Assignment 11.2 Machine Learning

David Pahmer

2022-06-03

Machine Language

```
library(useful)
```

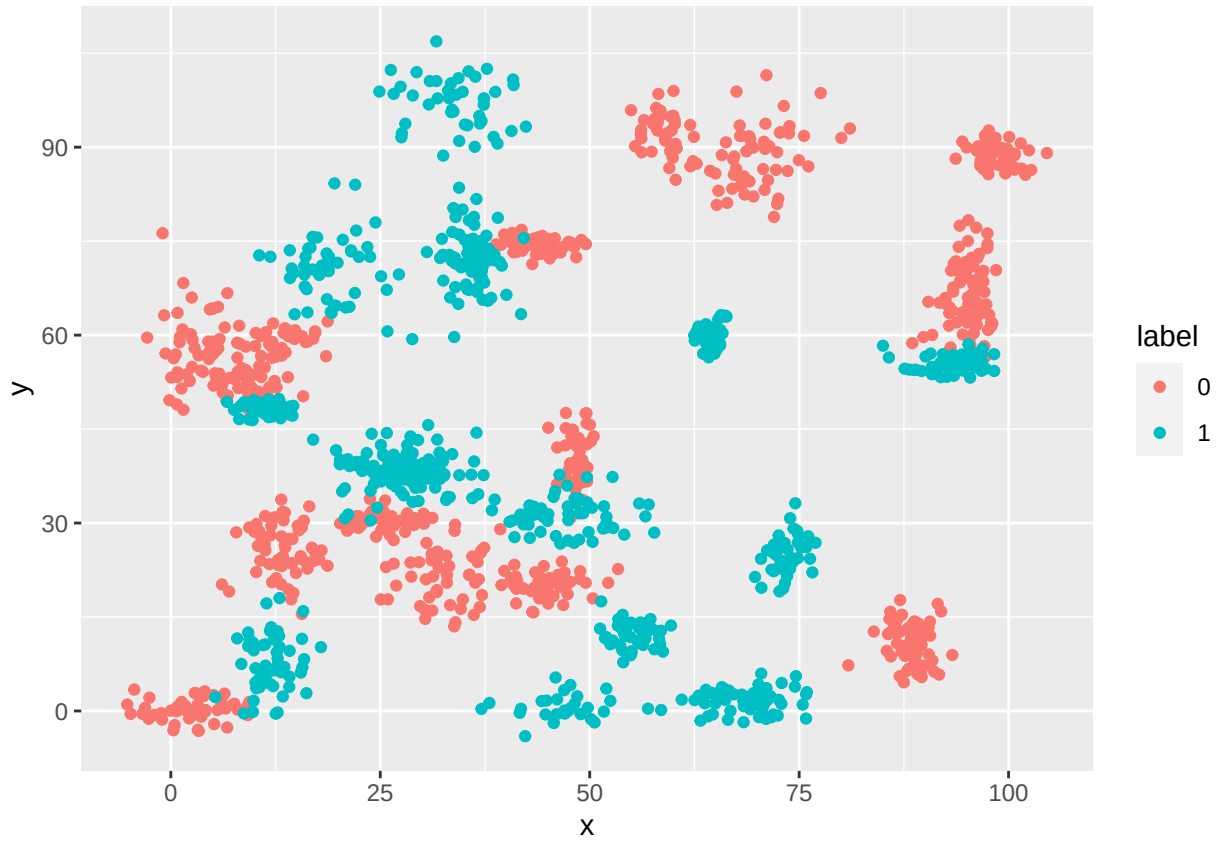
In this problem, you will use the nearest neighbors algorithm to fit a model on two simplified datasets. The first dataset (found in binary-classifier-data.csv) contains three variables; label, x, and y. The label variable is either 0 or 1 and is the output we want to predict using the x and y variables (You worked with this dataset last week!). The second dataset (found in trinary-classifier-data.csv) is similar to the first dataset except that the label variable can be 0, 1, or 2.

```
## Loading required package: ggplot2
```

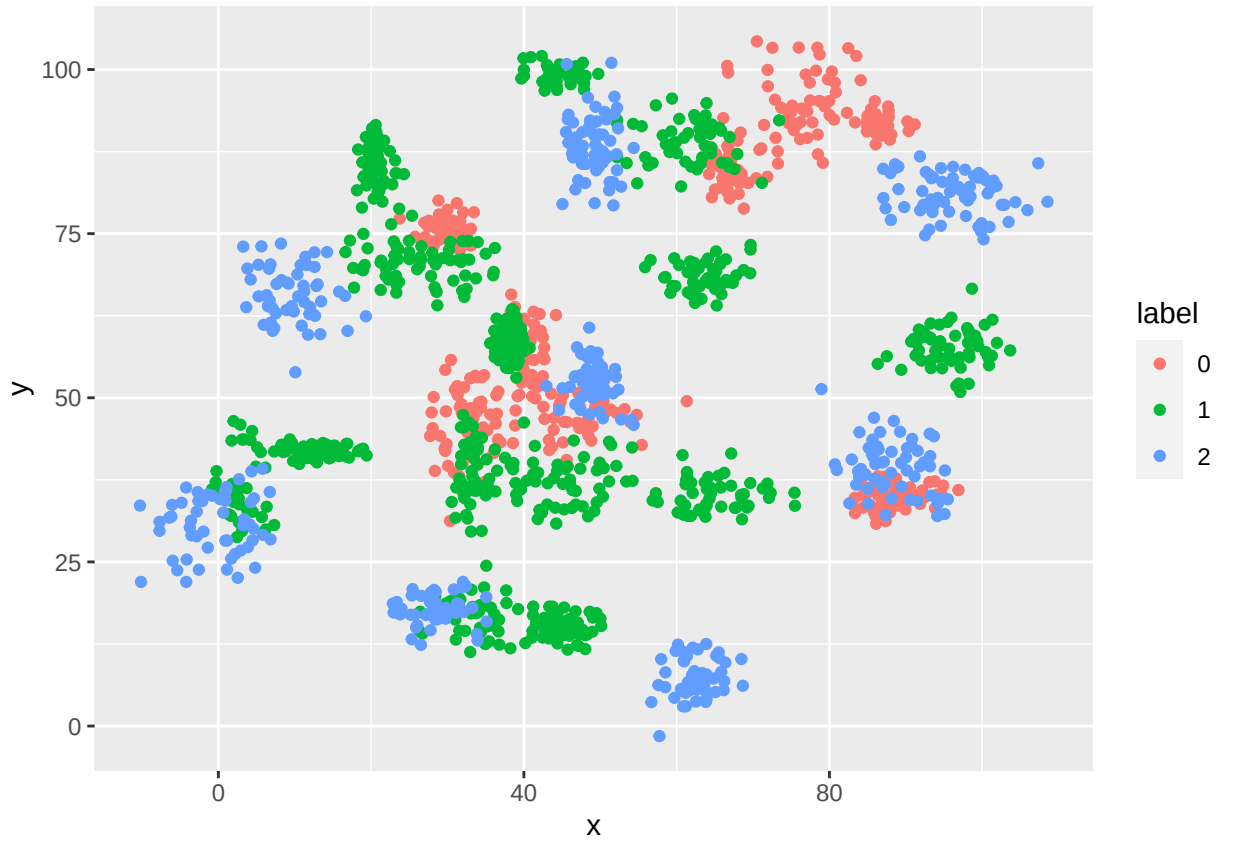
```
library(ggplot2)
library(caTools)
library(class)

# Let's set the training data arbitrarily to 75% of the whole dataset,
#  and the remaining 25% for the testing data.

setwd("C:/users/pahme/onedrive/documents/github/dsc520/data")
binclasdata <- read.csv("binary-classifier-data.csv")
binclasdata$label <- as.factor((binclasdata$label))
bdiv <- sample.split(binclasdata[,1],SplitRatio = .75)
btrain <- binclasdata[bdiv,]
btest <- binclasdata[!bdiv,]

triclasdata <- read.csv("trinary-classifier-data.csv")
triclasdata$label <- as.factor(triclasdata$label)
tdiv <- sample.split(triclasdata[,1], SplitRatio = .75)
ttrain <- triclasdata[tdiv,]
ttest <- triclasdata[!tdiv,]
```

Plot the data from each dataset using a scatter plot.

```
(bscatter <- ggplot(data=binclasdata, aes(x=x, y=y, color=label)) + geom_point())
```




```
(tscatter <- ggplot(data=triclasdata, aes(x=x, y=y, color=label)) + geom_point())
```



This indicates that while there are clusters of data, they don't line up cleanly on two sides of a dividing line.

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. I think no need to scale in this case

```
bpred <- knn(btrain[2:3], btest[2:3], k=1, cl=btrain$label)

(acctable <- table(bpred, btest$label))
```

```
##
## bpred   0   1
##     0 188  10
##     1   4 173
```

```
mean(bpred==btest$label)
```

```
## [1] 0.9626667
```

So we see that for the simplest method using only the single nearest neighbor, the accuracy was about 96%, which is terrific! Perhaps even that will improve if we use more neighbors.

Let's try various k values, from 1 to 30 let's say.

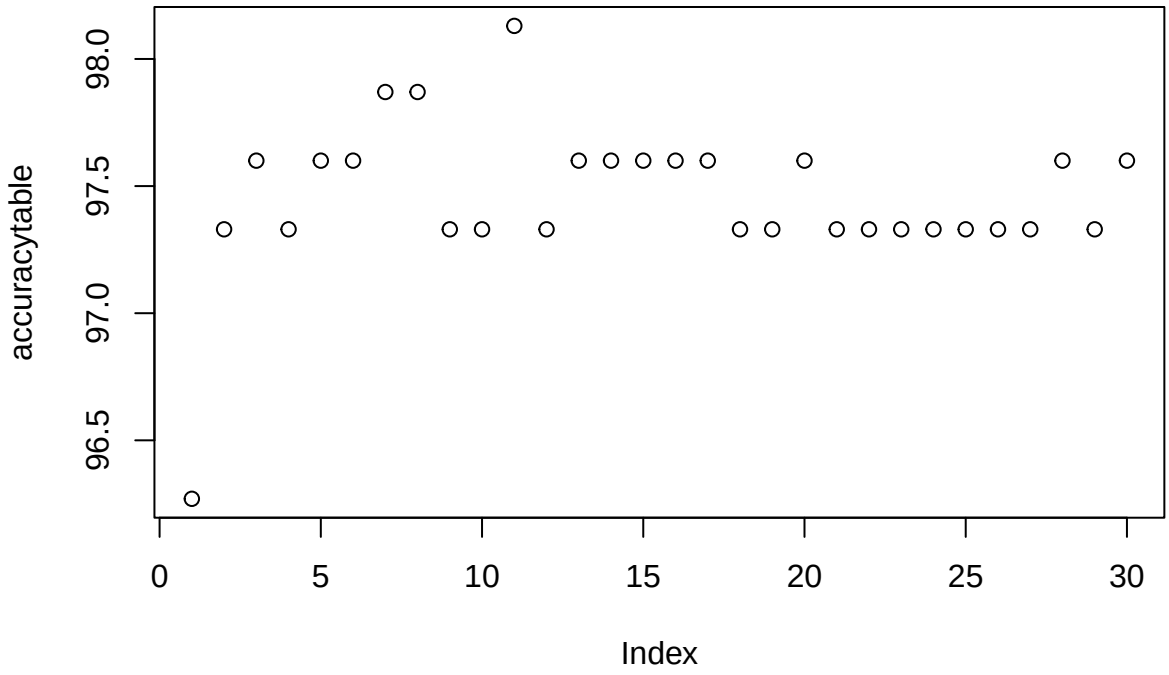
```
accuracytable <- c()
kopts <- c(1:30)

for (kval in kopts){
```

```
  bpredx <- knn(btrain[2:3], btest[2:3], k=kval, cl=btrain$label)
  accuracytable[kval] <- round(mean(bpredx== btest$label)*100,2)
}
accuracytable
```

```
##  [1] 96.27 97.33 97.60 97.33 97.60 97.60 97.87 97.87 97.33 97.33 98.13 97.33
## [13] 97.60 97.60 97.60 97.60 97.60 97.33 97.33 97.60 97.33 97.33 97.33 97.33
## [25] 97.33 97.33 97.33 97.60 97.33 97.60
```

This suggests that we probably didn't gain much by increasing the number of neighbors. Let's see that visually:

```
plot(accuracytable)
```



Actually there is a benefit to using more neighbors, but this is very variable- If we try this again, we might get very different results here. Actually, notice the accuracy scale along the y-axis- the numbers are really very close to each other and the range of accuracies is not wide.

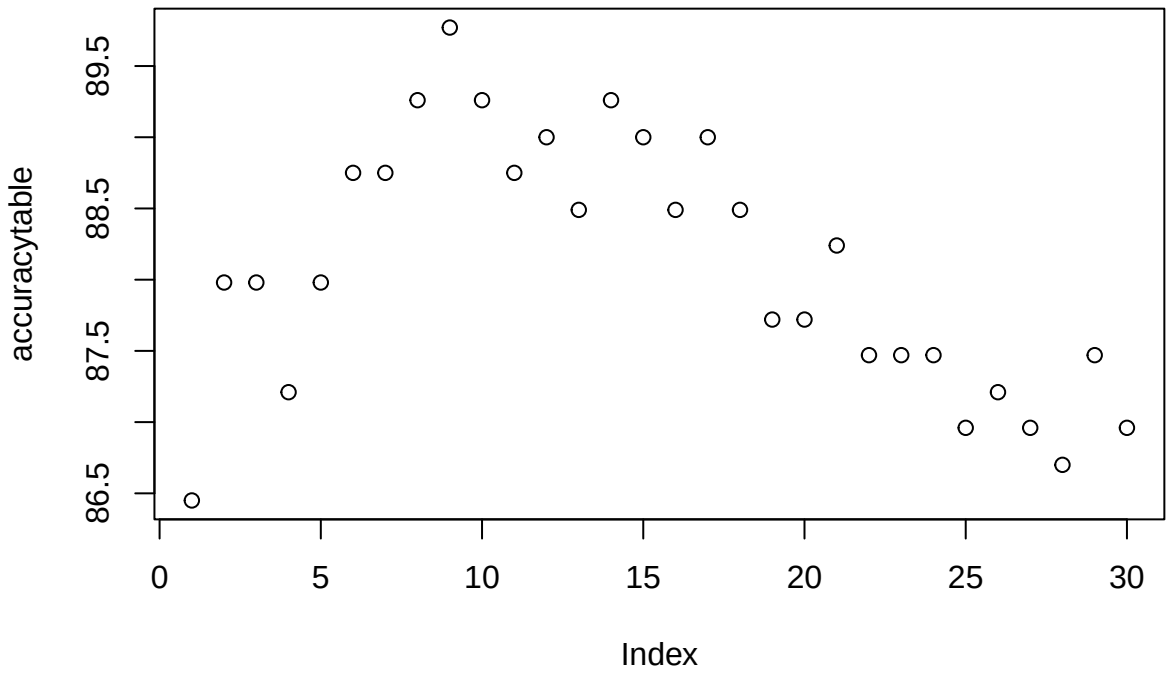
Now let's do the same for the triclass dataset:

```
accuracytable <- c()
kopts <- c(1:30)

for (kval in kopts){
  tpredx <- knn(ttrain[2:3], ttest[2:3], k=kval, cl=ttrain$label)
  accuracytable[kval] <- round(mean(tpredx== ttest$label)*100,2)
```

```
}
accuracytable
```

```
##  [1] 86.45 87.98 87.98 87.21 87.98 88.75 88.75 89.26 89.77 89.26 88.75 89.00
## [13] 88.49 89.26 89.00 88.49 89.00 88.49 87.72 87.72 88.24 87.47 87.47 87.47
## [25] 86.96 87.21 86.96 86.70 87.47 86.96
```

```
plot(accuracytable)
```



So for this dataset, there might be some improvement in accuracy using a few more neighbors, and possibly also a decrease in accuracy! However, this is not consistent: when running this again and again, we obtain totally different results. Still, just as before- the range of accuracies is narrow.

Decision boundary

Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?
Not at all. Clearly, there is no place to divide the categories by a straight line- whether the biclass or the triclass dataset.

How does the accuracy of your logistic regression classifier from last week compare? Why is the accuracy different between these two methods?
Using the logistic regression method we got as high as about 64% accuracy, but it is very poor compared with this nearest neighbor method, precisely because the groups don't easily divide along any relatively smooth divider.

Clustering

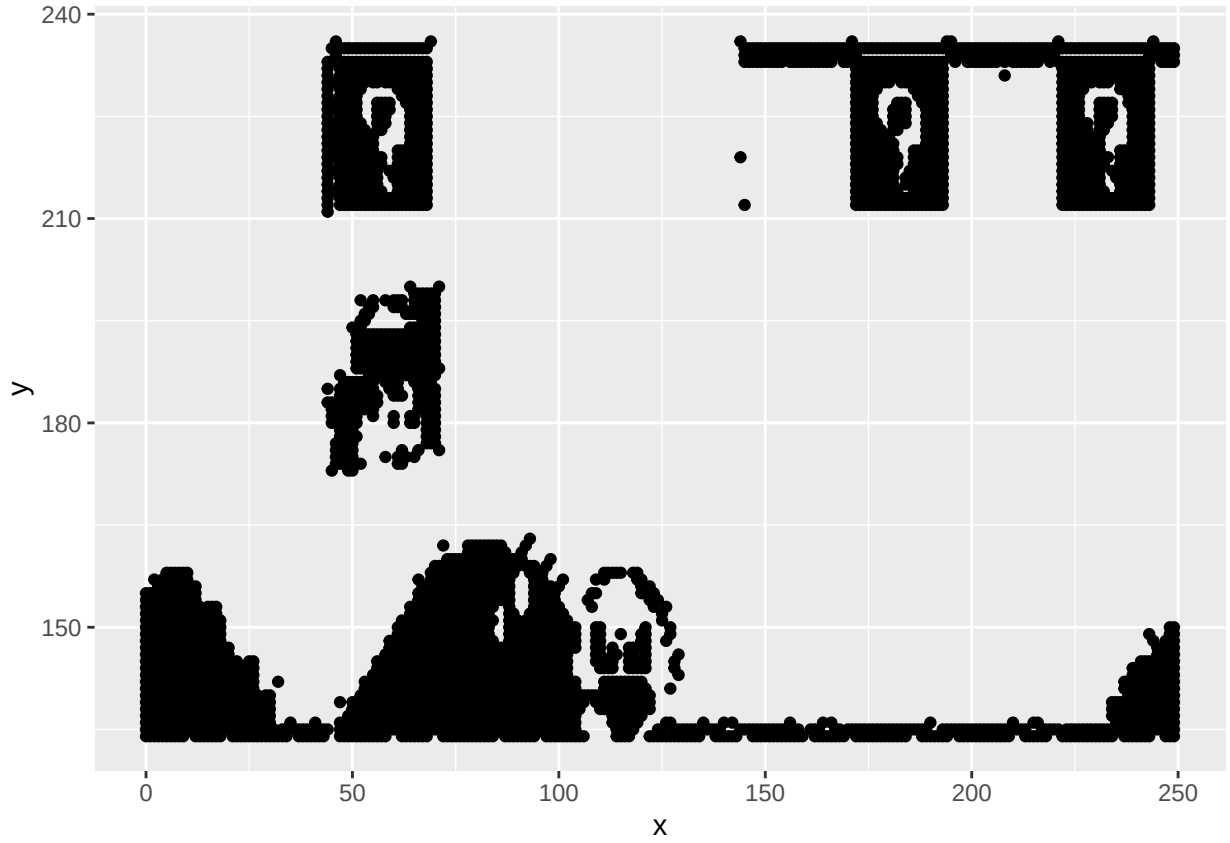
Labeled data is not always available. For these types of datasets, you can use unsupervised algorithms to extract structure. The k-means clustering algorithm and the k nearest neighbor algorithm both use the Euclidean distance between points to group data points. The difference is the k-means clustering algorithm does not use labeled data.

```
setwd("C:/users/pahme/onedrive/documents/github/dsc520/data")
clusdata <- read.csv("clustering-data.csv")
```

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.

```
ggplot(data=clusdata, aes(x=x, y=y)) + geom_point()
```

Plot the dataset using a scatter plot.

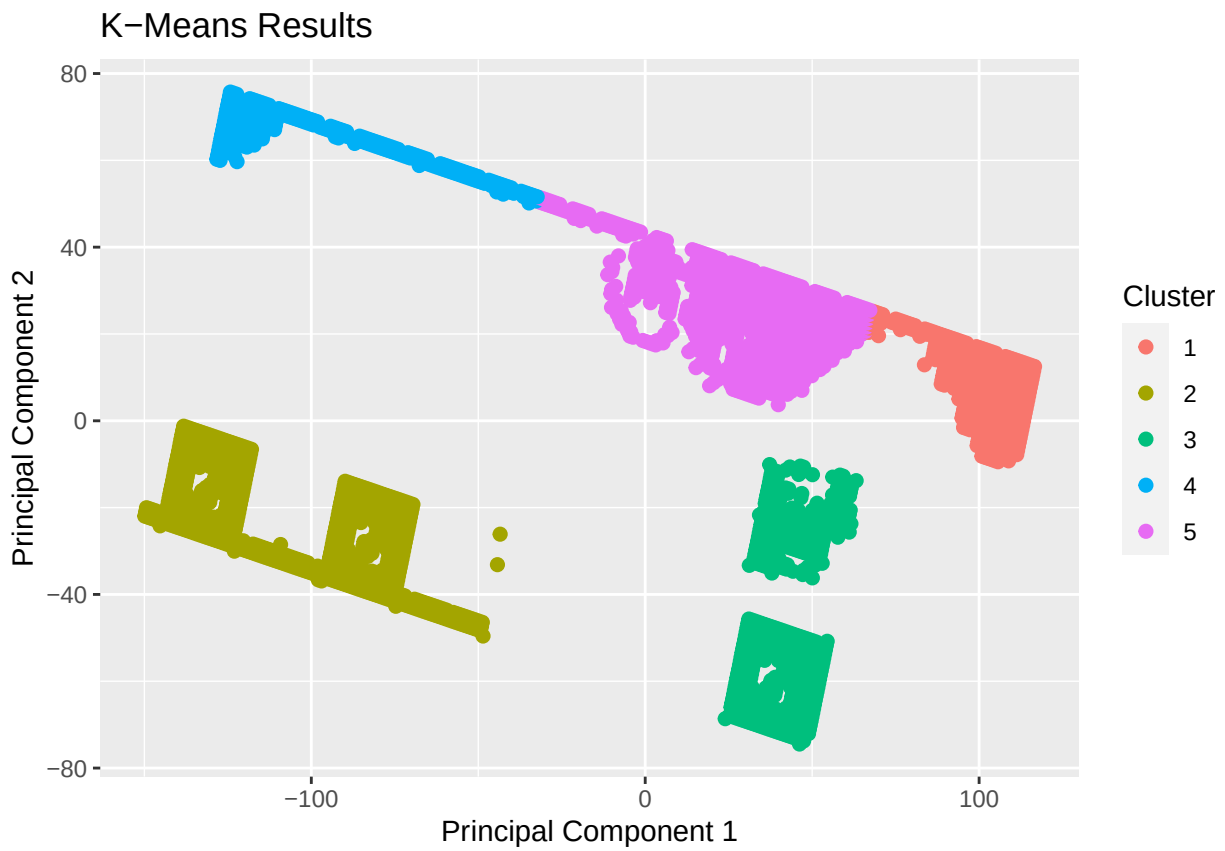


Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.

First let's see what we're talking about here. Let's try arbitrarily 5 clusters just to get our bearings:

```
set.seed(4325)
# that was an arbitrary number

clus.k <- kmeans(x=clusdata, centers=5, nstart=30)
```

```
# that was also an arbitrary number of starts

plot(clus.k, data=clusdata)
```

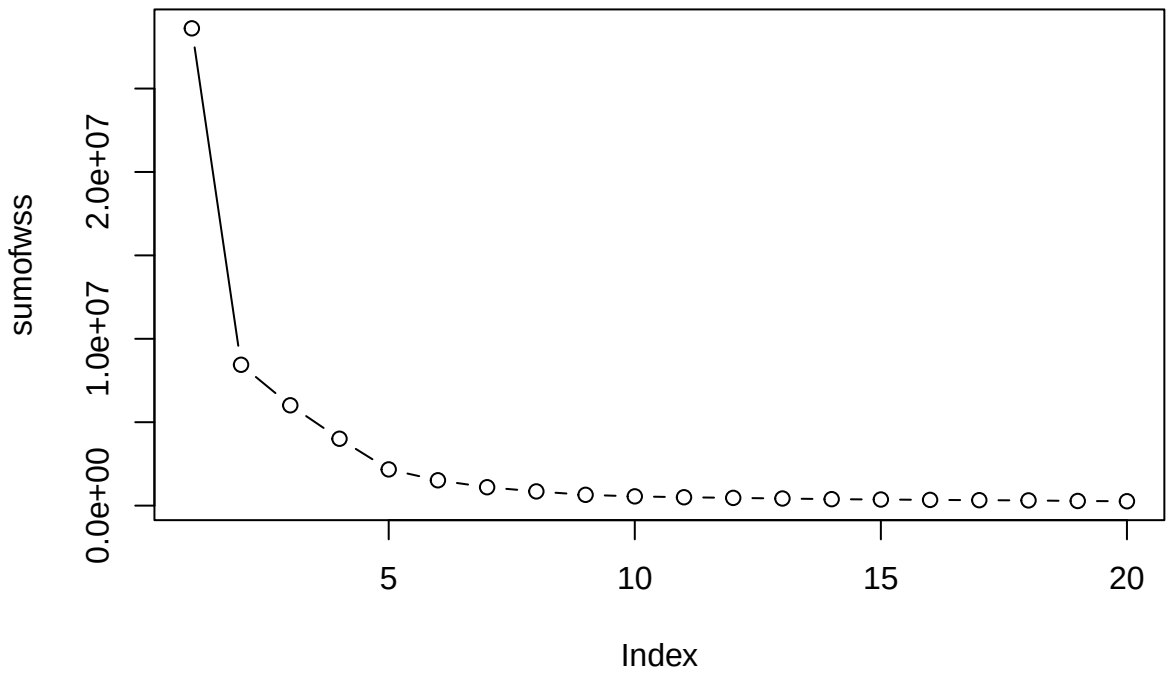


Well it's hard to know what to do with this... also the values don't line up...
Let's run through n-clusters from 2 to 20:

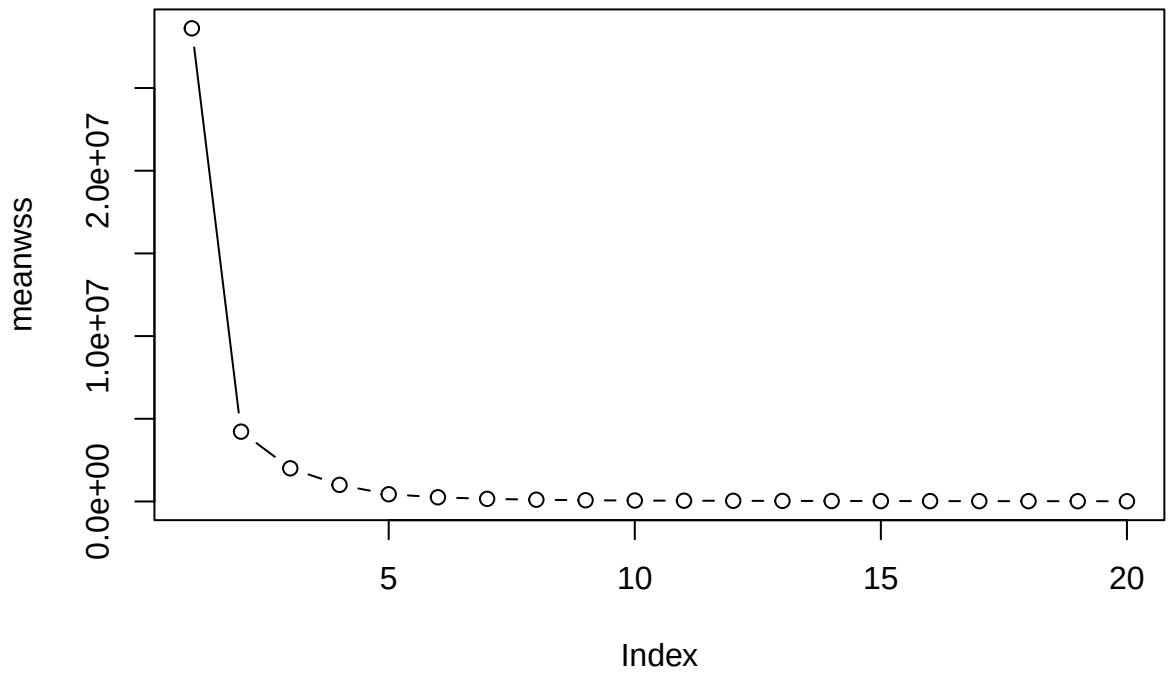
```
sumofwss <- c()
meanwss <- c()
acc <- c()

kopts <- c(1:20)

for (kval in kopts){
  clus.k <- kmeans(x=clusdata, centers=kval, nstart=10)
  sumofwss[kval] <- round(sum(clus.k$withinss))
  meanwss[kval] <- round(mean(clus.k$withinss))
  acc[kval] <- round(sum(clus.k$totss)/sum(clus.k$betweenss),2)
}
```

How will we measure improvement? Smallest sum of within-cluster distance? Smallest mean within-cluster distance? Smallest ratio of between-cluster distance to total distance? Let's plot them all and see:

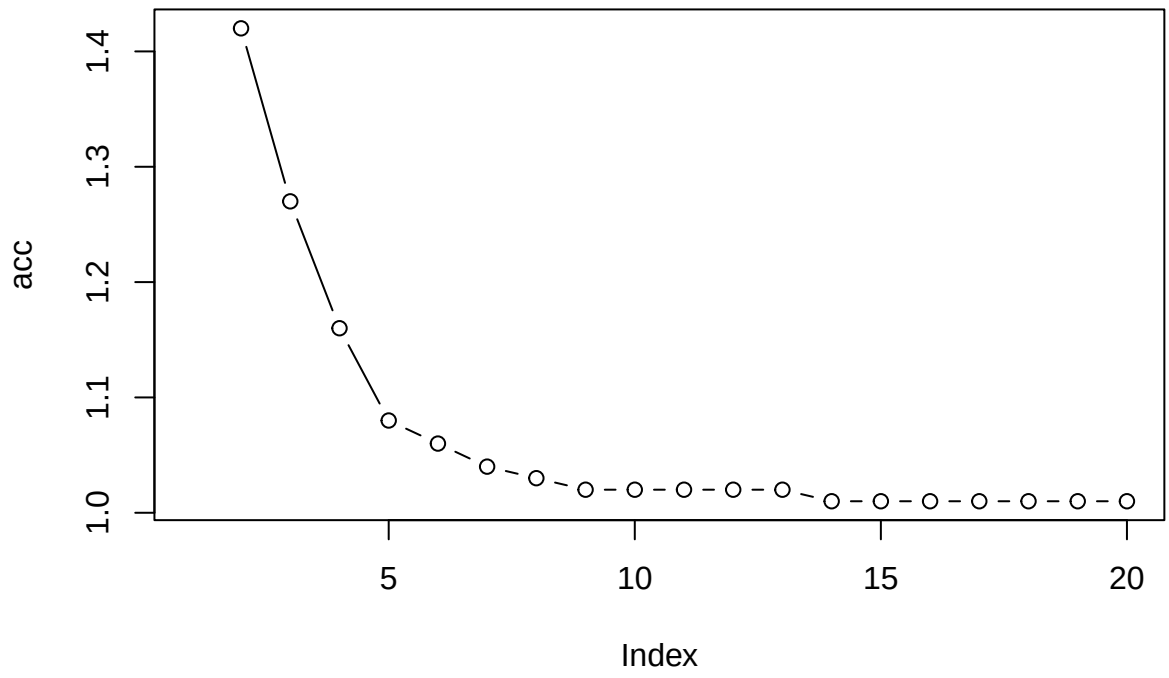
```
# sum of within cluster distances
plot(sumofwss, type="b")
```




```
# mean within cluster distances
plot(meanwss, type="b")
```



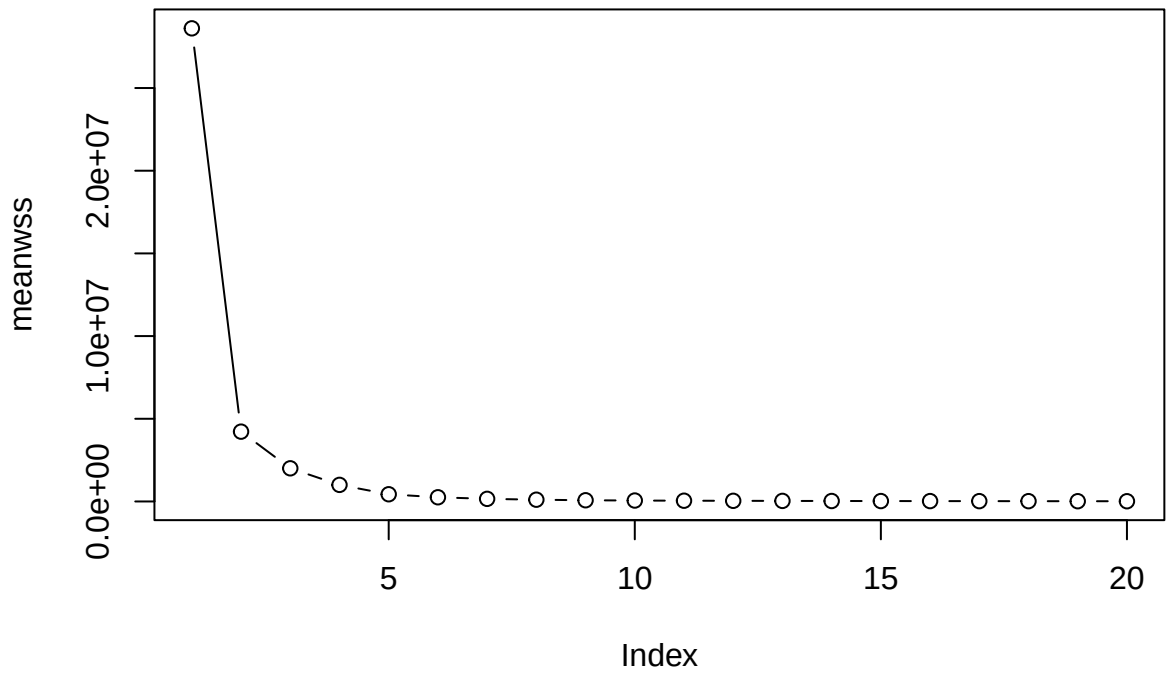

```
# accuracy- ratio of total distance to between distance
plot(acc, type="b")
```



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.
Well that settles it- the smallest mean within-cluster distance.

```
plot(meanwss, type="b")
```

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.



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?
It seems to fall at either 2 or 3 clusters.