Assignment: 10.2 Exercise

Raghuwanshi, Prashant

2021-08-14

#Introduction to Machine Learning

```
### library
library(ggplot2)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(class)
```

K Nearest Neighbor

binclass

```
### Set the working directory to the root of your DSC 520 directory

setwd("D:/MS_DataScience/DSC 520-Datastatiics.pdf/dsc520")
### Load the `data/binary-classifier-data.csv` to
binclass_df <- read.csv("data/binary-classifier-data.csv")
str(binclass_df)
```

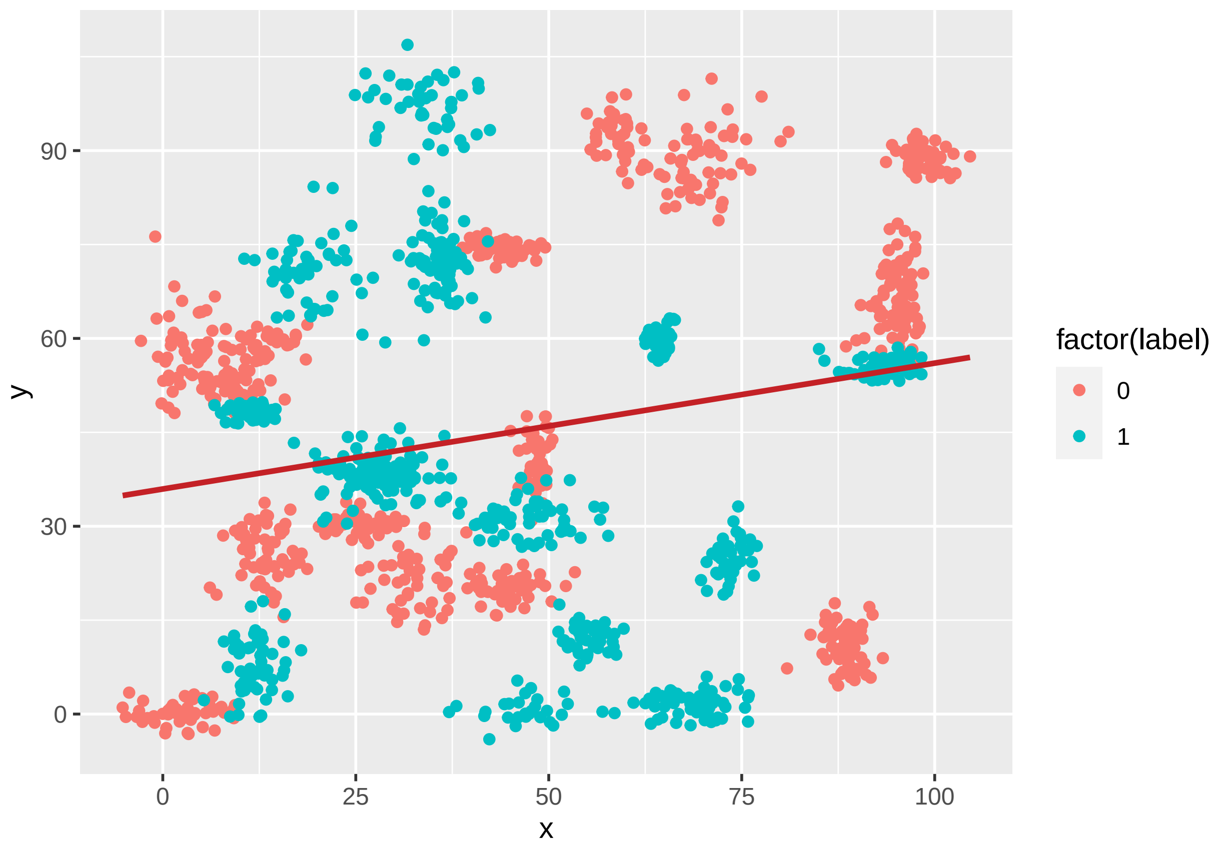
```
## 'data.frame':    1498 obs. of  3 variables:
##  $ label: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ x    : num  70.9 75 73.8 66.4 69.1 ...
##  $ y    : num  83.2 87.9 92.2 81.1 84.5 ...
```

Plot the data from each dataset using a scatter plot.

scatter plot for binclass_df (with fitted values)

```
ggplot(binclass_df, aes(x=x, y=y)) + geom_point(aes(color = factor(label))) + stat_smooth(method = "lm"
```

```
## `geom_smooth()` using formula 'y ~ x'
```



binclass_df - 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.

```
set.seed(123)
binclass <- sample(1:nrow(binclass_df),size=nrow(binclass_df)*0.7,replace = FALSE) #random selection of

train.binclass <- binclass_df[binclass,] # 70% training data
test.binclass <- binclass_df[-binclass,] # remaining 30% test data

### Creating seperate dataframe for 'Creditability' feature which is our target.
train.binclass_labels <- binclass_df[binclass,1]
test.binclass_labels <-binclass_df[-binclass,1]

### 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.
### Find the number of observation
NROW(train.binclass_labels)
```

```
## [1] 1048
```

```
###
binclass.knn.3 <- knn(train=train.binclass, test=test.binclass, cl=train.binclass_labels, k=3)
binclass.knn.5 <- knn(train=train.binclass, test=test.binclass, cl=train.binclass_labels, k=5)
binclass.knn.15 <- knn(train=train.binclass, test=test.binclass, cl=train.binclass_labels, k=15)
binclass.knn.20 <- knn(train=train.binclass, test=test.binclass, cl=train.binclass_labels, k=20)
```

```
binclass.knn.25 <- knn(train=train.binclass, test=test.binclass, cl=train.binclass_labels, k=25)
binclass.knn.33 <- knn(train=train.binclass, test=test.binclass, cl=train.binclass_labels, k=33)
### Calculate the proportion of correct classification for k = 3, 5,15,20,25
binclass.ACC.3 <- 100 * sum(test.binclass_labels == binclass.knn.3)/NROW(test.binclass_labels)
binclass.ACC.5 <- 100 * sum(test.binclass_labels == binclass.knn.5)/NROW(test.binclass_labels)
binclass.ACC.15 <- 100 * sum(test.binclass_labels == binclass.knn.15)/NROW(test.binclass_labels)
binclass.ACC.20 <- 100 * sum(test.binclass_labels == binclass.knn.20)/NROW(test.binclass_labels)
binclass.ACC.25 <- 100 * sum(test.binclass_labels == binclass.knn.25)/NROW(test.binclass_labels)
binclass.ACC.33 <- 100 * sum(test.binclass_labels == binclass.knn.33)/NROW(test.binclass_labels)
```

Accuracy

```
binclass.ACC.3
```

```
## [1] 98
```

```
binclass.ACC.5
```

```
## [1] 97.55556
```

```
binclass.ACC.15
```

```
## [1] 97.55556
```

```
binclass.ACC.20
```

```
## [1] 98
```

```
binclass.ACC.25
```

```
## [1] 98.44444
```

```
binclass.ACC.33
```

```
## [1] 97.55556
```

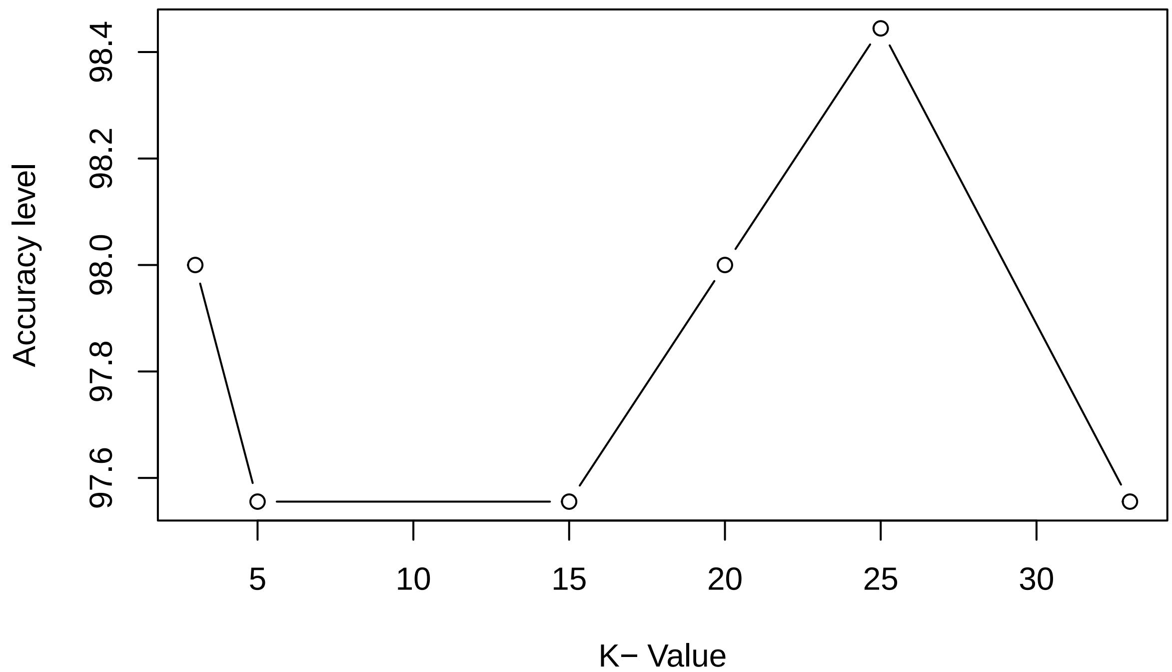
```
### Accuracy DF
binclass.acc.df = data.frame(c(3, 5, 15, 20, 25, 33), c(binclass.ACC.3, binclass.ACC.5, binclass.ACC.15
```

From the output you can see that for K = 25, we achieve the maximum accuracy, i.e. 98%

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.

Accuracy plot

```
plot(binclass.acc.df, type="b", xlab="K- Value",ylab="Accuracy level")
```



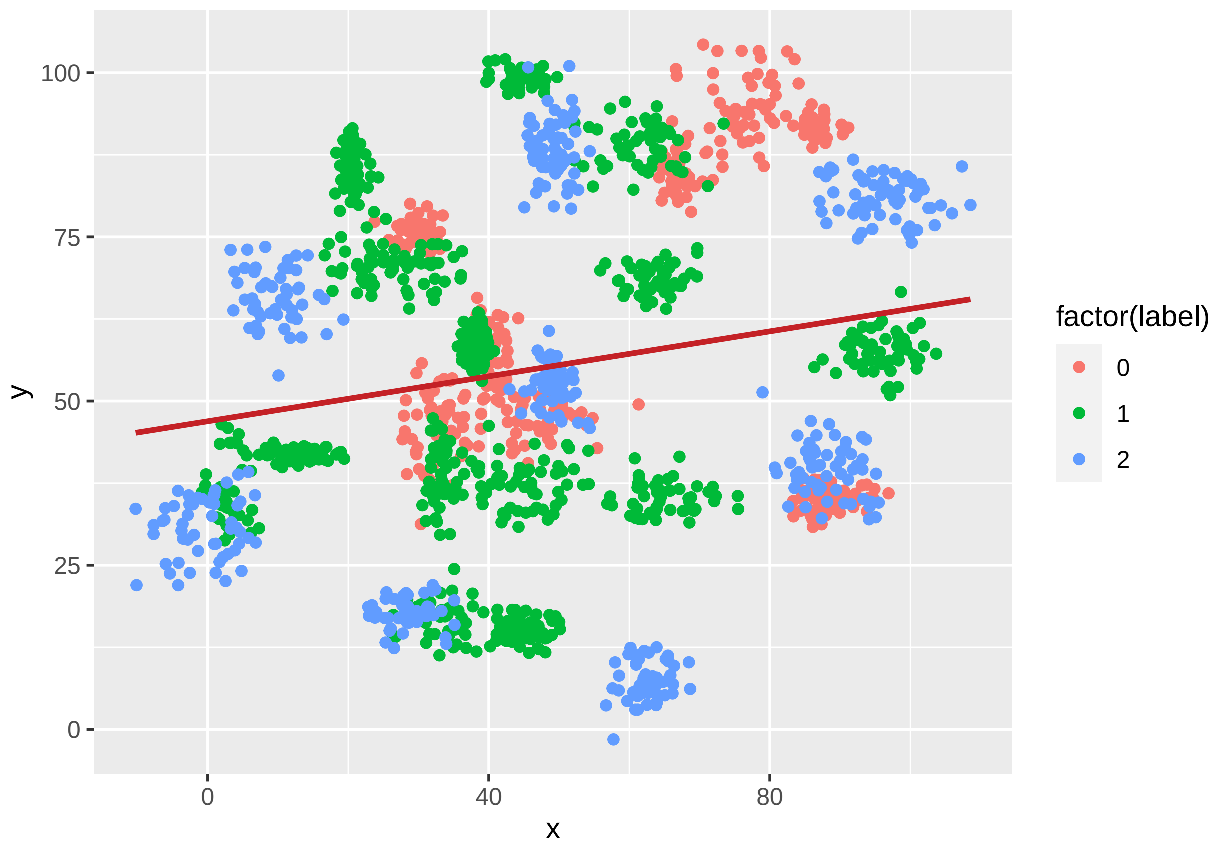
triclass

```
### Set the working directory to the root of your DSC 520 directory

setwd("D:/MS_DataScience/DSC 520-Datastatiics.pdf/dsc520")
### Load the `data/ trinary-classifier-data.csv` to
triclass_df <- read.csv("data/trinary-classifier-data.csv")
str(triclass_df)
```

```
## 'data.frame':    1568 obs. of  3 variables:
##  $ label: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ x    : num  30.1 31.3 34.1 32.6 34.7 ...
##  $ y    : num  39.6 51.8 49.3 41.2 45.5 ...
```

```
### scatter plot for triclass_d (with fitted values)
ggplot(triclass_df, aes(x=x, y=y)) +
  geom_point(aes(color = factor(label))) + stat_smooth(method = "lm", col = "#C42126", se = FALSE, size
```

```
## `geom_smooth()` using formula 'y ~ x'
```



binclass_df - 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.

```
set.seed(123)
triclass <- sample(1:nrow(triclass_df),size=nrow(triclass_df)*0.7,replace = FALSE) #random selection of

train.triclass <- triclass_df[triclass,] # 70% training data
test.triclass <- triclass_df[-triclass,] # remaining 30% test data

### Creating seperate dataframe for 'Creditability' feature which is our target.
train.triclass_labels <- triclass_df[triclass,1]
test.triclass_labels <-triclass_df[-triclass,1]

### 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.
### Find the number of observation
NROW(train.triclass_labels)
```

```
## [1] 1097
```

```
###
triclass.knn.3 <- knn(train=train.triclass, test=test.triclass, cl=train.triclass_labels, k=3)
triclass.knn.5 <- knn(train=train.triclass, test=test.triclass, cl=train.triclass_labels, k=5)
triclass.knn.15 <- knn(train=train.triclass, test=test.triclass, cl=train.triclass_labels, k=15)
triclass.knn.20 <- knn(train=train.triclass, test=test.triclass, cl=train.triclass_labels, k=20)
```

```
triclass.knn.25 <- knn(train=train.triclass, test=test.triclass, cl=train.triclass_labels, k=25)
triclass.knn.33 <- knn(train=train.triclass, test=test.triclass, cl=train.triclass_labels, k=33)
### Calculate the proportion of correct classification for k = 3, 5,15,20,25
triclass.ACC.3 <- 100 * sum(test.triclass_labels == triclass.knn.3)/NROW(test.triclass_labels)
triclass.ACC.5 <- 100 * sum(test.triclass_labels == triclass.knn.5)/NROW(test.triclass_labels)
triclass.ACC.15 <- 100 * sum(test.triclass_labels == triclass.knn.15)/NROW(test.triclass_labels)
triclass.ACC.20 <- 100 * sum(test.triclass_labels == triclass.knn.20)/NROW(test.triclass_labels)
triclass.ACC.25 <- 100 * sum(test.triclass_labels == triclass.knn.25)/NROW(test.triclass_labels)
triclass.ACC.33 <- 100 * sum(test.triclass_labels == triclass.knn.33)/NROW(test.triclass_labels)
```

Accuracy

```
triclass.ACC.3
```

```
## [1] 93.20594
```

```
triclass.ACC.5
```

```
## [1] 92.14437
```

```
triclass.ACC.15
```

```
## [1] 89.17197
```

```
triclass.ACC.20
```

```
## [1] 87.47346
```

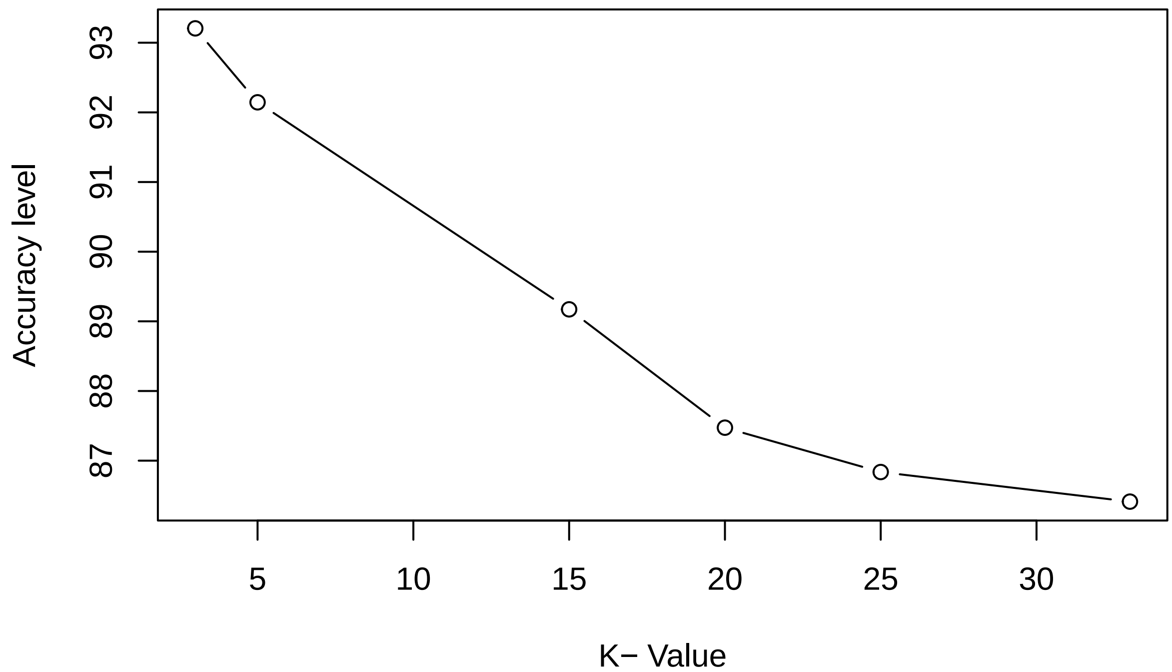
```
triclass.ACC.25
```

```
## [1] 86.83652
```

```
triclass.ACC.33
```

```
## [1] 86.41189
```

```
### Accuracy DF
triclass.acc.df = data.frame(c(3, 5, 15, 20, 25, 33), c(triclass.ACC.3, triclass.ACC.5, triclass.ACC.15
### From the output you can see that for K = 3, we achieve the maximum accuracy, i.e. 93%
### ### Plot the results in a graph where the x-axis is the different values of k and the y-axis is the
### Accuracy plot
plot(triclass.acc.df, type="b", xlab="K- Value",ylab="Accuracy level")
```



k-means algorithm

clustering-data

```
### Set the working directory to the root of your DSC 520 directory

setwd("D:/MS_DataScience/DSC 520-Datastatiics.pdf/dsc520")
### Load the `data/binary-classifier-data.csv` to
cluster_df <- read.csv("data/clustering-data.csv")
str(cluster_df)
```

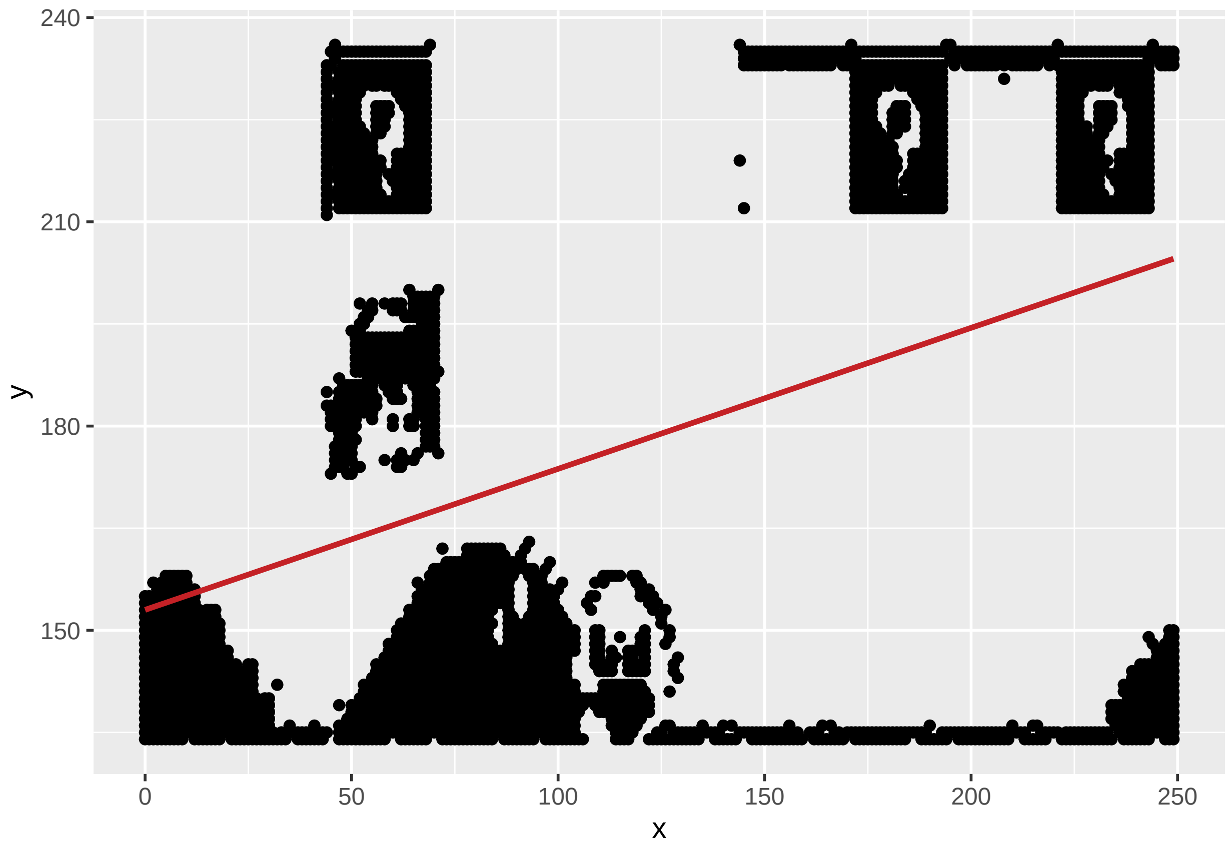
```
## 'data.frame':    4022 obs. of  2 variables:
##  $ x: int  46 69 144 171 194 195 221 244 45 47 ...
##  $ y: int  236 236 236 236 236 236 236 236 235 235 ...
```

```
cluster_mat <- data.matrix(cluster_df)
```

scatter plot for binclass_df (with fitted values)

```
ggplot(cluster_df, aes(x=x, y=y)) + geom_point() + stat_smooth(method = "lm", col = "#C42126", se = FALS
```

```
## `geom_smooth()` using formula 'y ~ x'
```



An optimal value of ???k??? is the value which gives us a converged set of clusters with minimum distortion. Greater the distortion, worse will be the clusters formed. ### The distortion can be calculated in terms of ???withinss??? from each of the clusters. ### Lesser the value of ???withinss??? of a particular cluster, more densely populated it will be, thus minimum distortion.

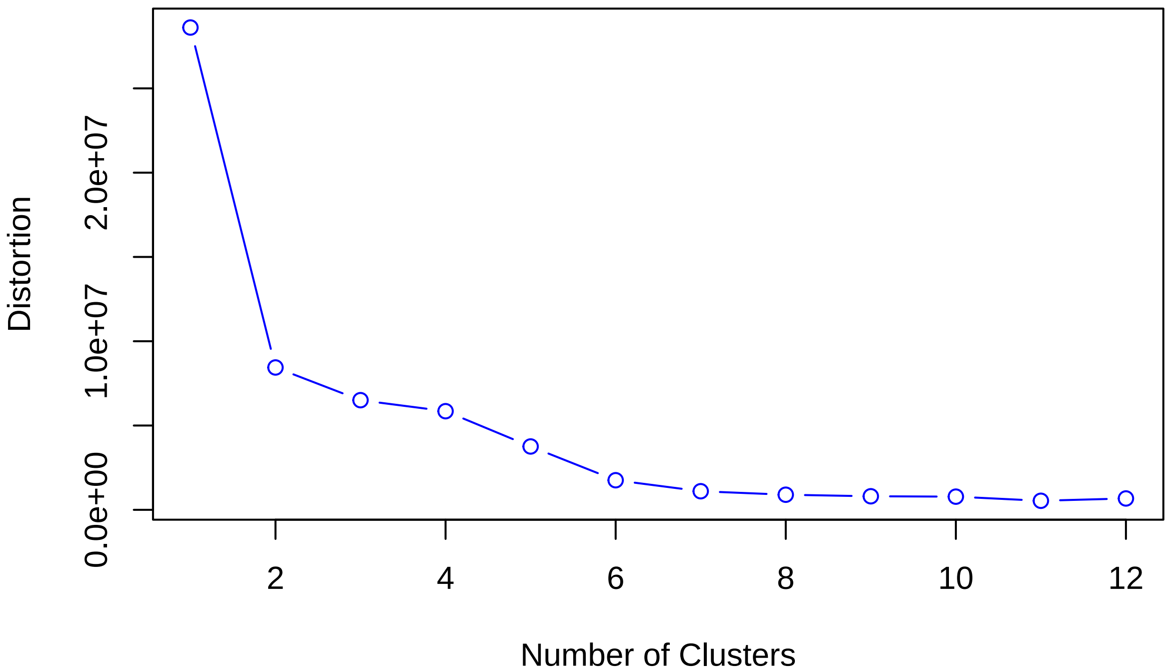
```
kmeans.wss.k <- function(cluster_mat, k){
  km = kmeans(cluster_mat, k)
  return (km$tot.withinss)
}

kmeans.dis <- function(cluster_mat, maxk){
    dis=(nrow(cluster_mat)-1)*sum(apply(cluster_mat,2,var))
    dis[2:maxk]=sapply (2:maxk, kmeans.wss.k, cluster_mat=cluster_mat)
    return(dis)
}
maxk = 12
dis = kmeans.dis(cluster_mat, maxk)
```

Elbow Curve:

This is the plot between ???k???, the number of clusters and the ???totwithinss??? (or distortion) for each value of k. You can see when the number of cluster is less, there is a gradual decrease in distortion but as we keep on increasing the value of k, the rate of reduction of distortion values becomes constant.

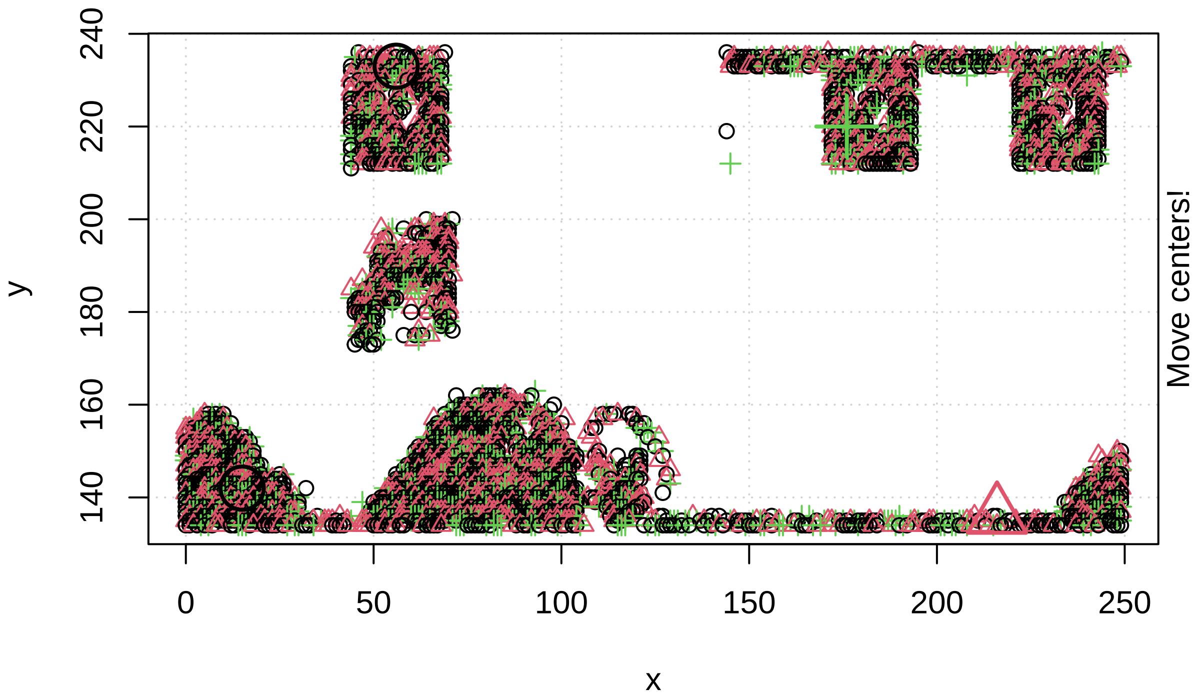
Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.

```
plot(1:maxk, dis, type='b', xlab="Number of Clusters", ylab="Distortion", col="blue")
```

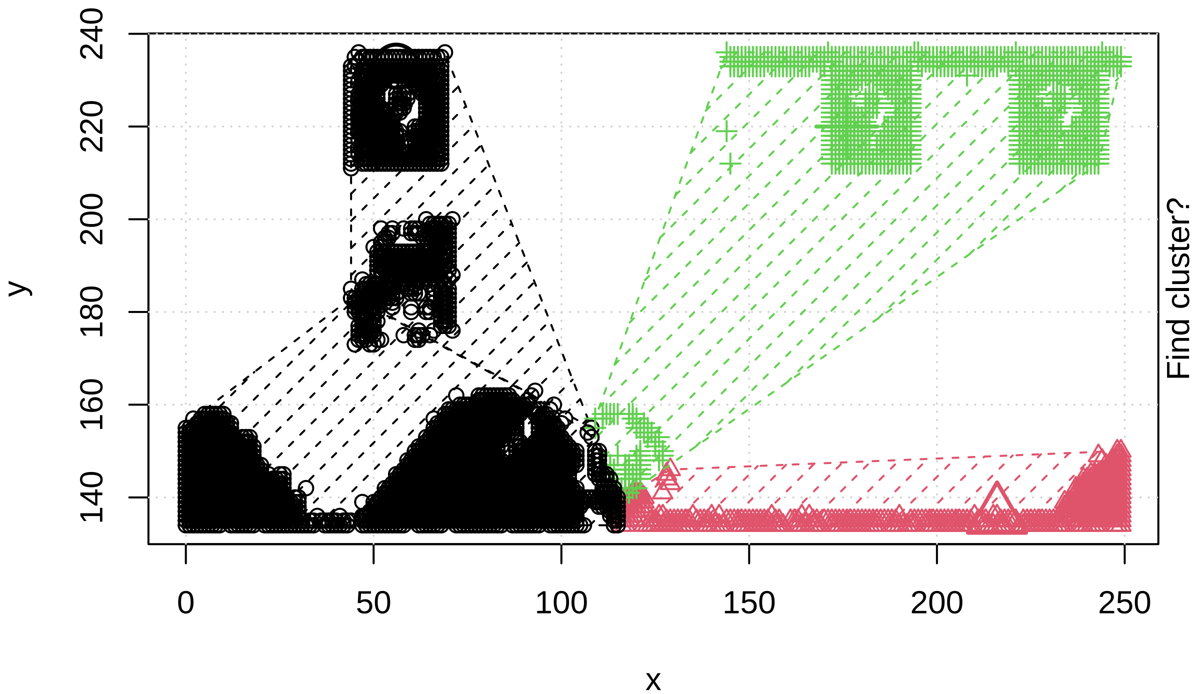


This value of k beyond which the distortion rate becomes constant is the optimal value. Here k=4

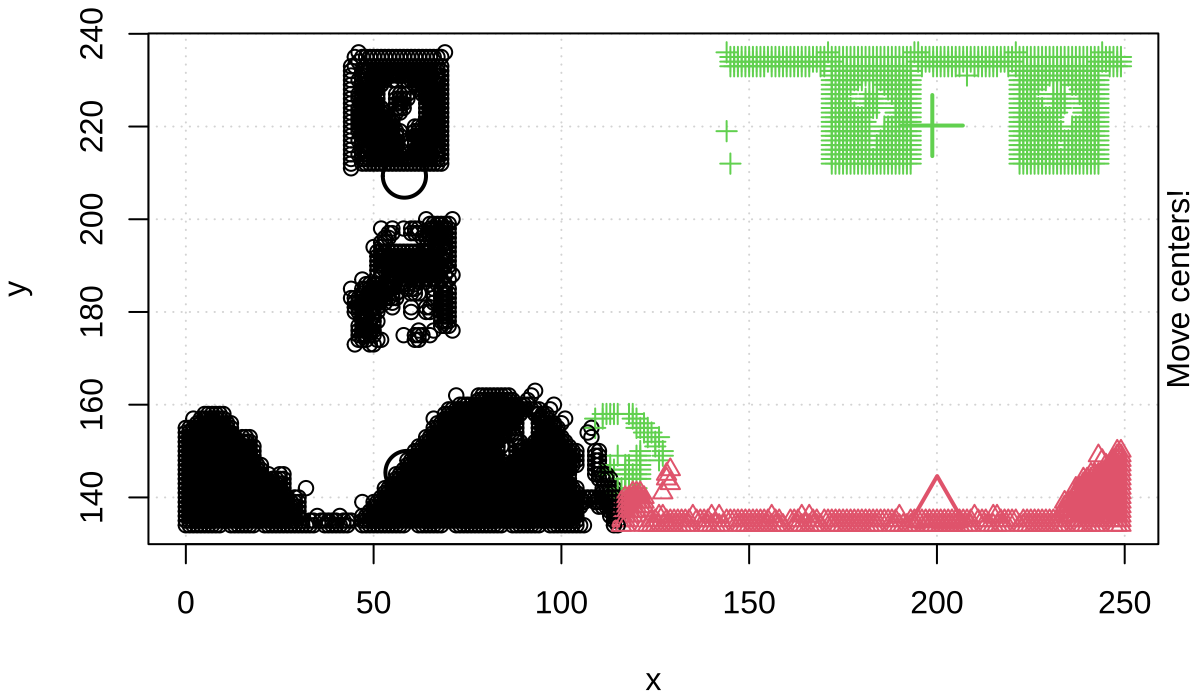
```
library(animation)
### Step 1: R randomly chooses four points
### Step 2: Compute the Euclidean distance and draw the clusters. You have one cluster in green at the 
### Step 3: Compute the centroid, i.e. the mean of the clusters
### Repeat until no data changes cluster
cl<- kmeans.ani(cluster_mat, 4)
```



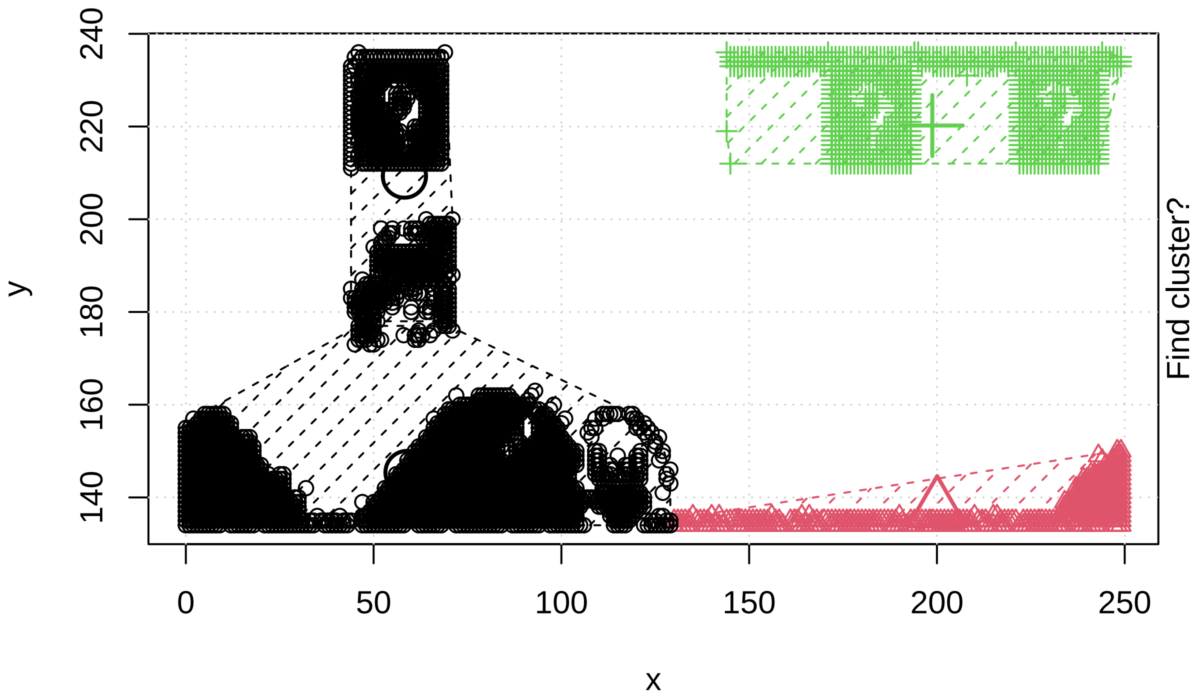
Move centers!
x
y
0
50
100
150
200
250
140
160
180
200
220
240



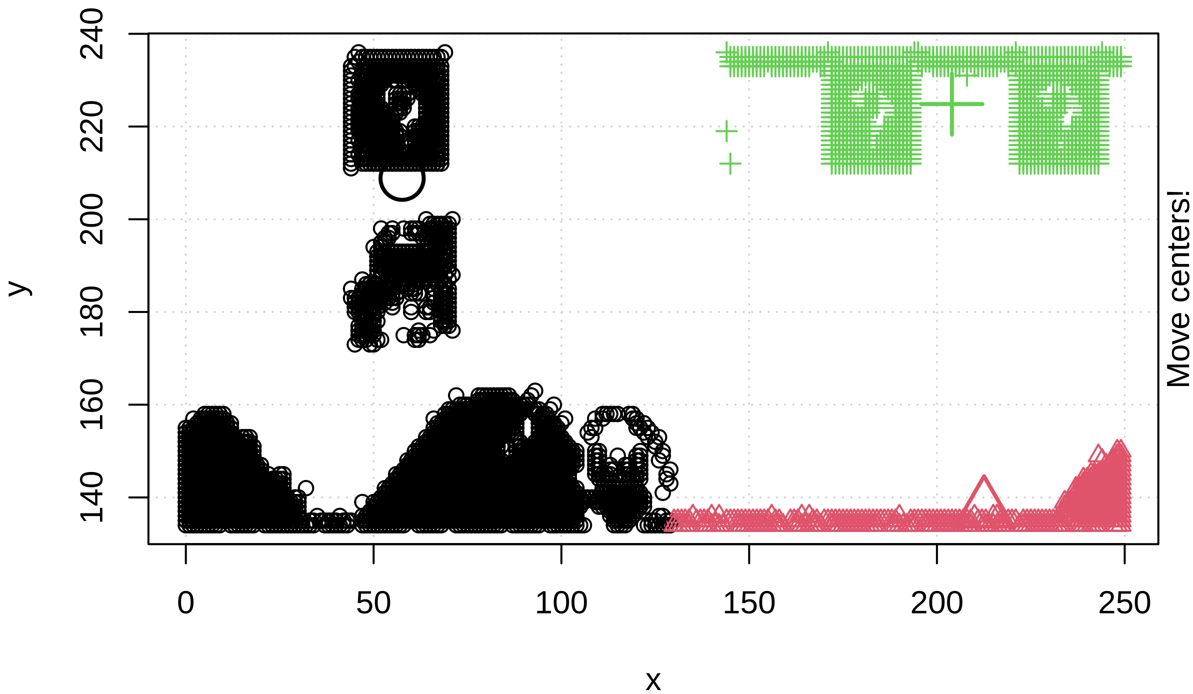
240
220
200
180
160
140
y
0
50
100
150
200
250
x
Find cluster?



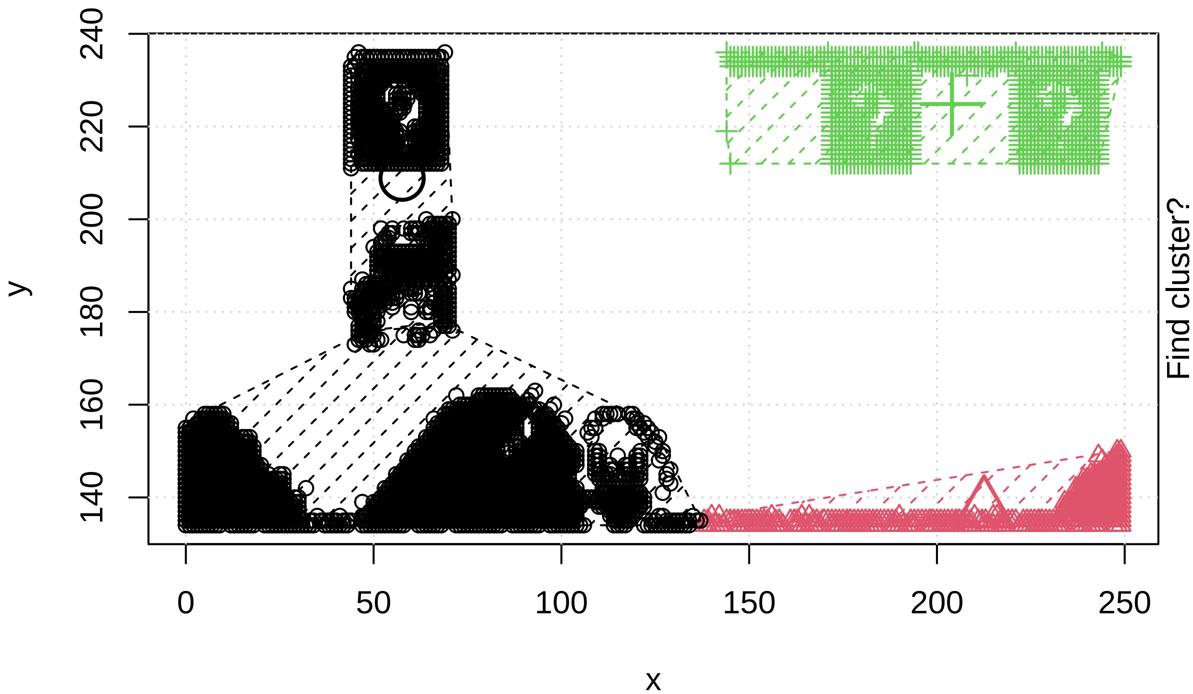
240
220
200
180
160
140
y
0
50
100
150
200
250
x
Move centers!

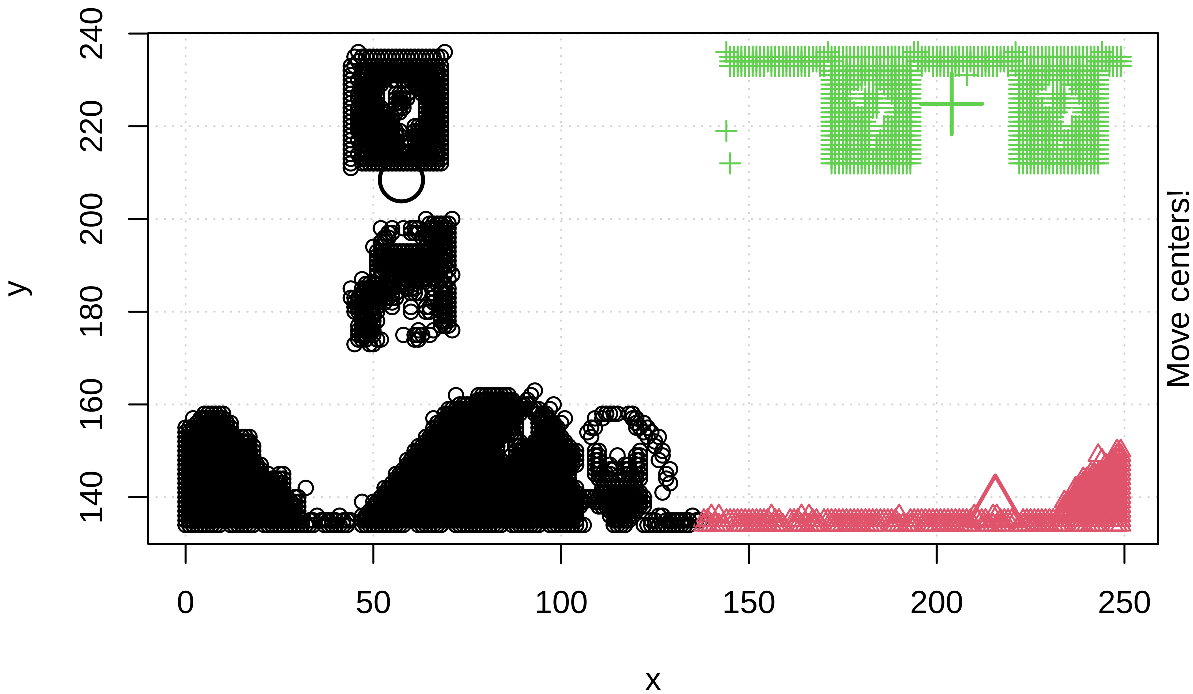


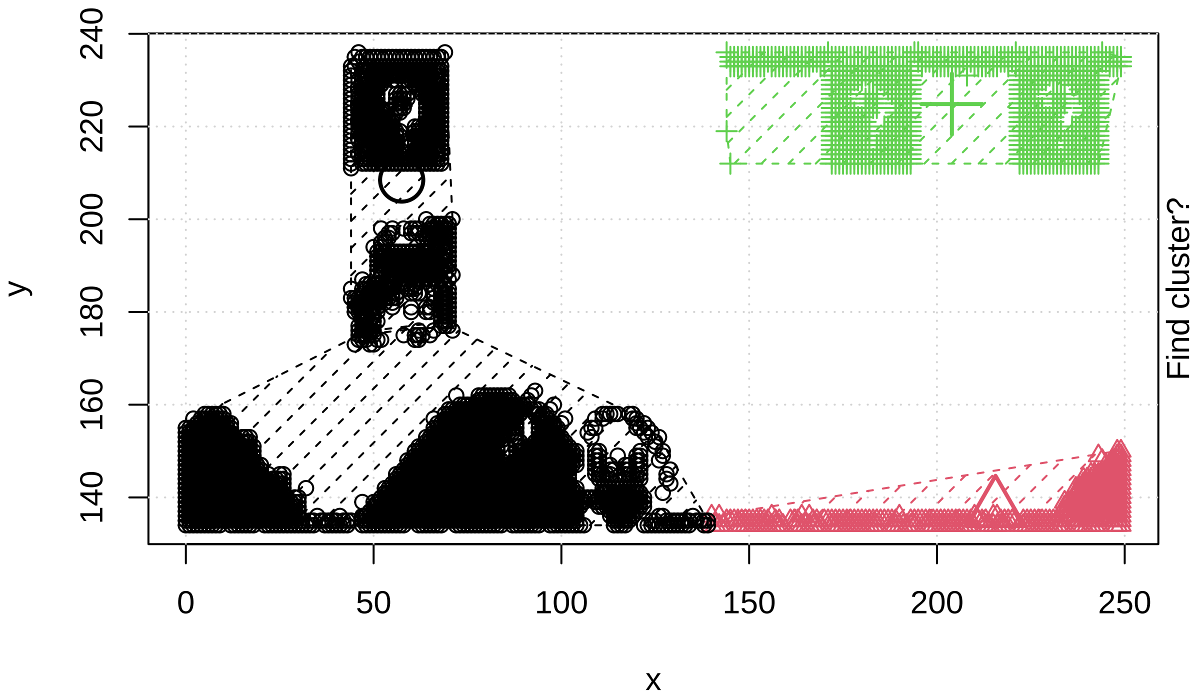
240
220
200
180
160
140
y
0
50
100
150
200
250
x
Find cluster?



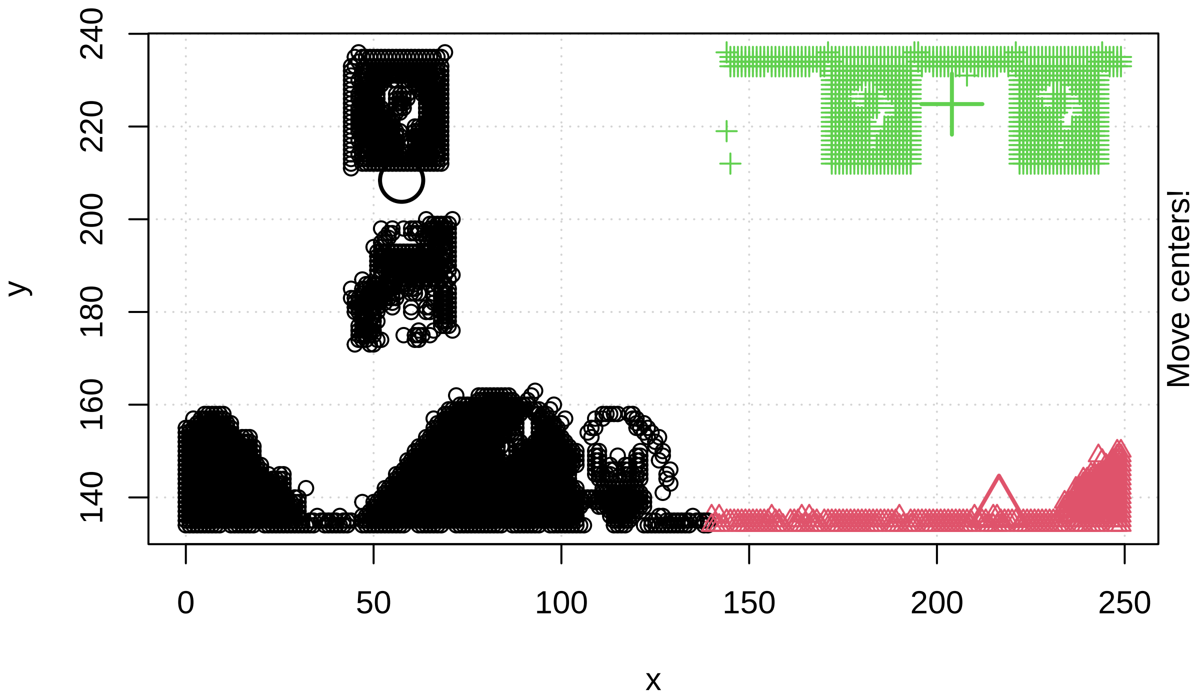
240
220
200
180
160
140
y
0
50
100
150
200
250
x
Move centers!


240
220
200
180
160
140
y
0
50
100
150
200
250
x
Find cluster?

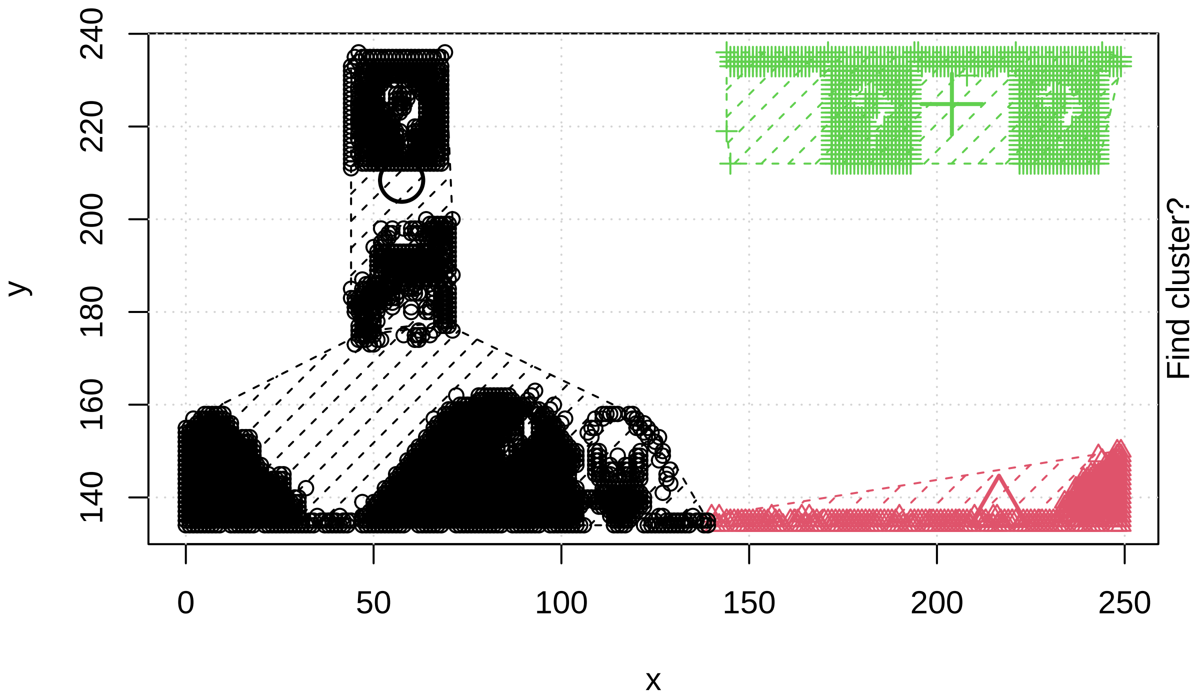

Move centers!
y
x
240
220
200
180
160
140
0
50
100
150
200
250



240
220
200
180
160
140
y
0
50
100
150
200
250
x
Find cluster?



240
220
200
180
160
140
y
0
50
100
150
200
250
x
Move centers!



It can be seen that the data is divided into 4 clusters. The cluster centers are

```
cl$centers
```

```
##              x        y
## [1,]  57.46296 208.4206
## [2,] 216.51786 137.4583
## [3,] 203.98579 224.8406
## [4,]  63.60771 144.7100
```