DSC 520 Week 11

Miles Peña

2024-02-27

Week 11 Exercise

1. Introduction to Machine Learning

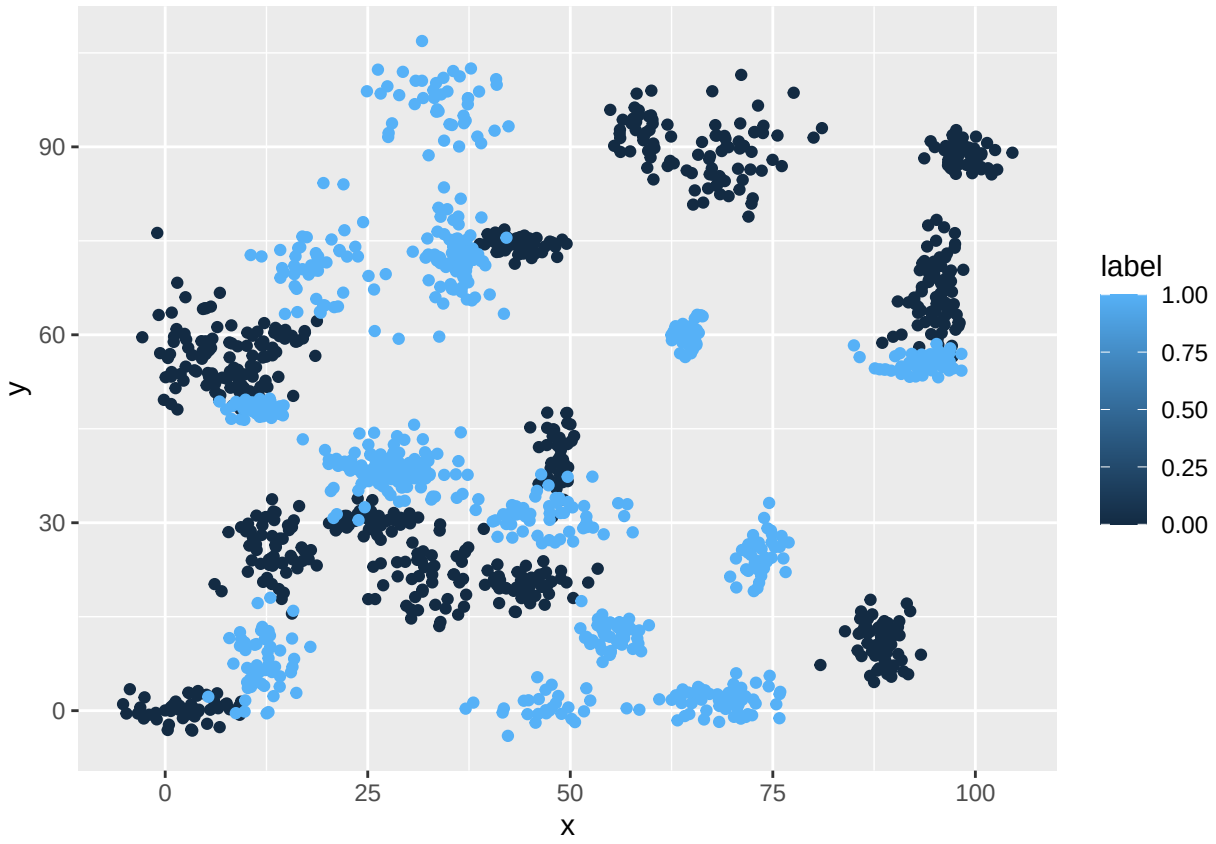
```
library(ggplot2)
setwd("/Users/milespena/Documents/R")
```

```
# Load CSV files into new variables

binaryData <- read.csv("binary-classifier-data.csv")
trinaryData <- read.csv("trinary-classifier-data.csv")
```

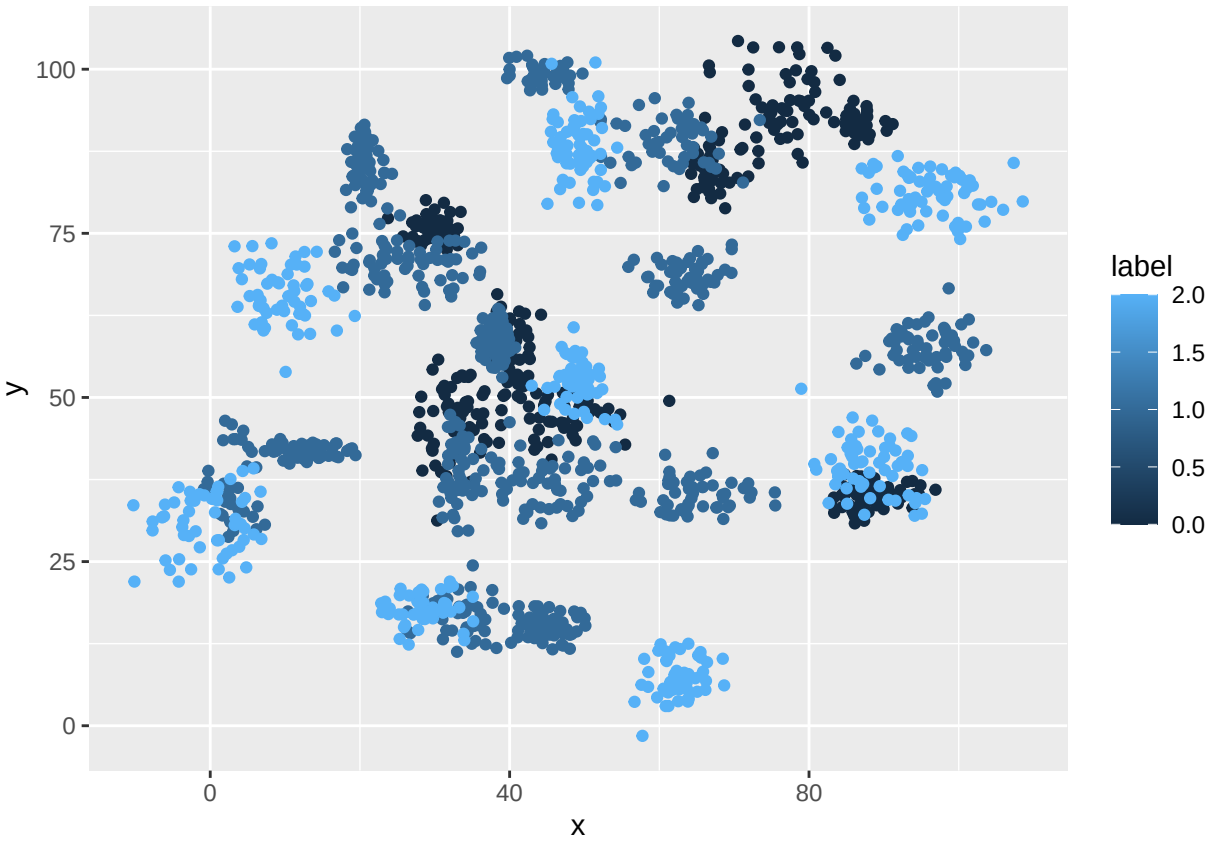
Plot the data from each dataset using a scatter plot.

```
# Create plot variables using ggplot

binaryPlot <- ggplot(binaryData, aes(x = x, y = y, color = label)) +
  geom_point()

trinaryPlot <- ggplot(trinaryData, aes(x = x, y = y, color = label)) +
  geom_point()
```

```
# Print out binary plot

binaryPlot
```




```
# Print trinary plot

trinaryPlot
```



The k nearest neighbors algorithm categorizes an input value by looking at the labels for the k nearest points and assigning a category based on the most common label. In this problem, you will determine which points are nearest by calculating the Euclidean distance between two points. As a refresher, the Euclidean distance between two points:

For this specific part, last week we split datasets into testing and training in order to perform analysis on it. I am not 100% sure if this is the correct way to do this portion of the assignment but it is the only way that I can think of doing it even after much research on the topic.

```
# Create data normalization

binary <- binaryData[, c("x", "y")]
trinary <- trinaryData[, c("x", "y")]
```

```
# Create train and test for binary

set.seed(200)
binarySelection <- sample(1:nrow(binary), size = nrow(binary) * 0.60,
                          replace = FALSE)
binaryTrain <- binaryData[binarySelection, ]
NROW(binaryTrain)
```

```
## [1] 898
```

```
binaryTest <- binaryData[-binarySelection, ]
NROW(binaryTest)
```

```
## [1] 600
```

```
trainLabelBinary <- binaryData[binarySelection, 1, drop = TRUE]
NROW(trainLabelBinary)
```

```
## [1] 898
```

```
testLabelBinary <- binaryData[-binarySelection, 1, drop = TRUE]
NROW(testLabelBinary)
```

```
## [1] 600
```

```
# Create train and test for trinary

set.seed(200)
trinarySelection <- sample(1:nrow(trinary), size = nrow(trinary)*0.60,
                           replace = FALSE)
trinaryTrain <- trinaryData[trinarySelection, ]
NROW(trinaryTrain)
```

```
## [1] 940
```

```
trinaryTest <- trinaryData[-trinarySelection, ]
NROW(trinaryTest)
```

```
## [1] 628
```

```
trainLabelTrinary <- trinaryData[trinarySelection, 1, drop = TRUE]
NROW(trainLabelTrinary)
```

```
## [1] 940
```

```
testLabelTrinary <- trinaryData[-trinarySelection, 1, drop = TRUE]
NROW(testLabelTrinary)
```

```
## [1] 628
```

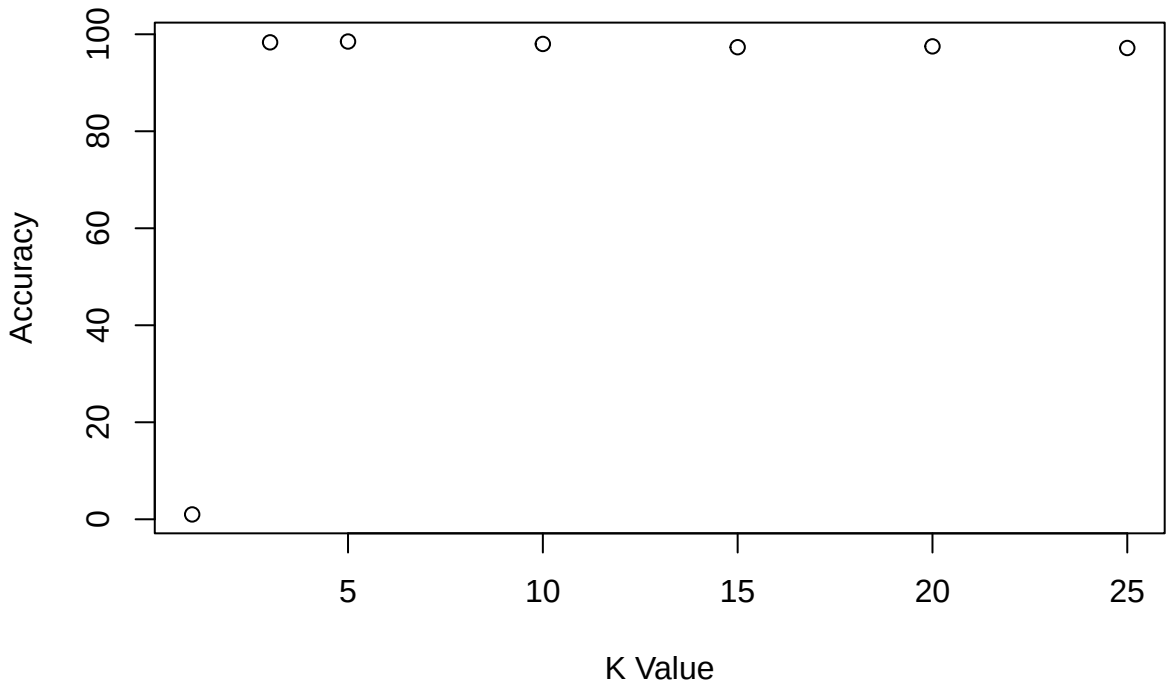
Fitting a model is when you use the input data to create a predictive model. There are various metrics you can use to determine how well your model fits the data. For this problem, you will focus on a single metric, accuracy. Accuracy is simply the percentage of how often the model predicts the correct result. If the model always predicts the correct result, it is 100% accurate. If the model always predicts the incorrect result, it is 0% accurate.

```
library(class)

nearest <- list(3, 5, 10, 15, 20, 25)
input = 1
binaryAccuracy = 1
for (input in nearest) {
  nearestNeighbor <- knn(train = binaryTrain, test = binaryTest,
                         cl = trainLabelBinary, k = input)
  binaryAccuracy[input] <- 100 * sum(testLabelBinary == nearestNeighbor) /
    NROW(testLabelBinary)
  k = input
  cat(k,'=', binaryAccuracy[input],'')
}
```

Fit a k nearest neighbors' model for each dataset for k=3, k=5, k=10, k=15, k=20, and k=25. Compute the accuracy of the resulting models for each value of k. Plot the results in a graph where the x-axis is the different values of k and the y-axis is the accuracy of the model.

```
## 3 = 98.33333 5 = 98.5 10 = 98 15 = 97.33333 20 = 97.5 25 = 97.16667
```

```
#Plot binary accuracy

plot(binaryAccuracy, type = "b", xlab = "K Value", ylab = "Accuracy")
```

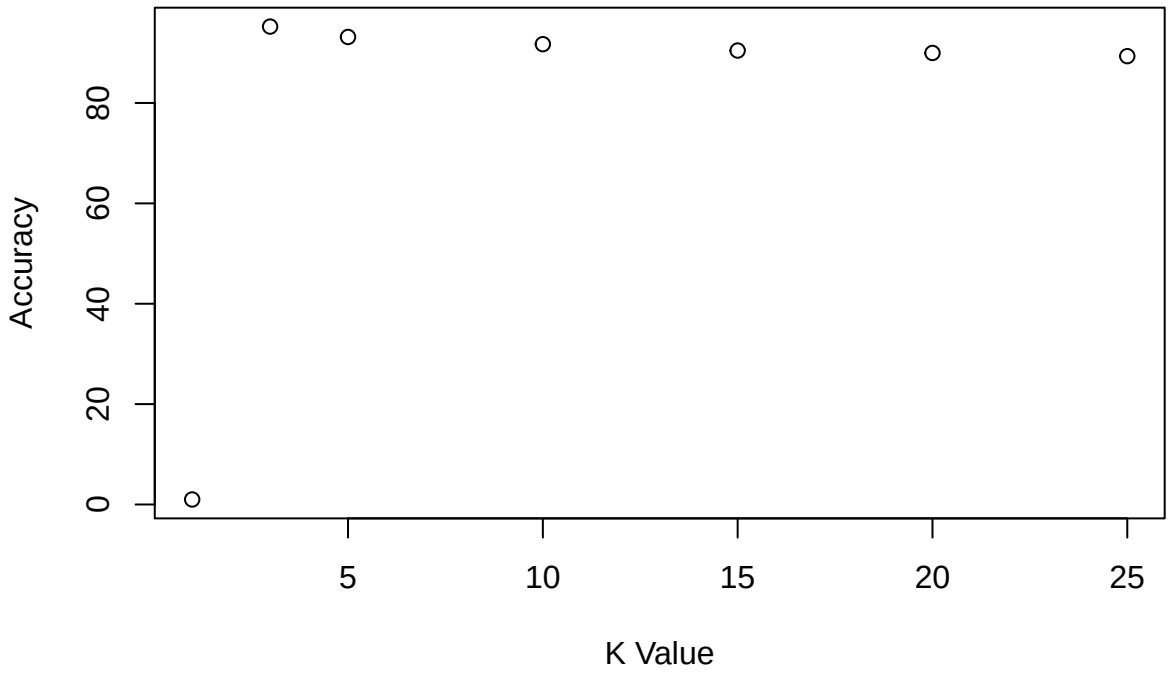



```
nearestTrinary <- list(3, 5, 10, 15, 20, 25)
inputTrinary = 1
trinaryAccuracy = 1
for (inputTrinary in nearestTrinary) {
  trinaryNearest <- knn(train = trinaryTrain, test = trinaryTest,
```

```
                        cl = trainLabelTrinary, k = inputTrinary)
  trinaryAccuracy[inputTrinary] <- 100 *
    sum(testLabelTrinary == trinaryNearest) / NROW(testLabelTrinary)
  k = inputTrinary
  cat(k,'=',trinaryAccuracy[inputTrinary], '')
}
```

```
## 3 = 95.22293 5 = 93.15287 10 = 91.71975 15 = 90.44586 20 = 89.96815 25 = 89.33121
```

```
# Plot trinary acccuracy

plot(trinaryAccuracy, type = "b", xlab = "K Value", ylab = "Accuracy")
```



Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?

From how I understand a linear classifier, I do not believe that this would work well on these datasets. Based on the original scatterplots from the first part of this assignment, there is no way to split the points by label with a line across in any way as there are small clusters of mixed labels all throughout the plot.

How does the accuracy of your logistic regression classifier from last week compare? Why is the accuracy different between these two methods?

I had the accuracy for last week's logistic regression classifier at 58% whereas for this week all the k-values returned between 89% and 98% telling me that these methods are more accurate. I believe that the accuracy is different between the two methods due to the use of KNN values which were not used in last week's assignment. We can tell how the accuracy decreases as the k-value increases so we would have a lower accuracy when we used a larger value last week.

2. Clustering

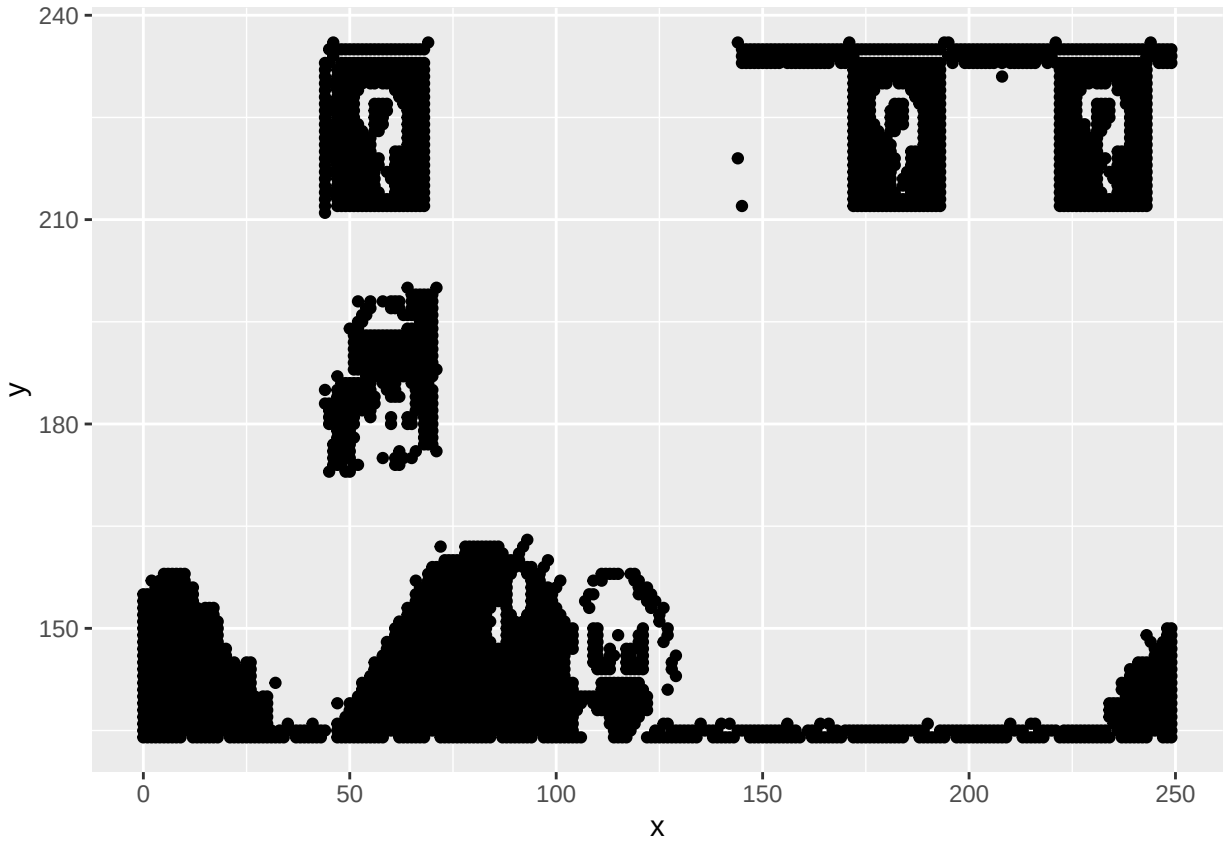
In this problem, you will use the k-means clustering algorithm to look for patterns in an unlabeled dataset. The dataset for this problem is found at data/clustering-data.csv.

Plot the dataset using a scatter plot.

```
# Load CSV file

clusteringData <- read.csv("clustering-data.csv")
```

```
# Plot data with ggplot

clusteringPlot <- ggplot(clusteringData, aes(x = x, y= y)) + geom_point()
```

```
# Print clustering plot

clusteringPlot
```

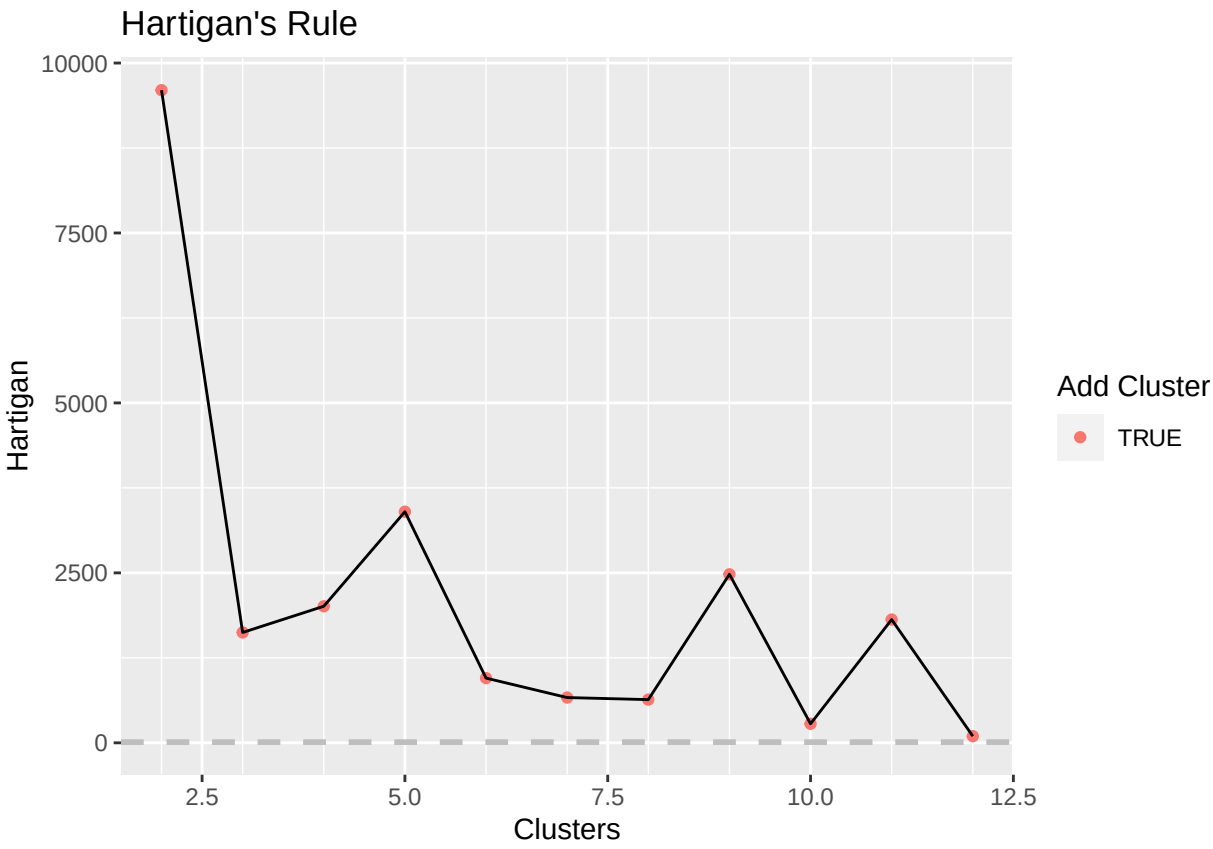


Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.

```
library(useful)
clusterBest <- FitKMeans(clusteringData, max.clusters = 12, nstart = 2,
                         seed = 1234)
clusterBest
```

```
##    Clusters   Hartigan AddCluster
## 1         2 9600.61378       TRUE
## 2         3 1623.33823       TRUE
## 3         4 2008.86000       TRUE
## 4         5 3400.01180       TRUE
## 5         6  952.09929       TRUE
## 6         7  665.84661       TRUE
## 7         8  635.58247       TRUE
## 8         9 2479.09613       TRUE
## 9        10  279.81141       TRUE
## 10       11 1813.35810       TRUE
## 11       12   97.74938       TRUE
```

```
PlotHartigan(clusterBest)
```

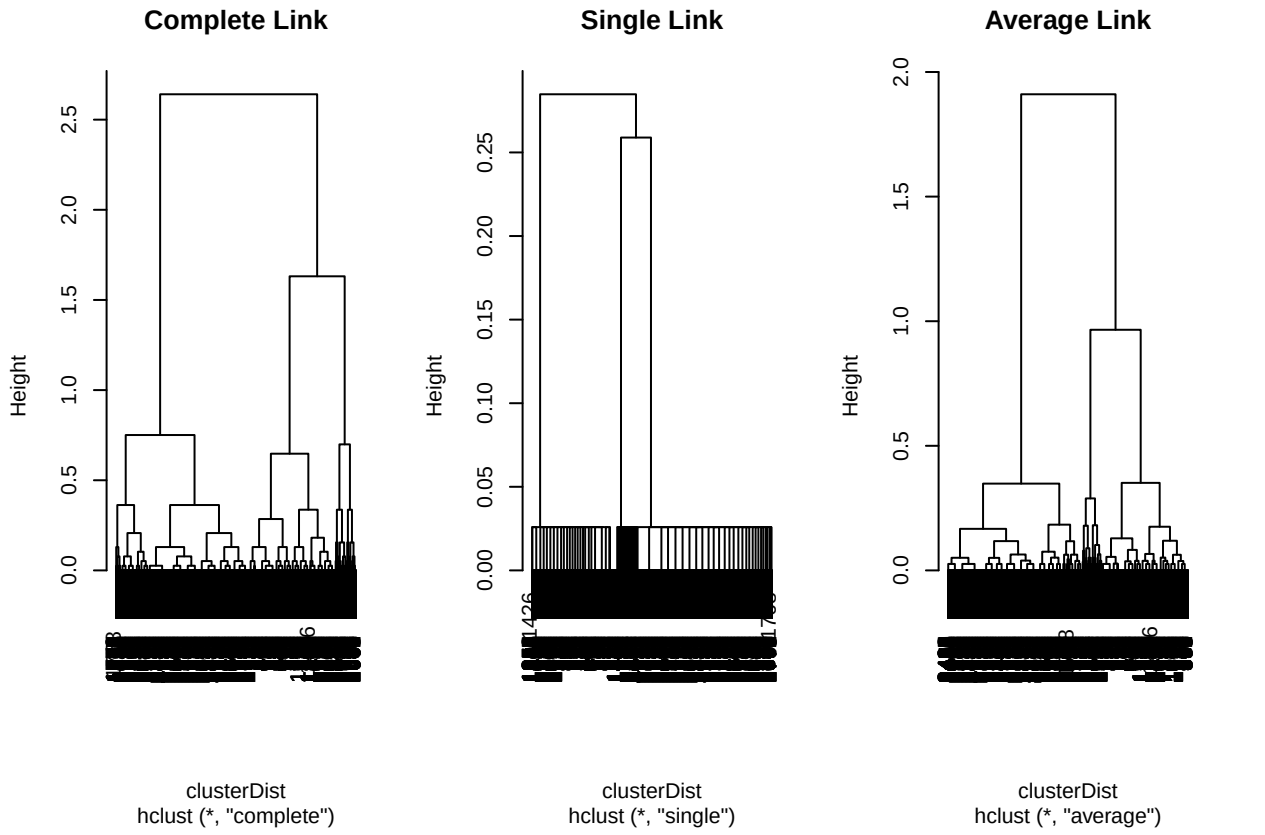


As k-means is an unsupervised algorithm, you cannot compute the accuracy as there are no correct values to compare the output to. Instead, you will use the average distance from the center of each cluster as a measure of how well the model fits the data. To calculate this metric, simply compute the distance of each data point to the center of the cluster it is assigned to and take the average value of all of those distances.

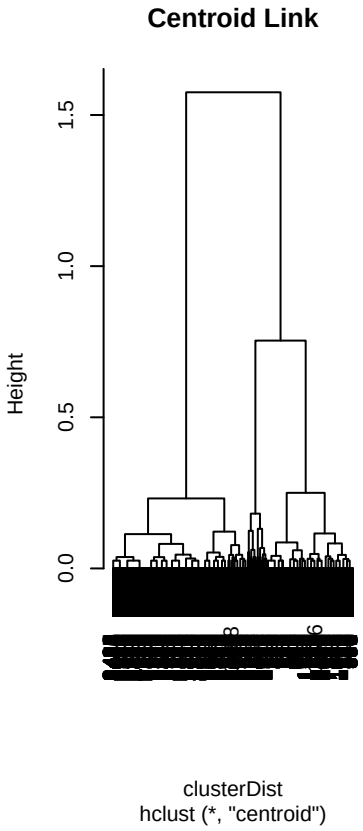
```
standardCluster <- scale(clusteringData[-1])
k_meansCluster <- kmeans(standardCluster, 4)
attributes(k_meansCluster)
```

```
## $names
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
##
## $class
## [1] "kmeans"
```

```
# Calculate the distance between

clusterDist <- dist(standardCluster)

# create hclust

clusterComplete <- hclust(clusterDist, method = "complete")
clusterSingle <- hclust(clusterDist, method = "single")
clusterAverage <- hclust(clusterDist, method = "average")
clusterCentroid <- hclust(clusterDist, method = "centroid")

# plot data

par(mfrow= c(1,3))
plot(clusterComplete, main = "Complete Link")
plot(clusterSingle, main = "Single Link")
plot(clusterAverage, main = "Average Link")
```

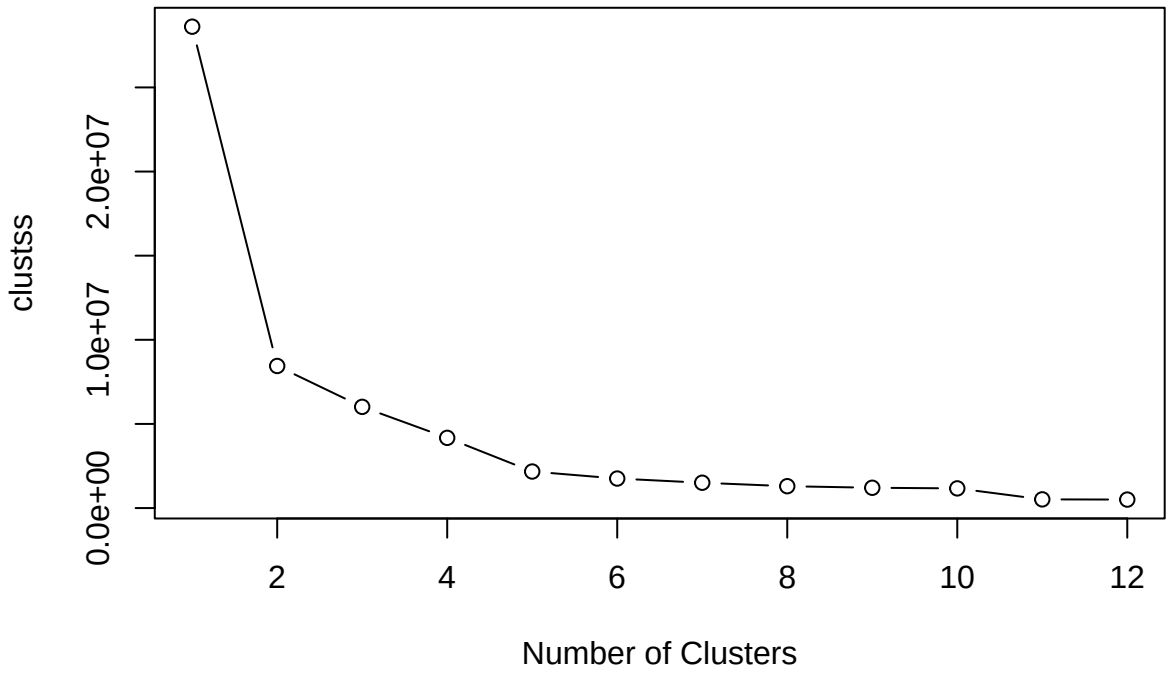



```r
plot(clusterCentroid, main = "Centroid Link")
```



Calculate this average distance from the center of each cluster for each value of k and plot it as a line chart where k is the x-axis and the average distance is the y-axis.

```
nearestCluster <- list(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
clusterInput = 1
seed = 1234
nc = 12
clusterPlot <- function(clusteringData, nc = 12, seed) +
  clustss <- (nrow(clusteringData) -1) * sum(apply(clusteringData, 2, var))
clustss <- (nrow(clusteringData)-1) * sum(apply(clusteringData, 2, var))
for (clustInput in 2:nc) {
   set.seed(seed)
   clustss[clustInput] <-sum(kmeans(clusteringData,
                                    centers = clustInput)$withinss)}

# Plot my cluster using the plot function
plot(1:nc, clustss, type = "b", xlab = "Number of Clusters")
```



One way of determining the "right" number of clusters is to look at the graph of k versus average distance and finding the "elbow point". Looking at the graph you generated in the previous example, what is the elbow point for this dataset?

The optimal number of clusters, or the correct value of k, is the point at which the value begins to decrease slowly which is also known as the "elbow point". By looking at the above graph, the "elbow point" appears to be at either k = 4 or k = 5.