_test\_split(train\_val\_data, 20)  
train\_data = split\_data\_tv[[1]]  
validate\_data = split\_data\_tv[[2]]  
  
train\_data = as.matrix(train\_data)  
validate\_data = as.matrix(validate\_data)  
test\_data = as.matrix(test\_data)  
  
# 60% Training Data, %20 Validation Data, %20 Test Data

Once the data is split up, we can run trials where the SVM and KNN models are trained on 60% of the data and their performance judged with the 20% of the dataset that was set aside for validation purposes. The SVM and KNN models are varied by the k value for KNN and C value for SVM. The best performing model based on the validation set is chosen. In this case we found that C values between 1 and 100 made little difference in model performance and that the best k value was 4.

svm\_validation\_accs = c()  
knn\_validation\_accs = c()  
  
for (i in 1:5){  
 C = 10^i  
 svm\_validation\_accs = c(svm\_validation\_accs, ksvm\_accuracy(train\_data, validate\_data, "vanilladot", C))  
 knn\_validation\_accs = c(knn\_validation\_accs, kknn\_accuracy(train\_data = as.data.frame(train\_data), i, "rectangular"))  
}

## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters   
## Setting default kernel parameters

svm\_index\_best\_acc = min(which(svm\_validation\_accs==max(svm\_validation\_accs)))  
knn\_index\_best\_acc = min(which(knn\_validation\_accs==max(knn\_validation\_accs)))  
best\_C\_val = 10^(svm\_index\_best\_acc-1)  
best\_k\_val = knn\_index\_best\_acc  
print(best\_C\_val)

## [1] 1

print(best\_k\_val)

## [1] 4

Lastly, we run the chosen models on the 20% of data set aside for testing and assess their performance. The model performances can be seena in the printout from the R code below.

test\_svm\_acc = ksvm\_accuracy(train\_data, test\_data, "vanilladot", best\_C\_val)

## Setting default kernel parameters

test\_knn\_acc = kknn\_accuracy(as.data.frame(test\_data), best\_k\_val, "rectangular")  
print(test\_svm\_acc)

## [1] 0.8615385

print(test\_knn\_acc)

## [1] 0.8230769

## Question 4.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

The Coca-Cola Company employs clustering models when dealing with pricing and trade strategy. Focusing on and targeting populations that are more likely to offer their business is a common practice in a variety of industries, and Coca-Cola Company is no exception.

In particular, attempts to capture potential customers involve the segmentation of the consumer base, effectively categorizing and clustering customers using a number of predictors outlined below:

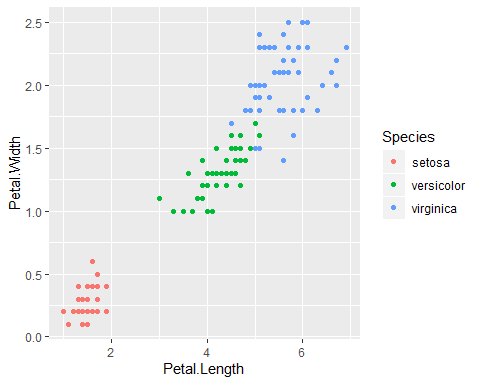
Brand Loyalty – consumers who almost exclusively purchase Coca-Cola products are labeled brand loyal. Alternatively, consumers who almost exclusively purchase competing brands are labeled competitor loyal, and consumers who frequently vary the brands they purchase are considered “Switchers”. Customers would be categorized into the distinct groupings by the frequency and total spend on products.

Price Sensitivity – another key predictor related to spend behavior, price sensitivity is a notable component of spend behavior, as customers who opt to purchase the lower-priced product regardless of brand can be categorized as switchers, and are ultimately the target demographic for Coca-Cola to target their pricing/marketing towards

When considering both predictors in clustering the consumers, a number of cluster permutations arise, but the target cluster for capturing market share would be Switchers and Competitor Loyal customers with a medium-high price sensitivity. With this knowledge in mind, the Trade Strategy/Revenue Optimization team recommends frequent sales and price points at which the Switchers or Competitor Loyals will opt to give their business to Coca-Cola.

## Question 4.2

###########################  
## K - Means Iris  
##########################  
library(datasets)  
library(ggplot2)  
#summary(iris)  
ggplot(iris, aes(Petal.Length, Petal.Width, color=Species))+geom\_point()



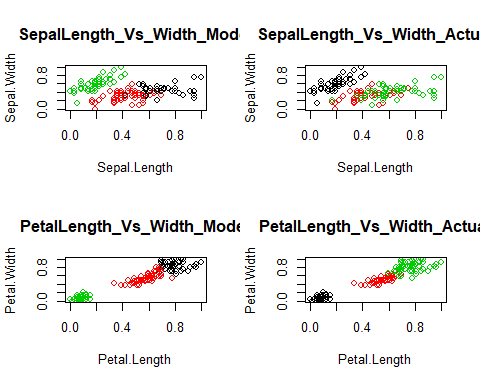
summary(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width   
## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100   
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300   
## Median :5.800 Median :3.000 Median :4.350 Median :1.300   
## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199   
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800   
## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500   
## Species   
## setosa :50   
## versicolor:50   
## virginica :50   
##   
##   
##

# Remove the species label from Iris dataset  
iris\_no\_class = iris[,-5]  
iris\_class = iris[,c("Species")]  
  
# Normalize data?  
normalize <- function(x){  
 return ((x-min(x))/(max(x)-min(x)))  
}  
  
iris\_no\_class$Sepal.Length<- normalize(iris\_no\_class$Sepal.Length)  
iris\_no\_class$Sepal.Width<- normalize(iris\_no\_class$Sepal.Width)  
iris\_no\_class$Petal.Length<- normalize(iris\_no\_class$Petal.Length)  
iris\_no\_class$Petal.Width<- normalize(iris\_no\_class$Petal.Width)  
  
kmeans\_acc = function(train\_data, k){  
 model = kmeans(train\_data,k)  
 #return(model$size)  
 par(mfrow=c(2,2))  
 plot(iris\_no\_class[c(1,2)],col=model$cluster, main="SepalLength\_Vs\_Width\_Model")  
 plot(iris\_no\_class[c(1,2)],col=iris\_class, main="SepalLength\_Vs\_Width\_Actual" )  
 plot(iris\_no\_class[c(3,4)],col=model$cluster, main="PetalLength\_Vs\_Width\_Model")  
 plot(iris\_no\_class[c(3,4)],col=iris\_class, main="PetalLength\_Vs\_Width\_Actual")  
 return(table(model$cluster,iris\_class))  
 #return(model$cluster)  
}  
  
# Test different values of k  
"  
for(i in 1:10){  
 kmeans\_acc(iris\_no\_class,i)  
}  
"

## [1] "\nfor(i in 1:10){\n kmeans\_acc(iris\_no\_class,i)\n}\n"

print(kmeans\_acc(iris\_no\_class, 3))



## iris\_class  
## setosa versicolor virginica  
## 1 0 3 36  
## 2 0 47 14  
## 3 50 0 0

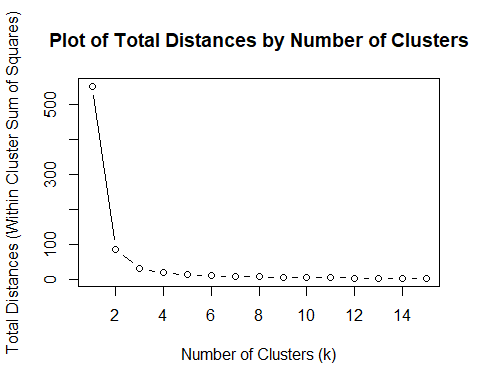
Although all four predictors (petal length & width and sepal length & width) help predict good clusters, Petal length/width are better predictors than Sepal length/width. As you can see from the plots above, the petal data helps cluster the data more distinctly and in group points are clustered more tightly. Since the data contains 3 dependant variables our a priori assumption was to use a k of 3. After looking at the empircal data (values/plots) our assumption is confirmed that a k of 3 is the best. As you can see from the code below our kmeans model with a K of 3 correctly predicts Setosa’s 100% of the time, Versicolor’s 94% of the time, and Virginica’s 72% of the time.

modelKof3 = kmeans(iris\_no\_class, 3)  
data = table(modelKof3$cluster,iris\_class)  
print(data)

## iris\_class  
## setosa versicolor virginica  
## 1 0 3 36  
## 2 0 47 14  
## 3 50 0 0

As you can see from the graph below our total data point to cluster distance can be reduced if we choose more clusters (higher k values). However this would overfit the data and lessen the accuracy. A k value other than 3 doesn’t make sense for this data.

set.seed(200)  
k.max <- 15  
#tot\_wss is a vector containing the total within-cluster sum of squares for each value of k from 1 to 15  
tot\_wss <- sapply(1:k.max, function(k){kmeans(iris[,3:4],k,nstart=20,iter.max=20)$tot.withinss})  
  
plot(1:k.max, tot\_wss, type="b", main = "Plot of Total Distances by Number of Clusters", xlab = "Number of Clusters (k)", ylab = "Total Distances (Within Cluster Sum of Squares)")



### Citations:

* <https://rpubs.com/AnanyaDu/361293>
* <http://rpubs.com/Nitika/kmeans_Iris>