ISYE6501 Homework 2

1/20/2019

Contents

Question 3.1

(a) cv

Find a good classifier using ksvm or kknn, include cross-validation (a) and splitting the data sets (b).

First try the cross validation with kknn

```
library(kknn)
rm(list = ls())
set.seed(42) # the answer of life
ccdata <- read.delim("~/Documents/R/GeorgiaTech/Validation/credit_card_data-headers.txt")

# preview data
head(ccdata)
```

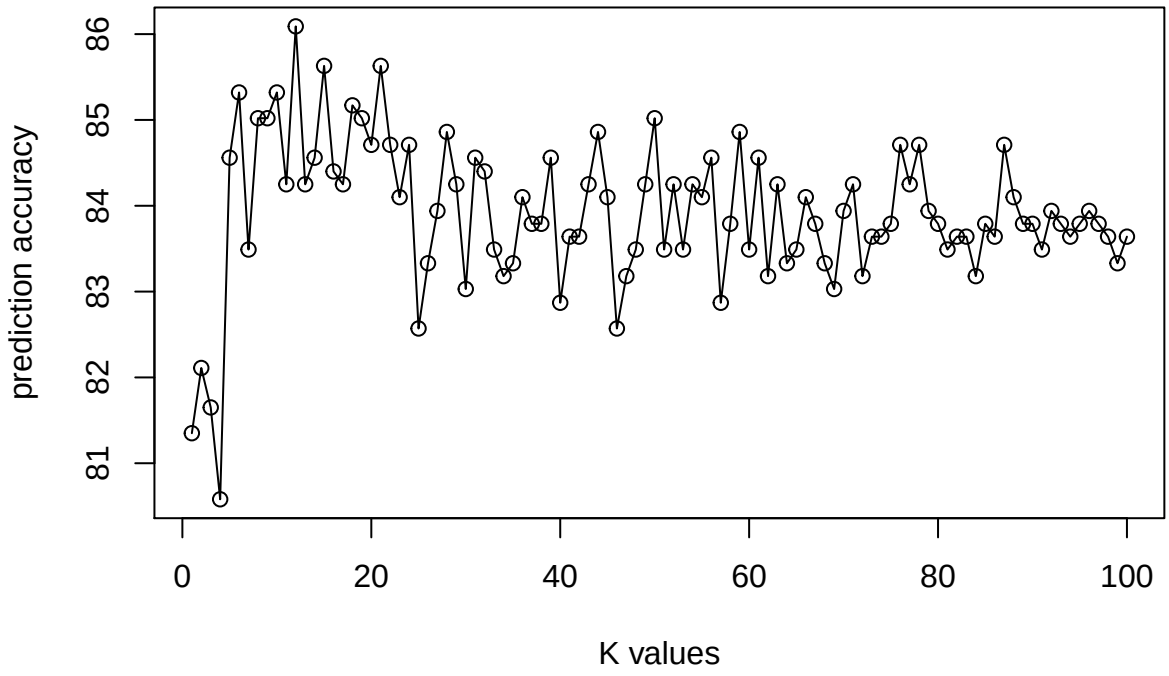
```
##   A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
## 1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
## 2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
## 3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
## 4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
## 5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
## 6  1 32.08 4.000 2.50  1   1   0   0 360   0  1
```

```
# test multiple k-values
max_k <- 100
max_k_folds <- 20
# vector to save predictions
accuracy <- c()
# now calculate the accuracy for kmax neighbors
for (neighbor in 1:max_k) {
    # this package includes cross validation
    model <- cv.kknn(R1~.,ccdata, kcv=10, # use the common k-folds (Dr. Sokol suggested it)
            k=neighbor,
            scale=TRUE)
    # round it to make it comparable
    pred <- round(model[[1]][,2])
```

```
    # show as percentage
    accuracy <- c(accuracy, round(sum(pred == ccdata$R1)/nrow(ccdata), digits = 4)*100)
}
# show accuracy for each k
print (accuracy)
```

```
##   [1] 81.35 82.11 81.65 80.58 84.56 85.32 83.49 85.02 85.02 85.32 84.25
##  [12] 86.09 84.25 84.56 85.63 84.40 84.25 85.17 85.02 84.71 85.63 84.71
##  [23] 84.10 84.71 82.57 83.33 83.94 84.86 84.25 83.03 84.56 84.40 83.49
##  [34] 83.18 83.33 84.10 83.79 83.79 84.56 82.87 83.64 83.64 84.25 84.86
##  [45] 84.10 82.57 83.18 83.49 84.25 85.02 83.49 84.25 83.49 84.25 84.10
##  [56] 84.56 82.87 83.79 84.86 83.49 84.56 83.18 84.25 83.33 83.49 84.10
##  [67] 83.79 83.33 83.03 83.94 84.25 83.18 83.64 83.64 83.79 84.71 84.25
##  [78] 84.71 83.94 83.79 83.49 83.64 83.64 83.18 83.79 83.64 84.71 84.10
##  [89] 83.79 83.79 83.49 83.94 83.79 83.64 83.79 83.94 83.79 83.64 83.33
## [100] 83.64
```

```
# see how accuracy behaves
plot(accuracy, ylab="prediction accuracy",xlab="K values")
lines(accuracy)
```




```
# this is used to print a table in 4 columns
half <- max_k/2
matrix <- cbind(seq(1,half), accuracy[1:half],seq((half+1), max_k), accuracy[(half+1):50])
colnames(matrix) <- c("k", "accuracy %", "k", "accuracy %")
library(knitr) # table library
table <- kable(matrix)
print (table)
```

```
##
##
##   k   accuracy %     k   accuracy %
## ---  -----------  ----  -----------
```

```
##   1        81.35    51        83.49
##   2        82.11    52        85.02
##   3        81.65    53        83.49
##   4        80.58    54        85.02
##   5        84.56    55        83.49
##   6        85.32    56        85.02
##   7        83.49    57        83.49
##   8        85.02    58        85.02
##   9        85.02    59        83.49
##  10        85.32    60        85.02
##  11        84.25    61        83.49
##  12        86.09    62        85.02
##  13        84.25    63        83.49
##  14        84.56    64        85.02
##  15        85.63    65        83.49
##  16        84.40    66        85.02
##  17        84.25    67        83.49
##  18        85.17    68        85.02
##  19        85.02    69        83.49
##  20        84.71    70        85.02
##  21        85.63    71        83.49
##  22        84.71    72        85.02
##  23        84.10    73        83.49
##  24        84.71    74        85.02
##  25        82.57    75        83.49
##  26        83.33    76        85.02
##  27        83.94    77        83.49
##  28        84.86    78        85.02
##  29        84.25    79        83.49
##  30        83.03    80        85.02
##  31        84.56    81        83.49
##  32        84.40    82        85.02
##  33        83.49    83        83.49
##  34        83.18    84        85.02
##  35        83.33    85        83.49
##  36        84.10    86        85.02
##  37        83.79    87        83.49
##  38        83.79    88        85.02
##  39        84.56    89        83.49
##  40        82.87    90        85.02
##  41        83.64    91        83.49
##  42        83.64    92        85.02
##  43        84.25    93        83.49
##  44        84.86    94        85.02
##  45        84.10    95        83.49
##  46        82.57    96        85.02
##  47        83.18    97        83.49
##  48        83.49    98        85.02
##  49        84.25    99        83.49
##  50        85.02   100        85.02
```

Here we can look the accuracy for each K. The graph shows that accuracy with $k < 5$ are the worst and then it behaves similarly. There are multiple k values, that have a small accuracy difference. After $k > 20$ accuracy seems to decrease (also look plot).Cross validation helps you make better use of your data, and the

averaging process helps give a better estimate of model quality. Cross validation is not used to pick a model but to make better use of the data and get a better estimate of model quality.

Our model determines that the best k is the following one

```
## [1] 12
```

With an accuracy of

```
## [1] "86.09 %"
```

The kknn tells us that the 12 nearest neighboors help to classify/cluster with `86.09%` accuracy. When I compare the kknn in Homework 1 this accuracy is minimal better and the k-value is the same. Also I tested the model with different k-folds, I'm not leaving the proof of this because it will be too long, but the result accuracy is the same but the k varies. In this case the best k is affected by the k-folds/groups selected but this is not relevant. According to wikipedia `10-fold cross-validation is commonly used, but in general k remains an unfixed parameter.`

Finally I learned that the kknn in Homework 1 did also cross validation k = 654. When k = n (the number of observations), the k-fold cross-validation is exactly the leave-one-out cross-validation.

(b) Now we split the data (kknn)

```
# another student suggested this library to split the data, so let's try it
library(caTools)
library(kernlab)
rm(list = ls())
set.seed(42) # the answer of life
ccdata <- read.delim("~/Documents/R/GeorgiaTech/Validation/credit_card_data-headers.txt")
# split 75% training, 7.5% validation and 7.5% test
sample = sample.split(ccdata, SplitRatio = 0.75)
train = subset(ccdata, sample == TRUE)
remaining  = subset(ccdata, sample == FALSE)
# split in half
sample2 = sample.split(remaining, SplitRatio = .50)
validation = subset(remaining, sample2 == TRUE)
test = subset(remaining, sample2 == FALSE)

# verify

is_equals <- nrow(test) + nrow(validation) + nrow(train) == nrow(ccdata)
print (is_equals) # is valid
```

```
## [1] TRUE
```

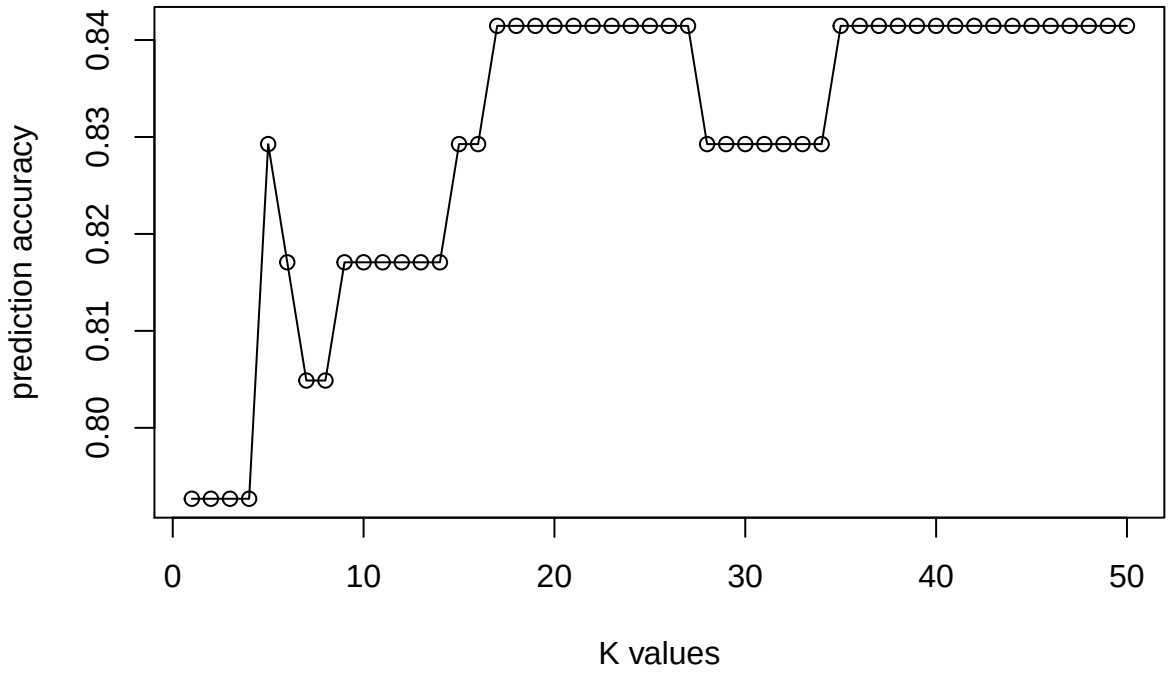
```
#pick first KNN
accuracy <- c()
for (k in 1:50) {

  # model knn using training and validation
  model_knn <- kknn(R1~.,train,validation,k=k,scale=TRUE)

    #  compare with validation
  prediction <-  round(model_knn$fitted.values)
  accuracy<-c(accuracy,sum(prediction == validation$R1) / nrow(validation))
```

```
}
print (paste("Best model is", which.max(accuracy)))
```

```
## [1] "Best model is 17"
```

```
print (paste("Best accuracy is ", max(accuracy)*100, "%"))
```

```
## [1] "Best accuracy is  84.1463414634146 %"
```




```
# now compare quality with test data
model_knn <- kknn(R1~.,train,test, k=which.max(accuracy), scale=TRUE)
# round to make it comparable
pred <- round(model_knn$fitted.values)
# calculate performance
ac <- sum(pred == test$R1) / nrow(test)
print (paste("KNN performance on test data is ",ac*100, "%"))
```

```
## [1] "KNN performance on test data is  87.7551020408163 %"
```

(b) now compare it with the SVM

The valid way to compare it is against the performance on the test

```
# make vector with c values to test
C_values <- c(10^(-7:7))
acc_svm <- c()
for (i in 1:15) {

    # fit model using training set
  model_ksvm <- ksvm(train$R1~., data=train, type = "C-svc", kernel = "vanilladot",
  C = C_values[i],
  scaled=TRUE)
```

```
    #  accuracy model using validation set
  pred <- predict(model_ksvm, validation[,1:10])
  acc_svm <- c(acc_svm, sum(pred == validation$R1) / nrow(validation))
}
```

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

```
print (paste("Best C ksvm ",C_values[which.max(acc_svm)]))
```

```
## [1] "Best C ksvm  0.001"
```

```
print (paste("KSVM quality on validation data  ",max(acc_svm)*100))
```

```
## [1] "KSVM quality on validation data   86.5853658536585"
```

```
model_scaled <- ksvm(train$R1~., data=train,
     type = "C-svc",
  kernel = "vanilladot", #could test with other non linear kernel
  C = C_values[which.max(acc_svm)], #pick the best model
     scaled=TRUE)
```

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

```
# estimate real quality/performance
ac <- sum(predict(model_scaled,test[,1:10]) == test$R1) / nrow(test)
print (paste("KSVM Quality on test data is ",ac*100, "%"))
```

```
## [1] "KSVM Quality on test data is  80.6122448979592 %"
```

We can say that the ksvm has better performance on validation data, and we can't use the test performance to compare the models because if we use the same data to pick the best model as we do to estimate how good the best one is, the model will appear to be better than it really is.

So we actually pick the `ksvm` as the best (validation data) and the estimated performance of the model is `80.61%` (test data).

Question 4.1

In my current work (telephone company) we use clustering algorithms to identify which cellphone numbers are related to the same family. In this case each family is a cluster we want to identify. For our company it is valuable to know this, so we can make specialized offers to the members of the family and also it is used to

contact only one of them instead of randomly call all the members of the family (we don't want to spam our 9 million users). The predictors we currently use are the following.

1. The location at night (we suppose that people that sleeps in the same location can be family)
2. The most numbers they have calls in. (We have this as a relative continous variable in the database)
3. The most numbers they have calls out.(We have this as a relative continous variable in the database.)
4. The owners of the cellphone line (we suppose that children (<18) have phones and their parents are the owners, also as a relative continous variable)
5. The most SMS sent (as relative continous variable, we suppose that family text each other frequently)

Question 4.2

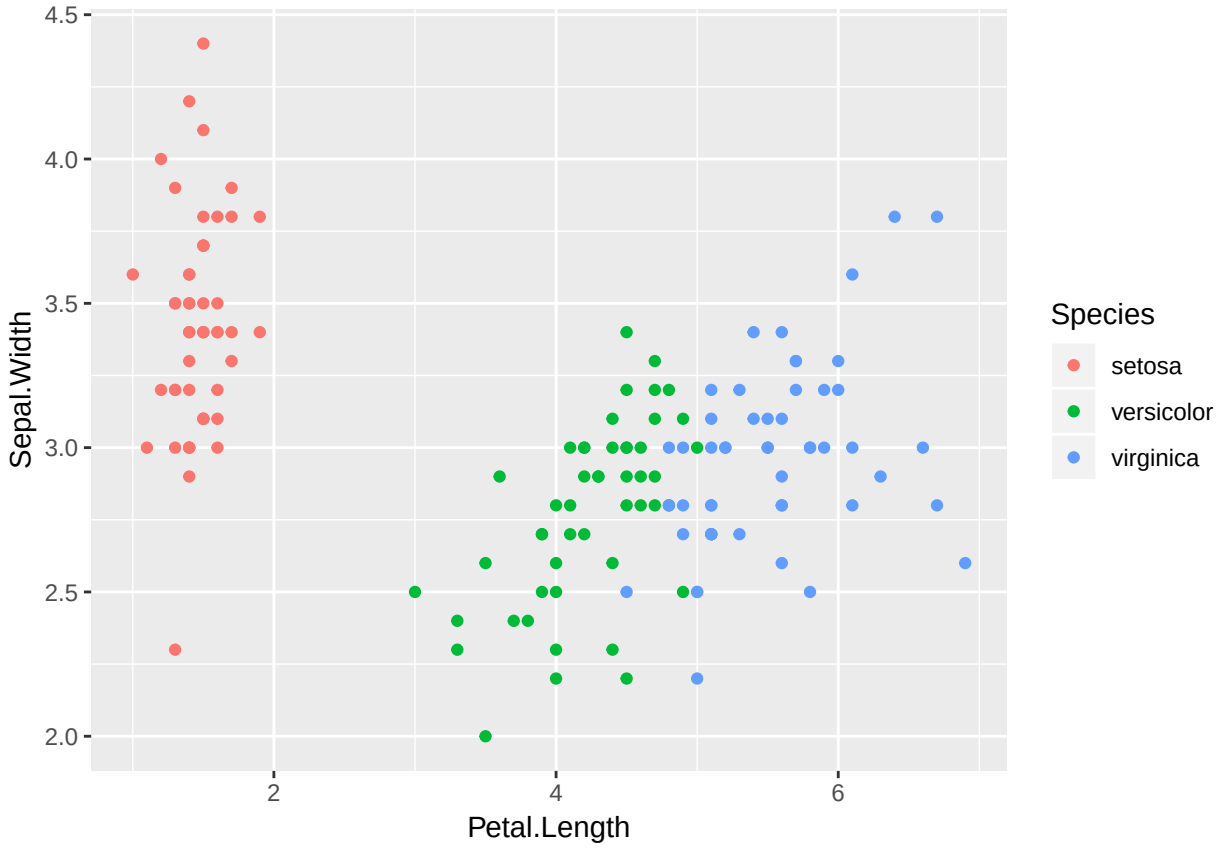
Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

```
rm(list = ls())
set.seed(42) # the answer of life
# default dataset in R, this has the response variable but it should be used only to compare
data <- iris
# according to the TA we can plot the data to have an idea how we can cluster
library(ggplot2)
```

```
##
## Attaching package: 'ggplot2'
```

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

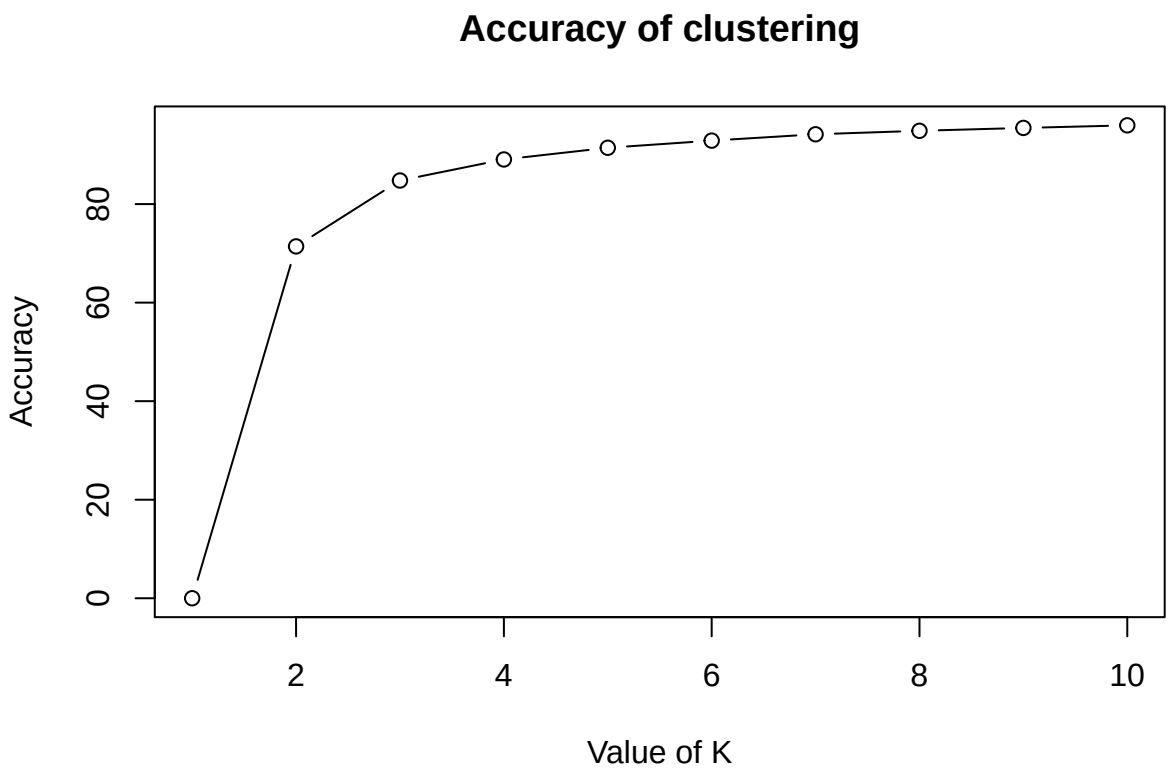
```
normalize <- function(x){
  return ((x-min(x))/(max(x)-min(x)))
}
# we should scale the data, mean = 0 and stdev = 1
data.scaled<- data[,c(1,2,3,4)]
data.scaled$Sepal.Length<- normalize(data.scaled$Sepal.Length)
data.scaled$Sepal.Width<- normalize(data.scaled$Sepal.Width)
data.scaled$Petal.Length<- normalize(data.scaled$Petal.Length)
data.scaled$Petal.Width<- normalize(data.scaled$Petal.Width)
# after testing a few plots, this give us 3 good potential groups
#ggplot(data , aes(Petal.Length, Petal.Width, color = Species)) + geom_point()
#ggplot(data , aes(Sepal.Length, Sepal.Width, color = Species)) + geom_point()
#ggplot(data , aes(Sepal.Length, Petal.Width, color = Species)) + geom_point()
ggplot(data , aes(Petal.Length, Sepal.Width, color = Species)) + geom_point()
```



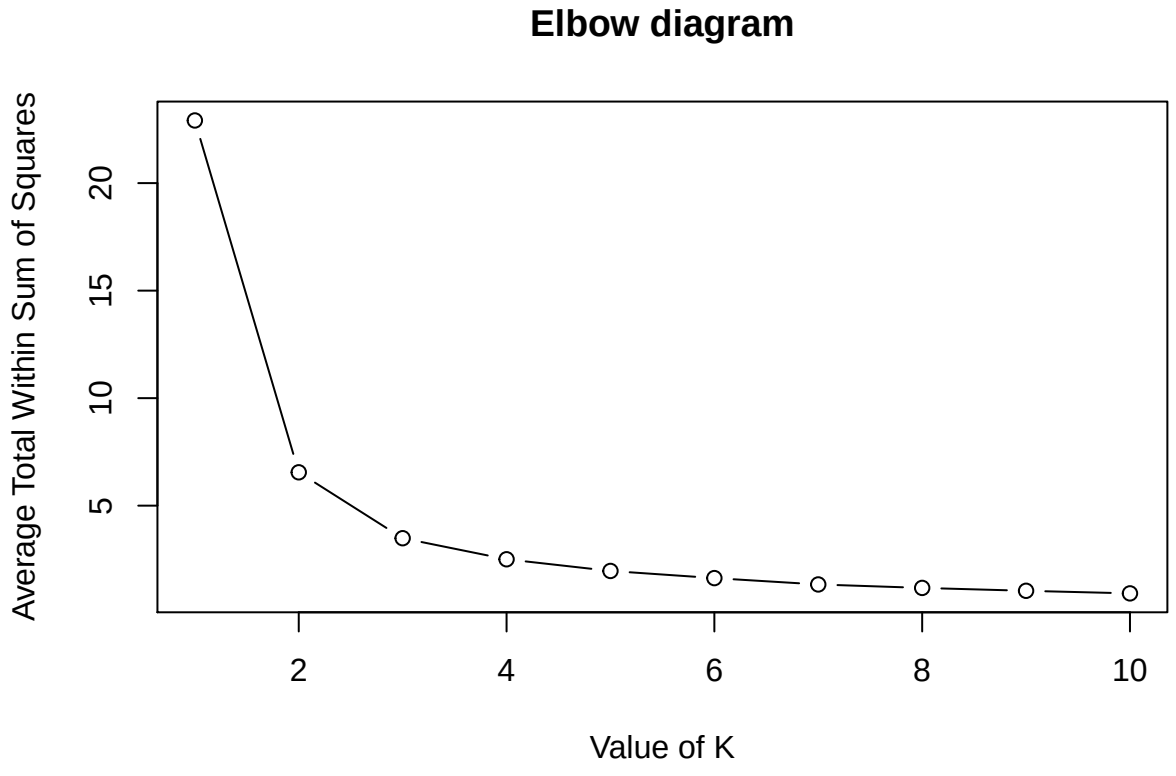

```
range<-1:10
n <-20 #Run the K Means algorithm n times
avg <-integer(length(range)) #Set up an empty vector to hold all of points
avg2 <- integer(length(range))
for(k in range){ # For each value of the range variable
 value <-integer(n) #Set up an empty vector to hold the n tries
 within_cluster_sum<-integer(n)
 for(i in 1:n){
   k.temp <-kmeans(data.scaled[c(1,4)],centers=k, nstart = 10) #Run kmeans
   # cluster sum of squares by cluster
   value[i] <-k.temp$betweenss/k.temp$totss*100 #store accuracy
   within_cluster_sum[i] <- k.temp$tot.withinss
 }
 avg[k] <-mean(value) #Average the n total accuracy
 avg2[k] <-mean(within_cluster_sum) #Average the n total accuracy
}
# see the values and the plot
avg
```

```
##  [1] 4.651598e-14 7.141356e+01 8.478659e+01 8.906005e+01 9.143170e+01
##  [6] 9.288022e+01 9.417169e+01 9.487311e+01 9.545840e+01 9.598239e+01
```

```
plot(range,avg,type="b", main="Accuracy of clustering",
 ylab="Accuracy",
 xlab="Value of K")
```




```r
# accuracy here is: measure of the total distance between points and their cluster centers
# we can see that when k=4, accuracy is around 90%
plot(range,avg2,type="b", main="Elbow diagram",
 ylab="Average Total Within Sum of Squares",
 xlab="Value of K")
```



```r
# compare other k's with Species
cluster3 <- kmeans(data.scaled[c(1,4)],3, nstart = 10)
cluster4 <- kmeans(data.scaled[c(1,4)],4, nstart = 10)
cluster5 <- kmeans(data.scaled[c(1,4)],5, nstart = 10)
# now use the actual clustering that we shouldn't know
# here we can compare how the kmeans clutered the species
table(cluster3$cluster, data$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          3        36
##   2     50          3         0
##   3      0         44        14
```

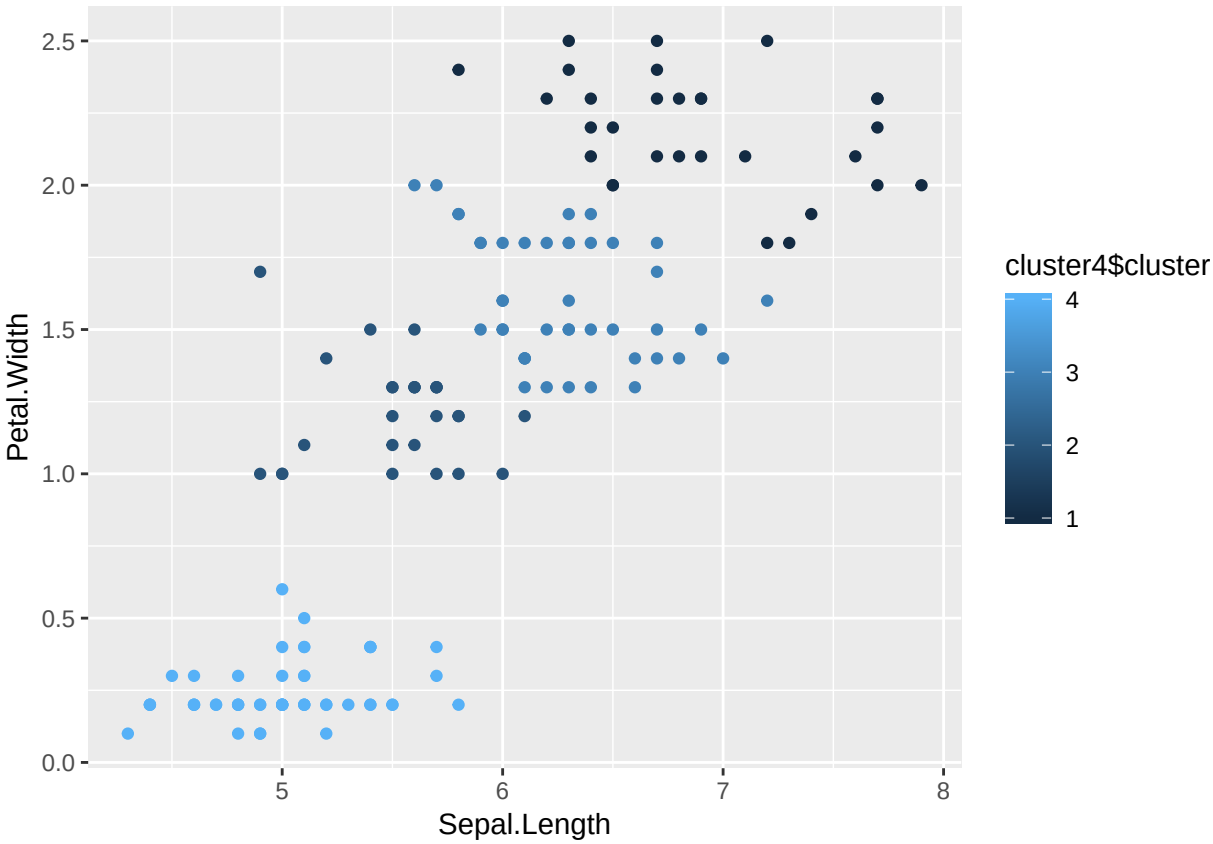
```r
table(cluster4$cluster, data$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          0        30
##   2      0         26         1
##   3      0         24        19
##   4     50          0         0
```

```r
table(cluster5$cluster, data$Species)
```

```
##
##     setosa versicolor virginica
##   1      0         26         1
##   2     50          0         0
##   3      0          0        20
##   4      0         24        18
##   5      0          0        11
```

```r
# plot cluster

ggplot(data, aes(Sepal.Length, Petal.Width, color = cluster4$cluster)) + geom_point()
```



Analysis of Clustering

In this problem we had 4 predictors to find the best cluster. Those predictors were Sepal.Length, Sepal.Width, Petal.Length and Petal.Width. To have better results I scaled the data (stdev=1).

To know which predictors to use, I followed the suggestion from the TA. Graph the predictors with `ggplot2` and see which one would generate some easy to divide group. So I tested and found out that using `Sepal.Length, Petal.Width` or using `Petal.Length, Petal.Width` could be good enough as predictors.

Then I tested multiple k's from 1 to 10. Then there were two types of measuring how good was the clusters (kmeans result).

1. Measure of the total distance between points and their cluster centers.
2. In this data set, we had the actual cluster each data point belongs to, but we won't have this data on a real clustering problem.

So I focused more on the first measuring type.

I found I way to measure the accuracy of the total distance between points by this equation = `The between-cluster sum of squares/The total sum of squares`.

In the first plot we can see that when k >= 3, the accuracy is `>94%`. Take notice that here there is no need to use the best accuracy we have, because we only need to know a good cluster enough for our needs. So I recommend to use k = 4 to cluster our iris data.

To confirm this I made the `Elbow diagram` using `Total within-cluster sum of squares` and I can confirm that when k >= 3, the cluster is good enough.

Finally I used the table to use the classification of Species and confirm the clustering. With k = 4, setosa is all clustered in one group, versicolor in 2 groups and virginica almost in 2 groups.