Homework 1 - Classification

Jeff Tilton

8/20/2018

Question 3.1

A: KNN Cross Validation

Data

```
library("kknn")
library(kableExtra)
raw = as.matrix(read.table('credit_card_data-headers.txt', header = TRUE, sep = '\t'))
range01 = function(x){(x-min(x))/(max(x)-min(x))}
data = apply(raw, 2, range01)
kable(head(data))
```

A1	A2	A3	A8	A9	A10	A11	A12	A14	A15	R1
1	0.2568421	0.0000000	0.0438596	1	0	0.0149254	1	0.1010	0.00000	1
0	0.6754887	0.1592857	0.1066667	1	0	0.0895522	1	0.0215	0.00560	1
0	0.1616541	0.0178571	0.0526316	1	1	0.0000000	1	0.1400	0.00824	1
1	0.2117293	0.0550000	0.1315789	1	0	0.0746269	0	0.0500	0.00003	1
1	0.0965414	0.2008929	0.0600000	1	1	0.0000000	1	0.0600	0.00000	1
1	0.2756391	0.1428571	0.0877193	1	1	0.0000000	0	0.1800	0.00000	1

Cross Validation

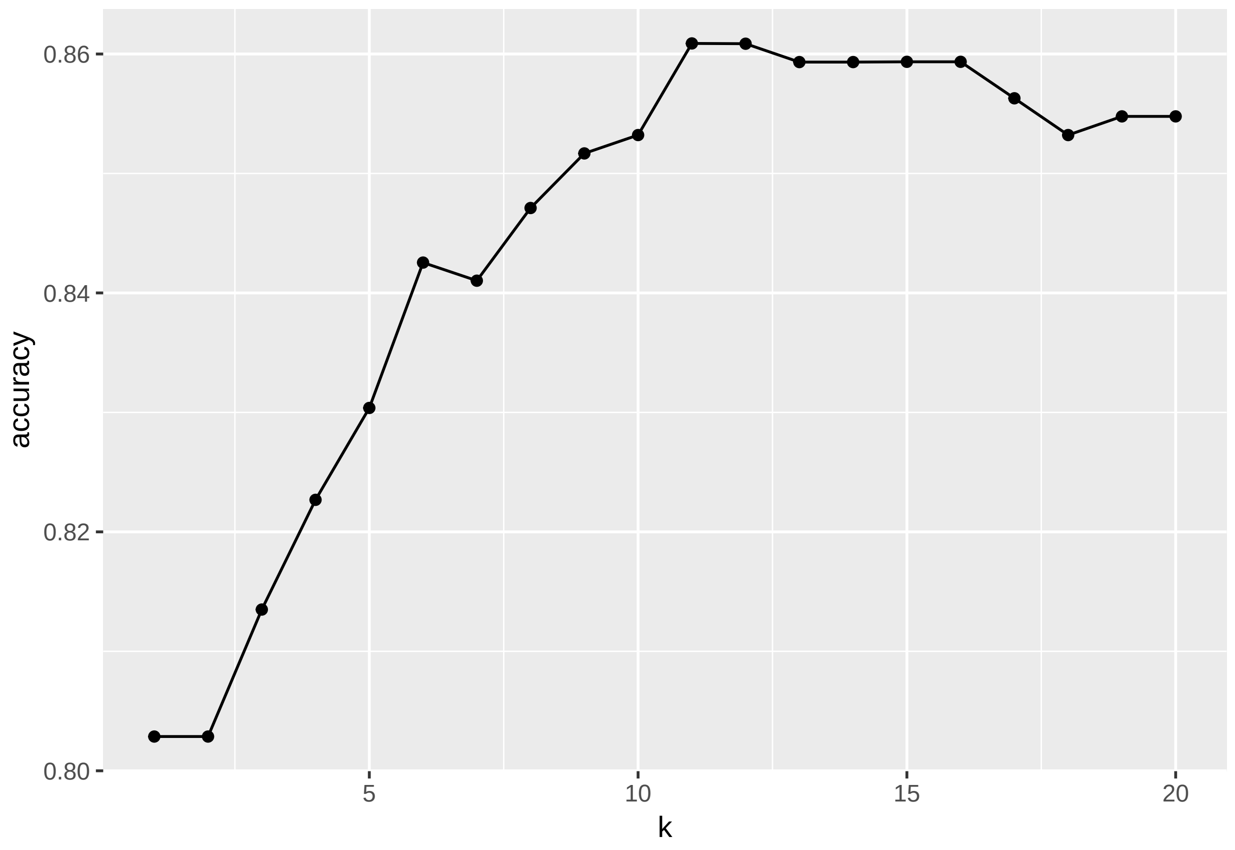
```
library("data.table")
library("magrittr")

##Reference: Jake Drew (https://stats.stackexchange.com/users/49529/jake-drew),
##How to split a data set to do 10-fold cross validation,
##URL (version: 2014-07-04): https://stats.stackexchange.com/q/105839


#Randomly shuffle the data

data = data[sample(nrow(data)),]

#Create 10 equally size folds
folds = cut(seq(1,nrow(data)),breaks=10,labels=FALSE)


#empty data table to store accuracy values
final_accuracy = data.table(k_1 = numeric(), k_2 = numeric(), k_3 = numeric(),
                            k_4 = numeric(), k_5 = numeric(), k_6 = numeric(),
                            k_7 = numeric(), k_8 = numeric(), k_9 = numeric(),
                            k_10 = numeric(), k_11 = numeric(), k_12 = numeric(),
                             k_13 = numeric(), k_14 = numeric(), k_15 = numeric(),
                            k_16 = numeric(), k_17 = numeric(), k_18 = numeric(),
                            k_19 = numeric(), k_20 = numeric())
```

```r
#Perform 10 fold cross validation
for(i in 1:10){
    #Segement your data by fold using the which() function
    testIndexes = which(folds==i,arr.ind=TRUE)
    test = data[testIndexes, ]
    train = data[-testIndexes, ]

    acc = c()
    for(k in 1:20){
      model = kknn(R1~.,
                 as.data.frame(train),
                 as.data.frame(test),
                 k = k, distance = 1,
                 kernel = "triangular", scale = TRUE)

      y_hat = fitted(model)
      accuracy = sum(round(y_hat) == test[,11]) / nrow(test)
      acc[k] = accuracy
    }
    final_accuracy = rbind(final_accuracy, as.list(acc))

}

#I want to get the column name with the best accuracy
accuracy = colMeans(final_accuracy)
max_accuracy =  colMeans(final_accuracy)  %>% max
k_value = accuracy[accuracy == max_accuracy] %>%
          as.data.frame %>%
          rownames %>%
          .[1] %>%
          strsplit(., '_') %>%
          unlist %>%
          .[2]
kable(accuracy)
```

	x
k_1	0.8028671
k_2	0.8028671
k_3	0.8134965
k_4	0.8226807
k_5	0.8303730
k_6	0.8425408
k_7	0.8410256
k_8	0.8471096
k_9	0.8516783
k_10	0.8532168
k_11	0.8608858
k_12	0.8608625
k_13	0.8593240
k_14	0.8593240
k_15	0.8593473
k_16	0.8593473
k_17	0.8562937
k_18	0.8532168
k_19	0.8547786
k_20	0.8547786

Plot

```r
library("ggplot2")
k = seq(1, length(accuracy), by=1)
y = as.list(melt(accuracy))
df = data.frame( k=k, y=y)
df['accuracy'] = df['value']
gg = ggplot(data = df, aes(x = k, y = accuracy)) +
  geom_line() + geom_point()

plot(gg)
```



Discussion

The 10 fields cross validation yielded an accuracy of 0.86 with 11 neighbors.

B: KNN Split Data into Training, Validation, and Test data sets

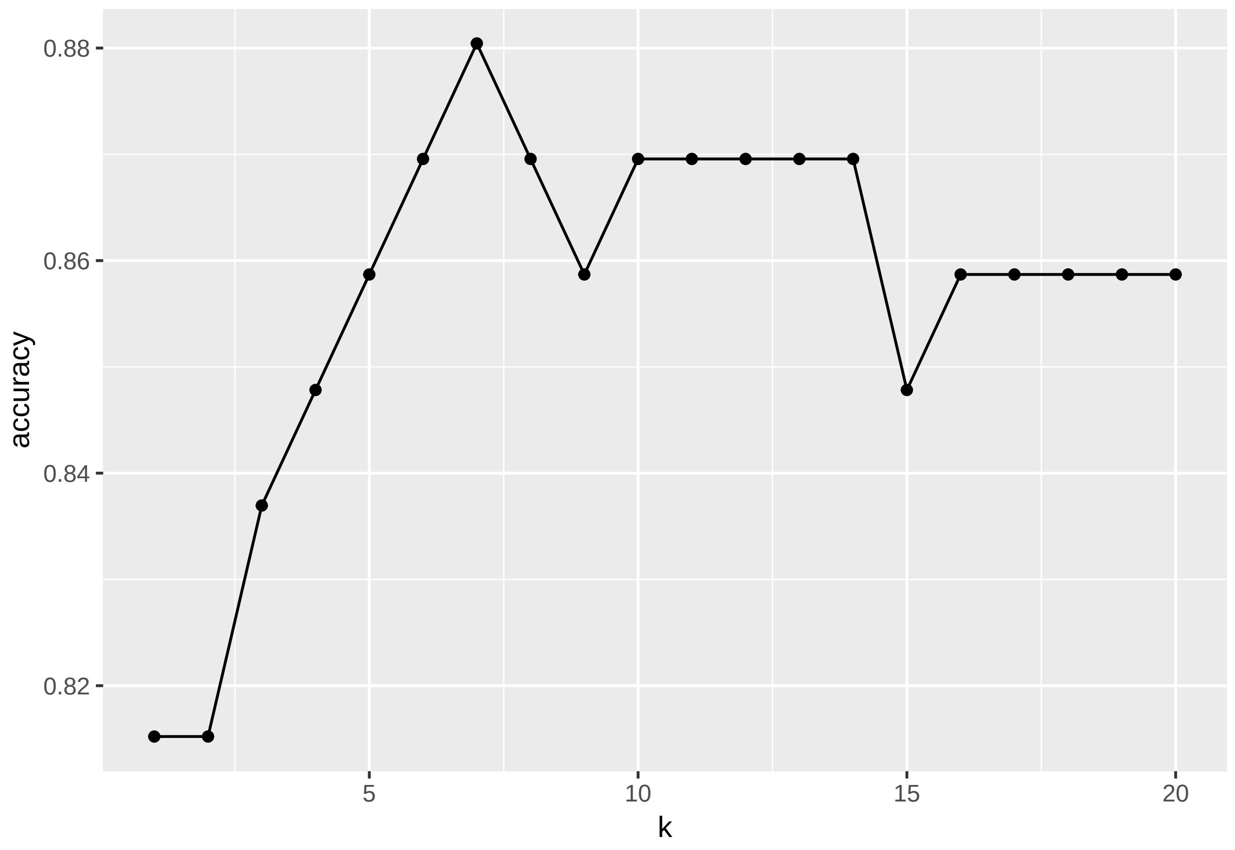
```r
assignment = sample(1:3, size = nrow(data), prob = c(0.7, 0.15, 0.15), replace = TRUE)

# Create a train, validation and tests from the original data frame
train = data[assignment == 1, ]  # subset the grade data frame to training indices only
validate = data[assignment == 2, ]  # subset the grade data frame to validation indices only
test = data[assignment == 3, ]   # subset the grade data frame to test indices only


#empty data table to store accuracy values
final_accuracy = data.table(k_1 = numeric(), k_2 = numeric(), k_3 = numeric(),
                            k_4 = numeric(), k_5 = numeric(), k_6 = numeric(),
                            k_7 = numeric(), k_8 = numeric(), k_9 = numeric(),
                            k_10 = numeric(), k_11 = numeric(), k_12 = numeric(),
                            k_13 = numeric(), k_14 = numeric(), k_15 = numeric(),
                            k_16 = numeric(), k_17 = numeric(), k_18 = numeric(),
                            k_19 = numeric(), k_20 = numeric())


acc = c()
```

```r
for(k in 1:20){
  model = kknn(R1~.,
               as.data.frame(train),
               as.data.frame(validate),
               k = k, distance = 1,
               kernel = "triangular", scale = TRUE)

  y_hat = fitted(model)
  accuracy = sum(round(y_hat) == validate[,11]) / nrow(validate)
  acc[k] = accuracy
}

#Get the best K value
final_accuracy = rbind(final_accuracy, as.list(acc))
accuracy = colMeans(final_accuracy)
max_accuracy = max(accuracy)
k_value = accuracy[accuracy == max_accuracy] %>%
          as.data.frame %>%
          rownames %>%
          .[1] %>%
          strsplit(., '_') %>%
          unlist %>%
          .[2]

#Use test set
model = kknn(R1~.,
               as.data.frame(train),
               as.data.frame(test),
               k = as.integer(k_value), distance = 1,
               kernel = "triangular", scale = TRUE)

y_hat = fitted(model)
test_accuracy = sum(round(y_hat) == test[,11]) / nrow(test) %>%
              round(., 2)
test_accuracy
```

```
## [1] 0.8348624
```

Plot

```r
library("ggplot2")
k = seq(1, length(accuracy), by=1)
y = as.list(melt(accuracy))
df = data.frame( k=k, y=y)
df['accuracy'] = df['value']
gg = ggplot(data = df, aes(x = k, y = accuracy)) +
  geom_line() + geom_point()

plot(gg)
```



Discussion

The splitting the data into train, validation, and test data sets yielded a test accuracy of 0.83 with 7 neighbors. The K-folds cross validation is a more robust method for model selection, but is more difficult to implement. Comparing the plots between the two methods, the k-folds plot displays an obvious pattern. The accuracy increases with the number of neighbors up to a point and then begins to lose accuracy. The split data plot displays a similar pattern, but is a little more random and if the seed is not set can vary wildly. The cross validation plot is more stable and displays a similar pattern no matter how the data is split, although the number of neighbors varies as well. Therefore, I choose the model determined by cross validation to be the best model for this assignment.

Question 4.1

There are many instances when cluster analysis can be useful. For example a web commerce site may want to target specific customers using a recommender system. Clustering the customers into groups before applying a supervised algorithm to the identified clusters can improve the accuracy of the supervised model. Some examples of predictors to use are:

1. Time spent on site
2. Type of products purchased
3. Average cost of item
4. Average cost of “cart”

Question 4.2: Use the R function kmeans to cluster the points as well as possible

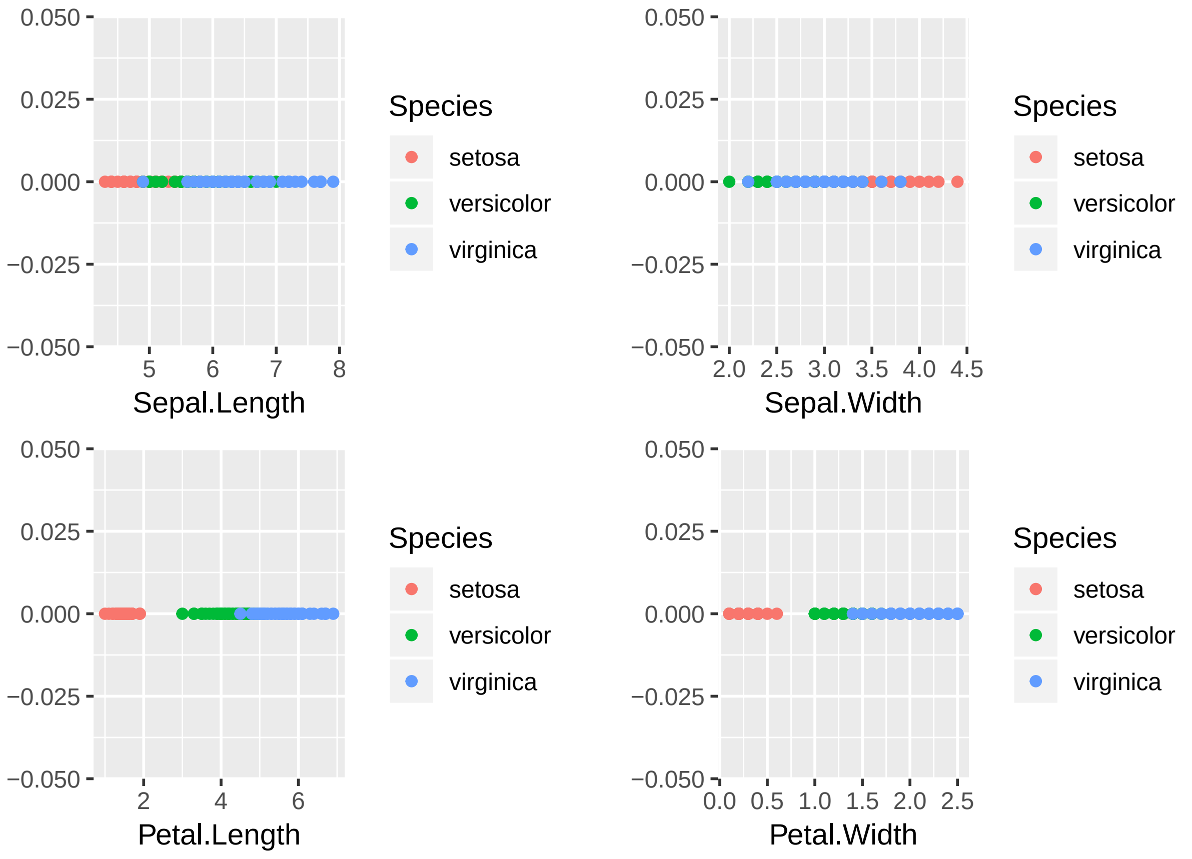
From my understanding, cluster analysis is often used as a non-supervised learning algorithm. Therefore, knowing the species ahead of time might not be realistic. However, I am plotting the data by each predictor, colored by species, because we were asked to cluster the points as well as possible. This will allow me to choose the best predictors.

```r
data(iris)
kable(head(iris))
```

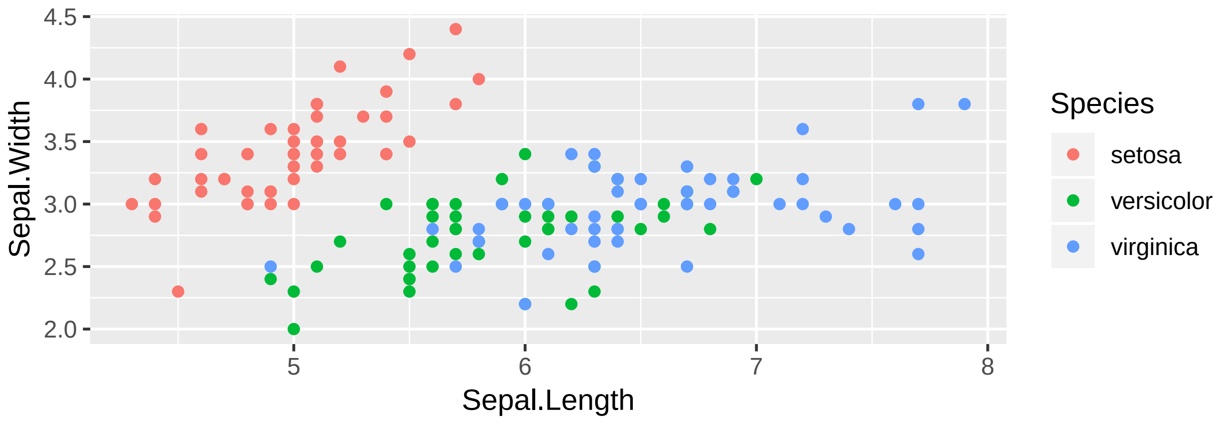
Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

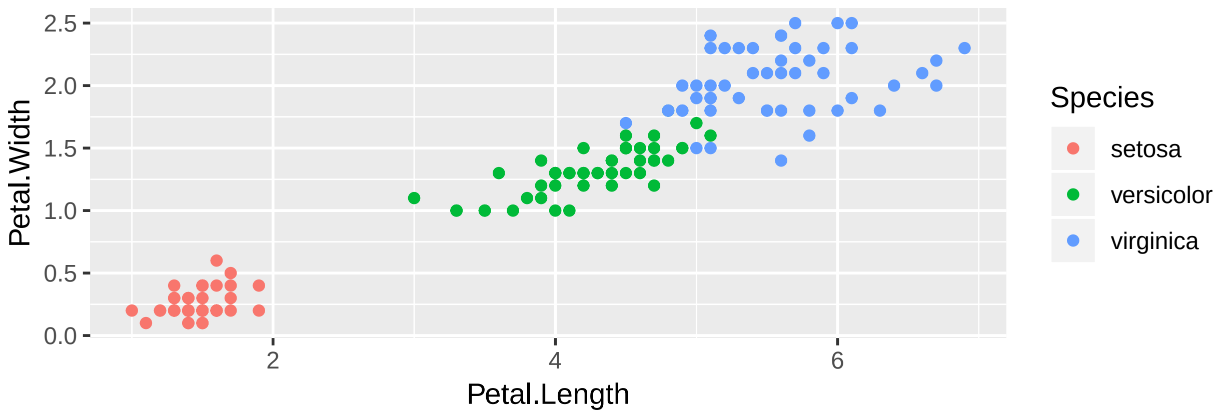
Plot data for visualization

```r
library(gridExtra)
sl = ggplot(data=iris, aes(x = Sepal.Length, y = c(0))) +
      geom_point(aes(color=Species)) +  ylab("")


sw = ggplot(data=iris, aes(x = Sepal.Width, y = c(0))) +
      geom_point(aes(color=Species)) +  ylab("")


pl = ggplot(data=iris, aes(x = Petal.Length, y = c(0)))  +
      geom_point(aes(color=Species)) +  ylab("")


pw = ggplot(data=iris, aes(x = Petal.Width, y = c(0))) +
       geom_point(aes(color=Species)) +  ylab("")


grid.arrange(sl, sw, pl, pw, nrow = 2)
```



```
sepal = ggplot(data=iris, aes(x = Sepal.Length, y = Sepal.Width)) +
        geom_point(aes(color=Species))

petal = ggplot(data=iris, aes(x = Petal.Length, y = Petal.Width)) +
        geom_point(aes(color=Species))

grid.arrange(sepal,petal, nrow = 2)
```





The above plots show that the species cluster the tightest around petal length and width. Therefore I will fit the data to each predictor individually, as well as the two predictors together and choose the best model.

Models

```
data(iris)
iris_scaled = apply(iris[,1:4], 2, range01)

pl = kmeans(iris_scaled[,3],centers = 3, nstart = 20)
pw = kmeans(iris_scaled[,4],centers = 3, nstart = 20)
p = kmeans(iris_scaled[,3:4],centers = 3, nstart = 20)

iris$pl_group = pl$cluster
iris$pw_group = pw$cluster
iris$p_group = p$cluster
```

I knew ahead of time that there should be three centers because there are three species. Once again, it might not always be possible to know ahead of time how many centers are necessary.

Petal Length

```
kable(table(pl$cluster, iris$Species))
```

setosa	versicolor	virginica
0	48	6
50	0	0
0	2	44

Petal Width

```r
kable(table(pw$cluster, iris$Species))
```

setosa	versicolor	virginica
50	0	0
0	48	4
0	2	46

Petal

```r
kable(table(p$cluster, iris$Species))
```

setosa	versicolor	virginica
50	0	0
0	2	46
0	48	4

Discussion

The models with the joint predictors petal length and petal width, as well as the model with the single petal width predictor clustered the data most accurately as seen above. Each model classified the setosa species correctly and missclassified upt to 2 versicolor and 6 virginica. The best model, for the data provided, is the one that uses the single predictor petal width because it is the simplest model with the best results.

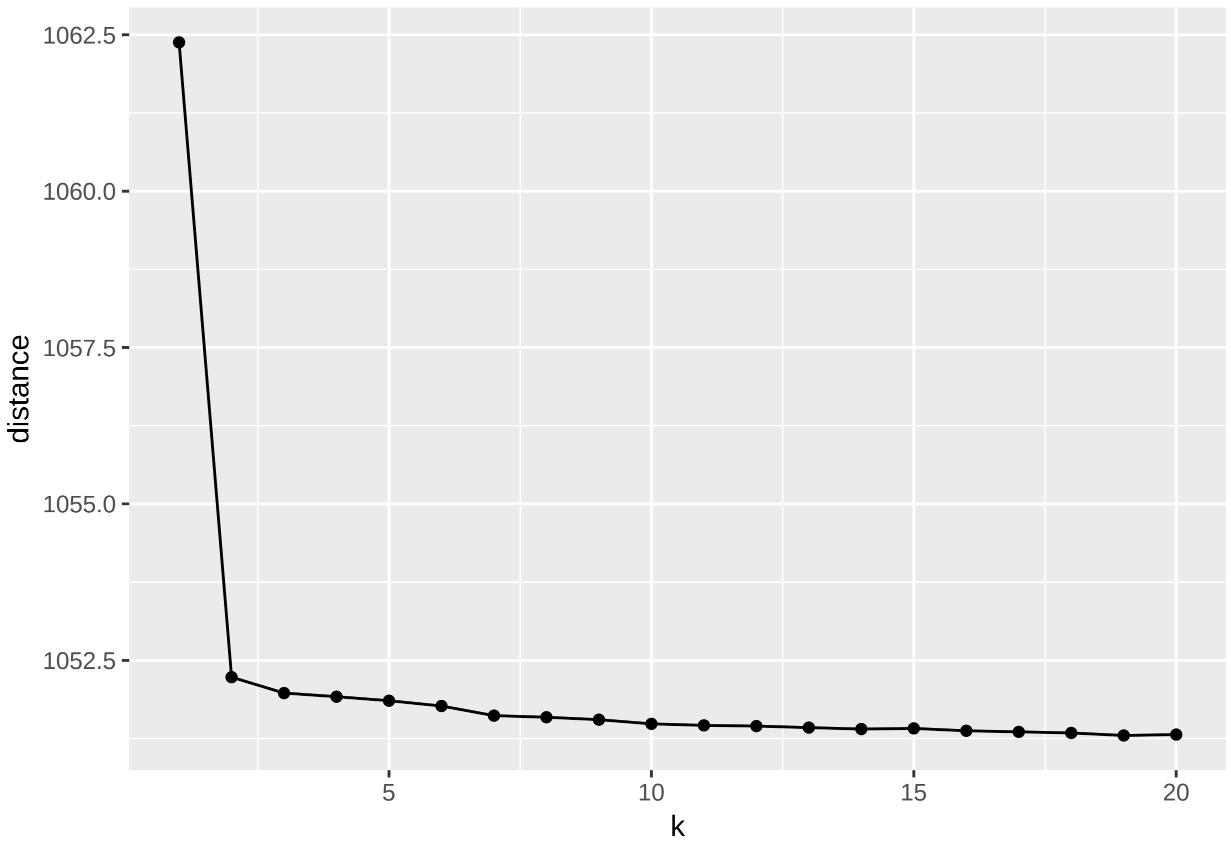
Cluster analysis as unsupervised learning

Unlike the above example, when the number of species are known, cluster analysis works well as an unsupervised learning algorithm to classify data into separate groups. One method, as discussed in lecture, is to plot the data into an Elbow-curve. The Elbow-curve plots the sum total distance of each point to it's cluster center as a function of k clusters. The point at which the rate of change of the sum total distance drops to form the elbow shape is the chosen number of cluster centers.

The sum total distance of points to cluster center is plotted below from 1 to 20 cluster centers.

```r
#This is extremely ugly, but I am trying to get the
#distance for each k value so I can plot it as seen in the lecture videos
distance_tbl = data.table(k = seq(1,20, by =1))
distance = function(x){
  k = x[1]
  ir = iris
  fit = kmeans(iris_scaled,centers = k, nstart = 20)
  ir$group = fit$cluster

  centers = data.table(fit$centers)

  dis = function(x){
      grp = as.integer(x['group'])
      center = centers[grp,]
      sl = as.numeric(x[1])
      sw = as.numeric(x[2])
      pl = as.numeric(x[3])
      pw = as.numeric(x[4])

      c_sl = center$Sepal.Length
      c_sw = center$Sepal.Width
      c_pl = center$Petal.Length
      c_pw = center$Petal.Width

      d = sqrt(
                  (sl-c_sl)^2 +
                  (sw - c_sw)^2 +
                  (pl - c_pl)^2 +
                  (pw - c_pw)^2
              )
      return(d)
      }
  d = sum(apply(ir, 1, dis))
  return(d)
}


distance_tbl$distance = apply(distance_tbl, 1, distance)
```

```
gg = ggplot(data = distance_tbl, aes(x = k, y = distance)) +
  geom_line() + geom_point()

plot(gg)
```



Discussion

The plot above shows that 2 cluster centers would be a reasonable estimate for the iris dataest. This makes sense when looking at the previous plots. Setosa has a significantly smaller petal length and width than the other two species. Versicolor and virginica overlap each other considerably especially in the sepal lenght and width.