ISYE 6051 : Homework 2
2/3/2021

Table of Contents

3.1 Cross Validation

a) cv.kknn

Using cv.kknn for k-fold validation with the Credit Approval data, I started with kcv = 10 to split data (standard 10 k folds per class materials).

```
#create a dataframe
matrix1 <- read.table("credit_card_data-headers.csv",header = TRUE,
sep =",")
```

```
#print out the first 10 observations
head(matrix1,n-10L)
```

```
     A    A2     A3     A8   A   A1   A1   A1   A1    A15   R
     1                       9    0    1    2    4          1
1    1  30.8  0.000   1.25   1    0    1    1  202      0   1
         3              0
2    0  58.6  4.460   3.04   1    0    6    1   43    560   1
         7              0
```

3	0	24.50	0.500	1.500	1	1	0	1	280	824	1
4	1	27.83	1.540	3.750	1	0	5	0	100	3	1
5	1	20.17	5.625	1.710	1	1	0	1	120	0	1
6	1	32.08	4.000	2.500	1	1	0	0	360	0	1
7	1	33.17	1.040	6.500	1	1	0	0	164	31285	1
8	0	22.92	11.585	0.040	1	1	0	1	80	1349	1
9	1	54.42	0.500	3.960	1	1	0	1	180	314	1
10	1	42.50	4.915	3.165	1	1	0	0	52	1442	1

```
#Starting number
set.seed(1)
```

```
# test multiple k values to determine the best accuracy
# set the total maximum value of k
kbestvalue <= 75
```

```
# create a vector to store the predictions
predicted_data <- rep(0,(nrow(matrix1)))

#accuracy vector for the training model cv.kknn
accumulated_data <- c()

#kcv = 10 for 10 k folds
```

To determine the best value for k (nearest neighbor), I used a loop calculating the best prediction accuracy for k = 1 to 75. Using a k that

is too large adds additional iterations and computational time and as noted by the data output, there is little variance in the prediction accuracy values.

```
for (i in 1:kbestvalue){
  modelcv <- cv.kknn(R1~.,
                     matrix1,
                     kcv=10,
                     k=i,
                     scale=TRUE)
  predicted_data <- round(modelcv[[1]][,2])
  accumulated_data <- c(accumulated_data,sum(predicted_data ==
matrix1[,11])/nrow(matrix1))
}
```

```
> print("Prediction Accuracy Values")
[1] "Prediction Accuracy Values"
> print(accumulated_data, digits = 4, justify = “right”)
```

Prediction Accuracy Values kcv = 10

```
  [1]  0.7982  0.8058  0.8043  0.8104  0.8052  0.8563  0.8410  0.8425  0.8471  0.8486  0.8578
 [12]  0.8547  0.8471  0.8517  0.8593  0.8456  0.8624  0.8379  0.8364  0.8242  0.8425  0.8502
 [23]  0.8425  0.8379  0.8364  0.8394  0.8287  0.8425  0.8471  0.8379  0.8349  0.8440  0.8364
 [34]  0.8410  0.8349  0.8440  0.8364  0.8318  0.8364  0.8257  0.8364  0.8379  0.8318  0.8440
 [45]  0.8425  0.8333  0.8364  0.8471  0.8333  0.8425  0.8440  0.8318  0.8379  0.8318  0.8318
 [56]  0.8394  0.8394  0.8440  0.8410  0.8379  0.8379  0.8379  0.8440  0.8318  0.8349  0.8486
 [67]  0.8349  0.8440  0.8349  0.8379  0.8349  0.8410  0.8364  0.8364  0.8410  0.8379  0.8410
 [78]  0.8440  0.8333  0.8502  0.8394  0.8287  0.8394  0.8471  0.8456  0.8440  0.8425  0.8456
 [89]  0.8440  0.8410  0.8471  0.8410  0.8394  0.8456  0.8333  0.8318  0.8303  0.8394  0.8471
[100]  0.8410
```

Using which.max () and max () I can determine that k = 17 has the greatest prediction accuracy at 86%.

```
which.max(accumulated_data)
max(accumulated_data)
```

```
[1] 17
```

```
[1] 0.8623853
```

Note: I also changed kcv = 654 which is equivalent to leave-one-out cross validation. Significant differences were not found at 85% for the maximum prediction accuracy of k. The computational time was considerably longer for a 1% decrease in the maximum prediction accuracy.

```
kcvvalue = 654

for (i in 1:kbestvalue){
  modelcv <- cv.kknn(R1~.,
                    matrix1,
                    kcv=kcvvalue,
                    k=i,
                    scale=TRUE)
  predicted_data <- round(modelcv[[1]][,2])
  accumulated_data <- c(accumulated_data,sum(predicted_data ==
matrix1[,11])/nrow(matrix1))
}
```

Prediction Accuracy Values kcv = 654

```
  [1]  0.8150  0.8150  0.8150  0.8150  0.8517  0.8456  0.8471  0.8486  0.8471  0.8502  0.8517
 [12]  0.8532  0.8517  0.8517  0.8532  0.8517  0.8517  0.8517  0.8502  0.8486  0.8471  0.8471
 [23]  0.8471  0.8456  0.8456  0.8456  0.8410  0.8379  0.8394  0.8410  0.8379  0.8364  0.8333
 [34]  0.8349  0.8349  0.8318  0.8318  0.8333  0.8318  0.8318  0.8287  0.8333  0.8303  0.8379
 [45]  0.8394  0.8394  0.8379  0.8394  0.8394  0.8364  0.8379  0.8379  0.8379  0.8394  0.8410
 [56]  0.8394  0.8304  0.8304  0.8364  0.8379  0.8379  0.8364  0.8333  0.8349  0.8333  0.8364
 [67]  0.8349  0.8394  0.8318  0.8379  0.8379  0.8379  0.8364  0.8364  0.8394  0.8379  0.8379
 [78]  0.8364  0.8364  0.8364  0.8379  0.8364  0.8349  0.8349  0.8364  0.8394  0.8379  0.8379
 [89]  0.8364  0.8379  0.8379  0.8379  0.8425  0.8410  0.8394  0.8394  0.8379  0.8364  0.8364
[100]  0.8379  0.8440
```

```
> which.max(accumulated_data)
[1] 12
> max(accumulated_data)
[1] 0.853211
```

Utilizing cv.kknn did not necessarily prove what the best model is for training. The first part of this question did select the best k-fold value for training this data and creating a cv.kknn model that is computationally efficient.

<u>b) Splitting Data</u>

I used the sample function to split data into train, validation and test groupings. Sample was used to generate randon assignments to the train, validation and test datasets versus a 1:n type of assignment.

```
#Starting number to ensure consistency
set.seed(1)

#use the sample function to create a random sample of 3 sizes
partitiondata <- sample(1:3,
                        size=nrow(matrix1),
                        prob=c(0.7, 0.15, 0.15),
                        replace = TRUE)

#assign the percentage of sampling based on the percentages in the
vector prob
train <- matrix1[partitiondata == 1,]
trainnum <- nrow(train)

validation <- matrix1[partitiondata == 2,]
validationnum <- nrow(validation)

testdata <- matrix1[partitiondata == 3,]
testnum <- nrow(testdata)
```

Training Dataset (I have displayed a limited amount of data.)

```
cat("Number of training data observations:", trainnum)
Number of training data observations: 459
   A1    A2     A3    A8 A9 A10 A11 A12 A14  A15 R1
1   1 30.83  0.000 1.250  1   0   1   1 202    0  1
2   0 24.50  0.500 1.500  1   1   0   1 280  824  1
3   0 22.92 11.585 0.040  1   1   0   1  80 1349  1
5   1 20.17  5.625 1.710  1   1   0   1 120    0  1
8   0 22.92 11.585 0.040  1   1   0   1  80 1349  1
9   1 54.42  0.500 3.960  1   1   0   1 180  314  1
10  1 42.50  4.915 3.165  1   1   0   0  52 1442  1
11  1 22.08  0.830 2.165  0   1   0   0 128    0  1
12  1 29.92  1.835 4.335  1   1   0   1 260  200  1
13  0 38.25  6.000 1.000  1   1   0   0   0    0  1
14  1 48.08  6.040 0.040  0   1   0   1   0 2690  1
16  1 36.67  4.415 0.250  1   0  10   0 320    0  1
19  1 21.83  0.250 0.665  1   1   0   0   0    0  1
22  1 23.25  1.000 0.835  1   1   0   1 300    0  1
23  0 47.75  8.000 7.875  1   0   6   0   0 1260  1
```

Validation Dataset (I have displayed a limited amount of data)

```
cat("Number of validation data observations:", validationnum)
Number of validation data observations: 100
   A1    A2     A3    A8 A9 A10 A11 A12 A14   A15 R1
4   1 27.83  1.540 3.750  1   0   5   0 100     3  1
6   1 32.08  4.000 2.500  1   1   0   0 360     0  1
7   1 33.17  1.040 6.500  1   1   0   0 164 31285  1
18  0 23.25  5.875 3.170  1   0  10   1 120   245  1
21  1 25.00 11.250 2.500  1   0  17   1 200  1208  1
29  1 57.42  8.500 7.000  1   0   3   1   0     0  1
```

```
 52  1 26.00  1.000  1.750 1   1   0   0 280      0   1
 61  1 56.75 12.250  1.250 1   0   4   0 200      0   1
 70  1 35.17 25.125  1.625 1   0   1   0 515    500   1
 76  1 34.08  6.500  0.125 1   1   0   0 443      0   0
 77  0 19.17  0.585  0.585 1   1   0   0 160      0   0
 80  1 49.58 19.000  0.000 1   0   1   1  94      0   0
 94  0 22.50 11.000  3.000 1   1   0   0 268      0   0
104  1 40.92  2.250 10.000 1   1   0   0 176      0   0
109  0 33.75  0.750  1.000 1   0   3   0 212      0   0
```

Test Dataset (I have displayed a limited amount of data)

```
cat("Number of testing data observations:", testnum)
```

```
Number of testing data observations: 95
   A1    A2     A3     A8 A9 A10 A11 A12   A14   A15 R1
15  0 45.83 10.500  5.000  1   0   7   0     0     0  1
17  1 28.25  0.875  0.960  1   0   3   0   396     0  1
20  0 19.17  8.585  0.750  1   0   7   1    96     0  1
35  0 22.58 10.750  0.415  1   0   5   0     0   560  1
37  1 27.25  1.585  1.835  1   0  12   0 0.583   713  1
39  1 27.25  0.585  0.250  1   0   2   1   260   500  1
41  1 34.17  9.170  4.500  1   0  12   0     0   221  1
43  1 29.67  1.415  0.750  1   0   1   1   240   100  1
46  1 54.33  6.750  2.625  1   0  11   0     0   284  1
49  1 41.50  1.540  3.500  0   1   0   1   216     0  1
68  1 25.50  0.375  0.250  1   0   3   1   260 15108  1
72  0 38.58  5.000 13.500  1   1   0   0   980     0  0
79  1 21.50  9.750  0.250  1   1   0   1   140     0  0
82  1 39.83  0.500  0.250  1   1   0   1   288     0  0
85  1 25.67  2.210  4.000  1   1   0   1   188     0  0
```

c) Using Split Data with kknn

Next, I will use this split data to validate using the kknn model. Kbestvalues once again is equal to 75 in order to determine the maximum prediction accuracy using k. Similar to k fold cross validation using cv.kknn, the maximum prediction accuracy falls at k = 26, 84%. K is relatively low at k<5. K = 10 (general k fold value used) is 83% which is very close to the maximum prediction accuracy k of 26.

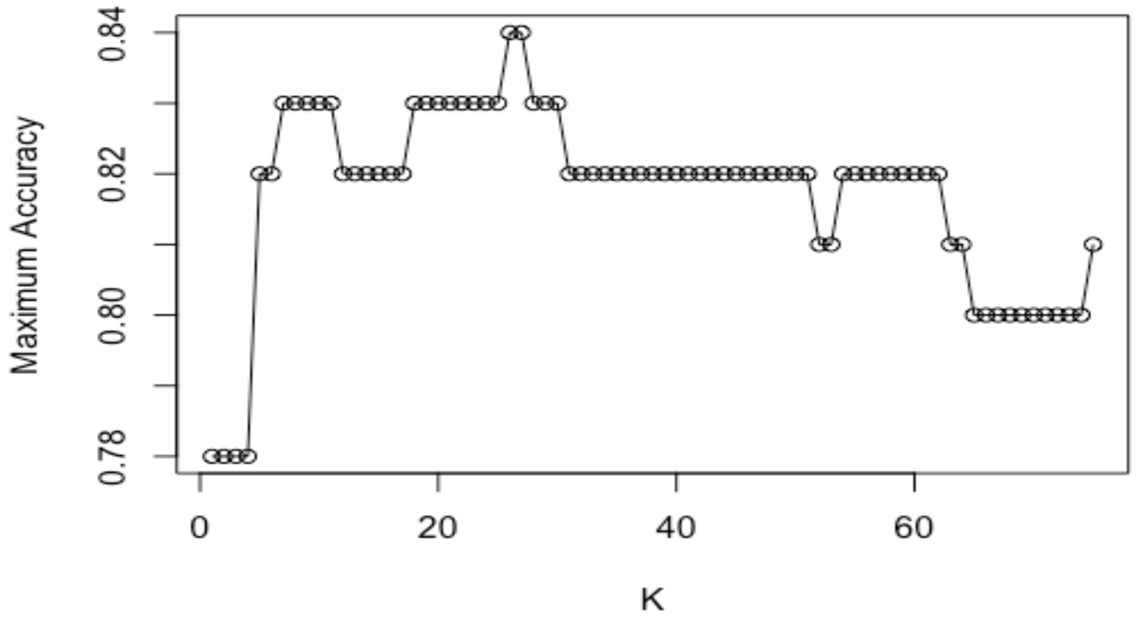
```
for (i in 1:kbestvalue){
  modelkknn <- kknn(R1~.,
                  train,
                  validation,
                  k=i,
                  scale=TRUE)
  predicted_data <- round(modelkknn$fitted.values)
  accumulated_data <- c(accumulated_data,sum(predicted_data ==
validation[,11])/nrow(validation))
}
```

```
[1] "Prediction Accuracy Values"
> print(accumulated_data, digits = 4, justify = "right")
```

Prediction Accuracy Values for kknn with split data

```
 [1]  0.78  0.78  0.78  0.78  0.82  0.82  0.83  0.83  0.83  0.83  0.83  0.82  0.82  0.82  0.82
[16]  0.82  0.82  0.83  0.83  0.83  0.83  0.83  0.83  0.83  0.83  0.84  0.84  0.83  0.83  0.83
[31]  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82
[46]  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82  0.82
[61]  0.82  0.82  0.81  0.81  0.80  0.80  0.80  0.80  0.80  0.80  0.80  0.80  0.80  0.80  0.81
```

```
> which.max(accumulated_data)
[1] 26
> max(accumulated_data)
[1] 0.84
```



Finally, I used the best k for the maximum predicted accuracy with the test data for kknn.

```
modelkknntest <- kknn(R1~.,
                  train,
                  test,
                  k=which.max(accumulated_data),
                  scale = TRUE)
predicted_data <-round(modelkknntest$fitted.values)
accumulated_data2 <- sum(predicted_data == test[,11])/nrow(test)
```

```
cat("Using the best k value of", which.max(accumulated_data), " the
maximum prediction accuracy is:", (max(accumulated_data) * 100), "%")
```

```
Using the best k value of 26  the maximum prediction accuracy is: 84 %
```

Cross validation in 3.1 appears to work slightly better than splitting the data with kknn. The difference is small though. Having a significantly larger data set may show the value in utilizing cross fold validation modeling versus modeling with split data.

4.1 Example of Clustering

The availability of community healthcare facilities in areas of the United States for both preventative medicine and to address care of major comorbidities has always been a concern for individuals living in underserved rural and urban centers. Lack of investment, low transportation options and communication have served to fuel a healthcare crisis where small medical issues turn into major medical emergencies. Determining what locations can service the most individuals in a community is an example of clustering similar data points. My focus would be on identifying commonality using the 5 following predictors:

1. Availability of transportation – lack of transportation inhibits individuals from getting to healthcare facilities outside of their community.
    a. Vehicle ownership
    b. Mass transit taking into account frequency of offerings
2. Distance to nearest healthcare facility – longer distances may impede going to a healthcare facilitiy.
    a. Time off work
    b. Commute time
3. Large concentrations of individuals who can be served.
4. Numbers of individuals with conditions of comorbidity.
5. Amount of investment interest – federal, state and city interest in establishing community healthcare facilities can provide funding sources.

4.2 Using kmeans in clustering

The first step is to view the Iris dataset. The current format is supervised since classification of the petal data points has already taken place as well as ordering of the classifications. K means clustering is designed to determine classifications of unsupervised data. A new dataframe is created using the non-classified data points.

```
install.packages("seasonal")
install.packages("dplyr")
library(stats)
library(dplyr)

View(iris)
```

Iris Dataset (I did not display all of the data for brevity)

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa

```
# To find clustering we need to look at data without classification or
tagging
datapoints = select(iris,c(1,2,3,4))
View(datapoints)
```

Sample of the extracted data points (no classification or tagging established)

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5.0	3.6	1.4	0.2

Kmeans is based on Euclidean distance (lecture and additional reading materials). It is important to normalize the data in order to minimize differences in distance that would make the kmeans algorithm inefficient.

Summary of datapoints prior to normalization

```
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
```

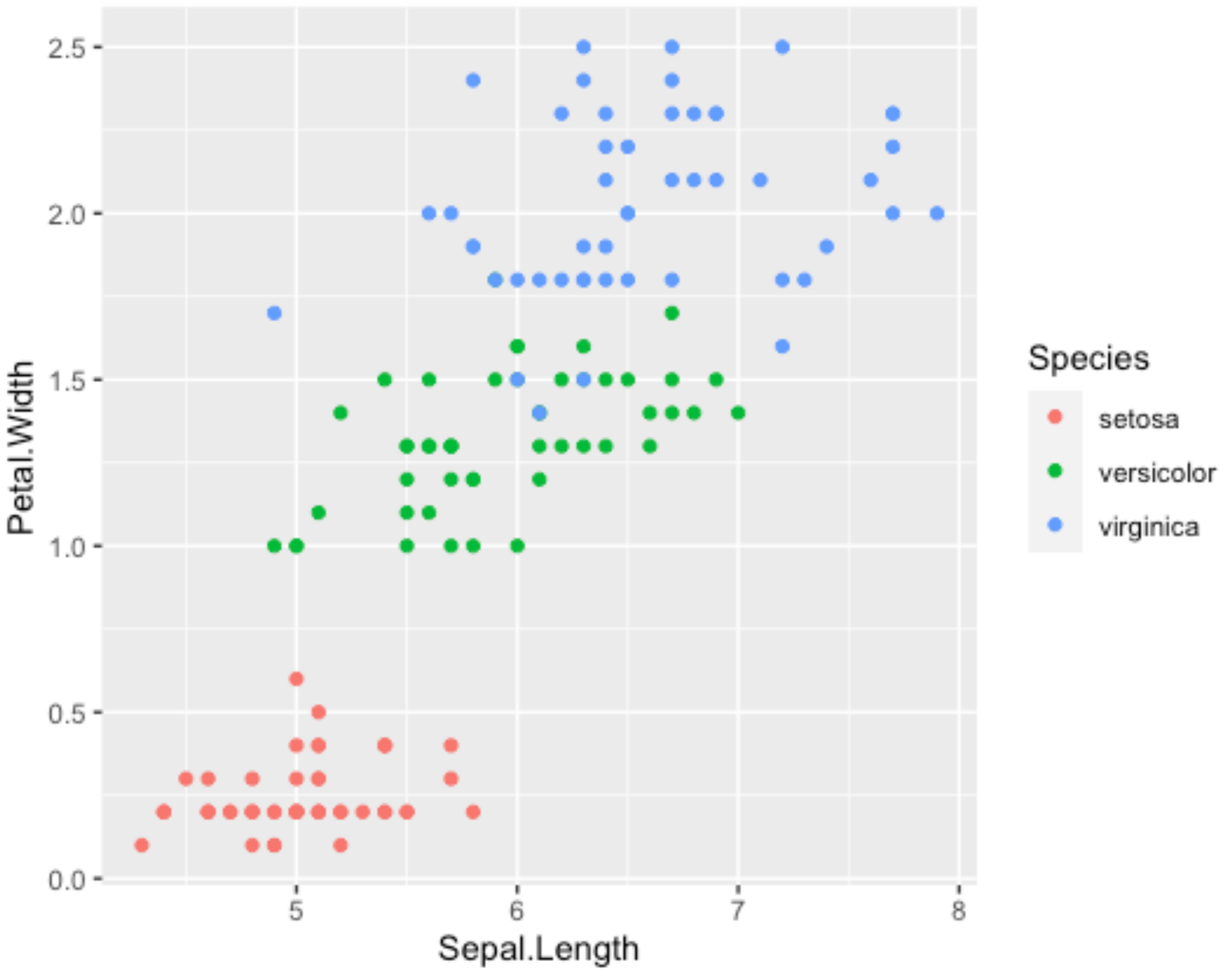
```
normalize = function(x){
  return((x - min(x, na.rm = FALSE))/(max(x, na.rm = FALSE) - min(x, na.rm
= FALSE)))
}

datapoints$Sepal.Length<-(normalize(datapoints$Sepal.Length))
datapoints$Sepal.Width<-(normalize(datapoints$Sepal.Width))
datapoints$Petal.Length<-(normalize(datapoints$Petal.Length))
datapoints$Petal.Width<-(normalize(datapoints$Petal.Width))
```

Summary of datapoints after normalization

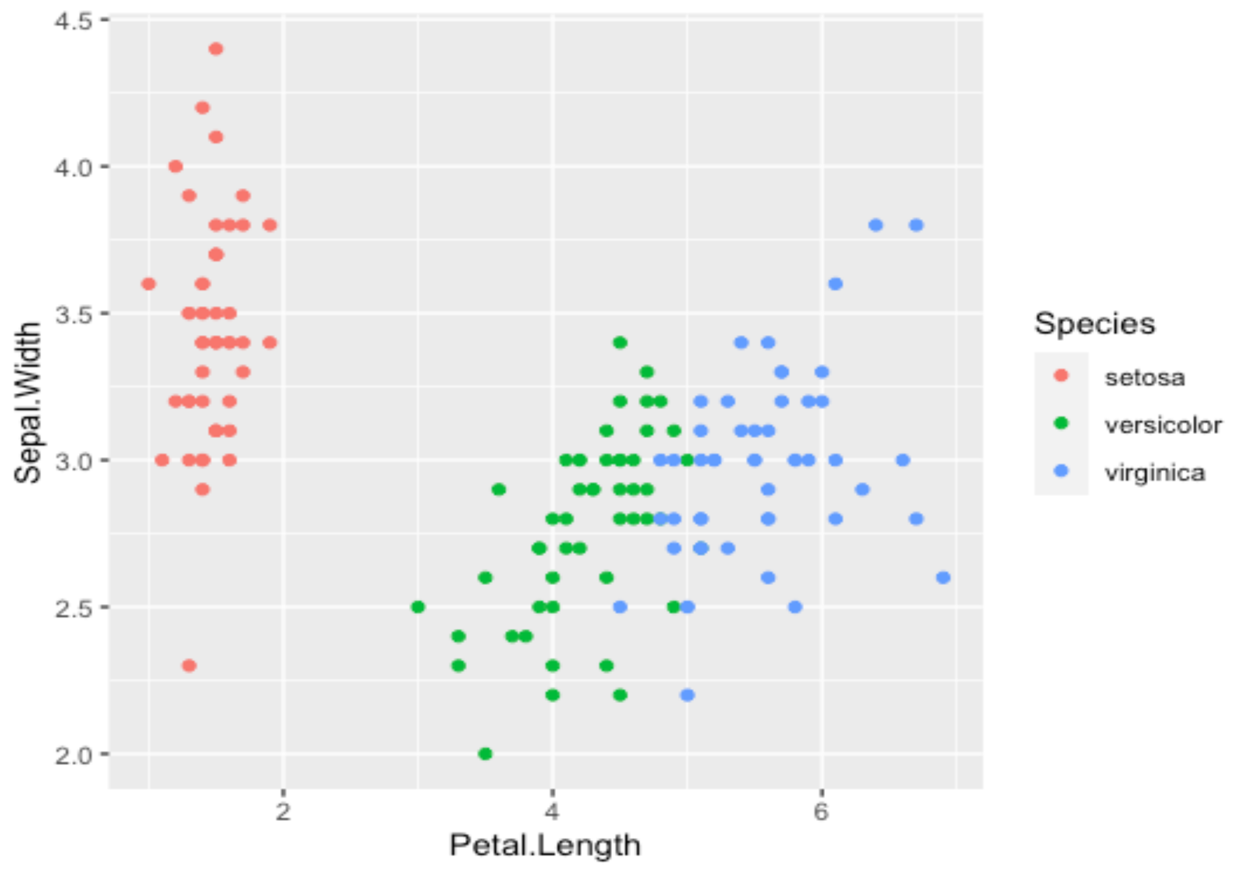
```
 Sepal.Length       Sepal.Width        Petal.Length       Petal.Width
 Min.   :0.0000     Min.   :0.0000     Min.   :0.0000     Min.   :0.0000
 1st Qu.:0.2222     1st Qu.:0.3333     1st Qu.:0.1017     1st Qu.:0.08333
 Median :0.4167     Median :0.4167     Median :0.4167     Median :0.4167
 Mean   :0.4287     Mean   :0.4287     Mean   :0.4287     Mean   :0.4287
 3rd Qu.:0.5833     3rd Qu.:0.5833     3rd Qu.:0.5833     3rd Qu.:0.5833
 Max.   :1.0000     Max.   :1.0000     Max.   :1.0000     Max.   :1.0000
```

As an initial visualization tool (and at the suggestion of team members), I plotted the original iris data in order to determine if there were obvious clusters to begin the evaluation using ggplot. I tried 4 different variations of length and width for sepal and petal. I am only showing 2 to demonstrate the consistency in output on the number of clusters.

```
ggplot(iris , aes(Sepal.Length, Petal.Width, color = Species)) +
geom_point()
```



```
ggplot(iris , aes(Petal.Length, Sepal.Width, color = Species)) +
geom_point()
```



```
data3 <- kmeans(datapoints,centers = 3, nstart = 20)
data3$centers
```

The means for each column of data of the clusters is:

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1    0.7072650   0.4508547   0.79704476  0.82478632
2    0.1961111   0.5950000   0.07830508  0.06083333
3    0.4412568   0.3073770   0.57571548  0.54918033
```

The clustering vector is:

```
data3$cluster
```

```
[1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [39] 2 2 2 2 2 2 2 2 2 2 2 2 1 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [77] 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1 1 1 3
[115] 1 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 3 1 1 1 3 1 1 1 3 1 1 1 3 1 1 3
```

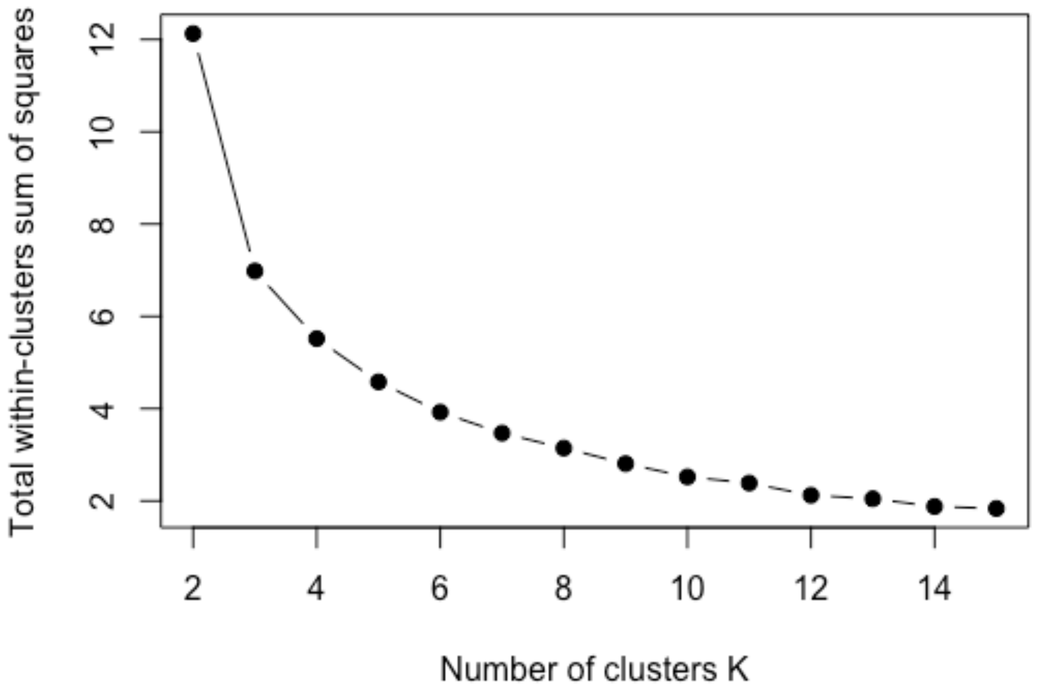
Size Cluster 1: 37
Size Cluser 2 : 50
Size Cluster 3: 61

The value of the kmeans clustering algorithm is that it assumes that there aren't classifiers, so utilizing the normalized data, I created an elbow diagram to evaluate the optimal K and determine if my earlier assumptions on the numbers of clusters was accurate.

```
#draw an elbow plot as if I was really using untagged data where I didn't
#have an initial understanding of what the clusters could actually be
#loading 2 additional libraries to aide in the wss calculations and plotting
formats

library(cluster)
install.packages("factoextra")
library(factoextra)

#determine number of clusters to use
k.max<- 15
wss <- sapply(2:k.max, function(k){kmeans(datapoints, k, nstart=20
)$tot.withinss})
print(wss)
```

```
[1] 12.127791  6.982216  5.516933  4.580323  3.923095  3.468937
3.144949
 [8]  2.806762  2.521014  2.387051  2.122243  2.047460  1.878915
1.835915
```

```
fviz_nbclust(datapoints, kmeans, method = "wss") +
geom_vline(xintercept = 3, linetype = 2)
```



The optimal k is once again reflected at 3 in the elbow diagram. The kmeans analysis is designed to perform analysis on observations to place them in the correct clusters for use in predictive modeling.