AC53013 KMeans Investigation

Robert Meredith

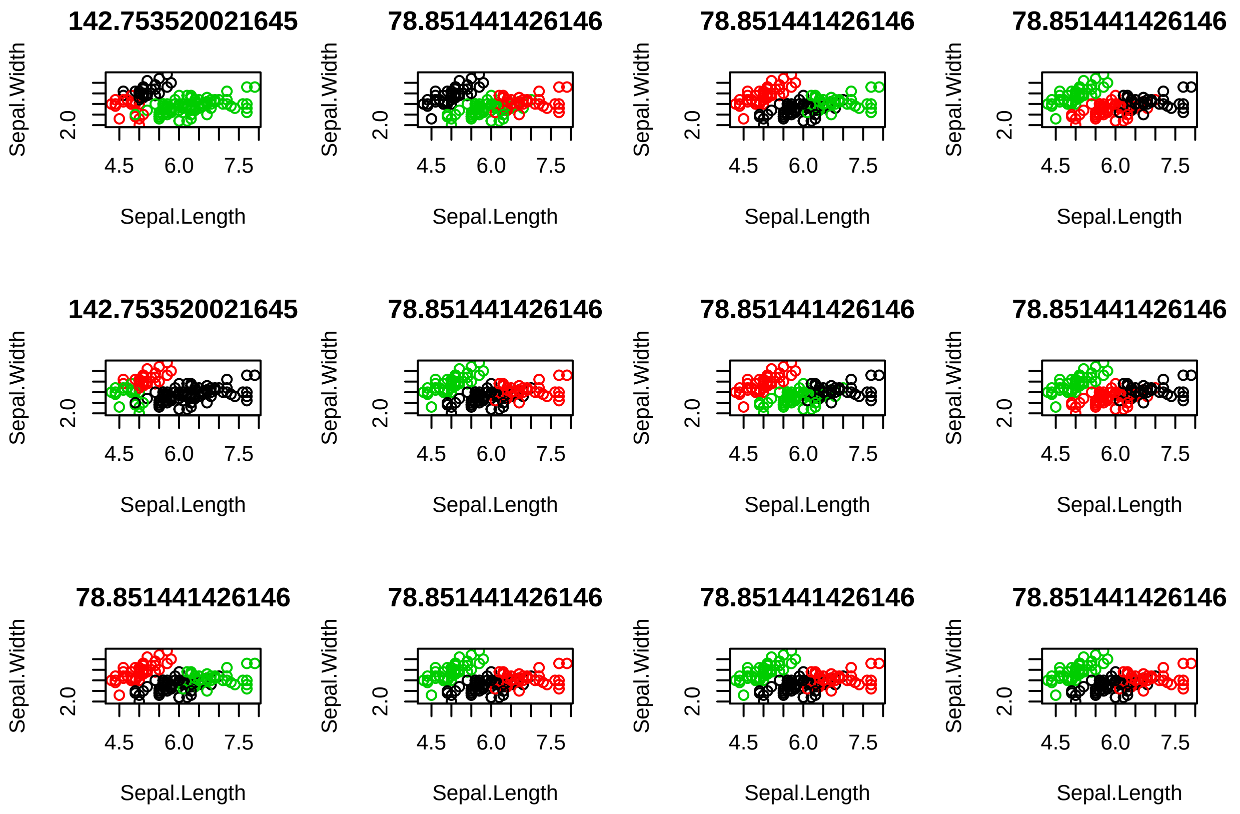
1st March 2020

Introduction

In this assignment I will examine the output of multiple runs of the R K Means algorithm on the Iris dataset and discuss how Kmeans is working on this data. I will illustrate this through the use of various visualisations.I will assume the reader knows about the KMeans algorithm and not go into choice of number of clusters.

Initial Data Exploration

Using the code provided for this assignment we can see that if we run the loop multiple times that we get different outputs from some of the runs. The clusters are represented by different coloured points on the histogram. Above each histogram is the kmeans "tot.withinss" which is the sum of the "Vector of within-cluster sum of squares, one component per cluster" (Datacamp, 2020). We would like this to be as low as possible as the lower the value the more homogeneity there is in the clusters.



It appears from a visual inspection of this inital exploration that actually there are only two different outputs. The R kmeans algorithm sometimes chooses a different colour for the clusters but there appear to only be two results one of which is optimum as it has a lower Total Withinss. As Arthur and Vassilvitskii mention "it is standard practice to run the k-means algorithm multiple times, and then keep only the best clustering found."(Arthur and Vassilvitskii, 2006)

To see if this is a behaviour of the particular implementation of KMeans in R a test with each of the algorithms provided with R Kmeans was carried out. This did not show any different results. The optimum result was chosen most of the time with the other result appearing about 20% of the time. The default algorithm in R Kmeans is Hartigan and Wong (1979) however "Lloyd" and "Forgy" are also available. (Datacamp, 2020)

Check Of Number Of Outputs

To test the assertion that there are only two results for any of the random starting point in this data set we can compare the cluster output centres and see how many variants there are. However Kmeans does not know which cluster is cluster 1 etc so first we need to compare the cluster centres in any order using all_equal ignoring row order. Running the code 1000 times shows that we never get a third set of cluster centres as an output.

```
## [1] "First set of centres is"

##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     5.901613    2.748387     4.393548    1.433871
## 2     5.006000    3.428000     1.462000    0.246000
## 3     6.850000    3.073684     5.742105    2.071053

## [1] "Count of first set of centres is:"

## [1] 806

## [1] "Second set of centres is"

##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     4.738095    2.904762     1.790476   0.3523810
## 2     5.175758    3.624242     1.472727   0.2727273
## 3     6.314583    2.895833     4.973958   1.7031250

## [1] "Count of second set of centres is:"

## [1] 194

## [1] "Count of third set of centres is:"

## [1] 0
```

This confirms that regardless of the random starting points chosen by the algorithm we have just two possible outputs. A fairly well known issue with the kmeans algorithm in general is that depending on the starting points chosen the algorithm "is liable to find a local minimum solution instead of a global one, and as such may not find the optimal partition." (Morissette and Chartier, 2013). We can see that in this case that the Kmeans algorithm is choosing a non optimal solution just under 20% of the time.

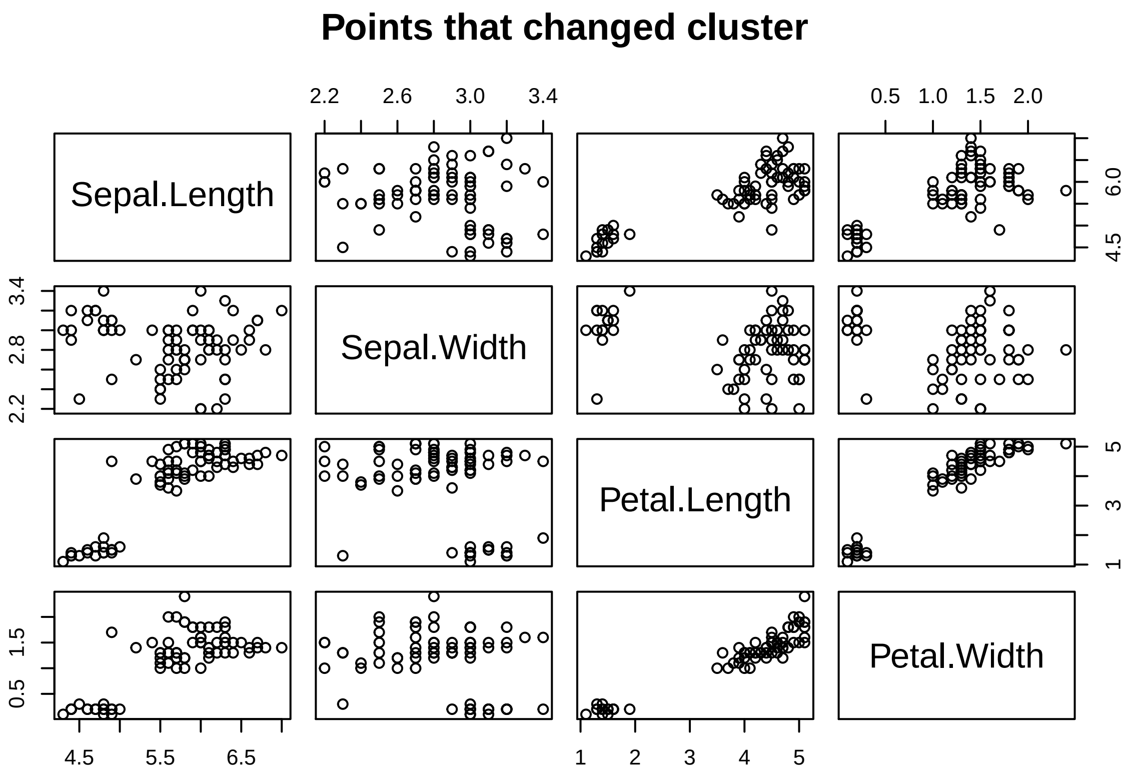
Check Behaviour Of Individual Data Points

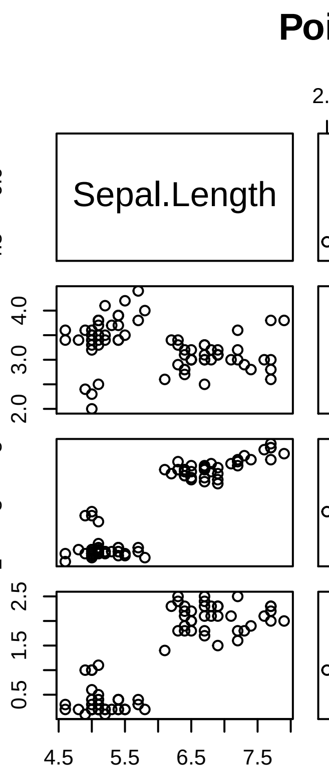
To observe the behaviour of the data points that potentially end up in different clusters for the two possible outputs we can compare and plot the output vectors. There is a problem with this in that the cluster numbers in multiple runs of KMeans can change. Cluster 1 is not always cluster 1. Looking back at the plots in the first section we can see that the 3 clusters could be described as Top Left, Bottom Left and Right in each output.

So the first task is to enure we have two kmeans cluster vectors where the cluster numbers are consistent across the two possible outputs. The point with the max sepal width should be in the top left cluster. The point with the max sepal length should be in the right cluster. So we will run the outputs until we match two cluster vectors that agree. We will arbitrarily say we want the top left cluster to be cluster 1 and the right cluster to be cluster 3.

```
## [1] "The Max Sepal Width Is In Row:"

## [1] 16

## [1] "The Max Sepal Length Is In Row:"

## [1] 132
```

We can now compare and plot the points that have moved (and secondly the ones that didn't move) between the two solutions with the pairs function that plots each of the four dimensions against the other:





We can see fairly readily from this visualisation that the points that didn't change cluster are fairly obviously clustered in each combination of plots. The points that move are obviously clustered in some dimensions but not others.

Code Appendix

```
knitr::opts_chunk$set(echo = TRUE)
iris
newiris <- iris
newiris$Species <- NULL
par(mfrow = c(3,4))
for(i in 1:12) {
kc <- kmeans(newiris, 3)
```

```r
plot(newiris[c("Sepal.Length", "Sepal.Width")], col=kc$cluster, main = kc$tot.withinss)
}
suppressWarnings(suppressMessages(library(tidyverse)))
newiris <- iris
newiris$Species <- NULL
kc <- kmeans(newiris, 3)
res1count <- 0
res2count <- 0
res3count <- 0
result1 <- kc$centers
result2 <- result1[FALSE,]

for (i in 1:1000){
  match = 0
  kc <- kmeans(newiris, 3)
  comparison <- kc$centers
  if ((all_equal(result1, comparison, ignore_col_order = TRUE, ignore_row_order = TRUE, convert = FALSE
    res1count = res1count + 1
    match = 1
  } else {
    res2count = res2count + 1
    result2 <- comparison
    match =1
  }
  if (match == 0){
    res3count = res3count + 1
  }
}

print("First set of centres is")
print(result1)
print("Count of first set of centres is:")
print(res1count)
print("Second set of centres is")
print(result2)
print("Count of second set of centres is:")
print(res2count)
print("Count of third set of centres is:")
print(res3count)

suppressWarnings(suppressMessages(library(tidyverse)))
newiris <- iris
newiris$Species <- NULL
row_max_width <- which.max(newiris$Sepal.Width)
print("The Max Sepal Width Is In Row:")
print(row_max_width)
print("The Max Sepal Length Is In Row:")
row_max_length <- which.max(newiris$Sepal.Length)
print(row_max_length)
kc <- kmeans(newiris, 3)
result1 <- kc$centers
result2 <- result1[FALSE,]
```

```r
for (i in 1:100){
  kc <- kmeans(newiris, 3)
  comparison <- kc$centers
  if ((all_equal(result1, comparison, ignore_col_order = TRUE, ignore_row_order = TRUE, convert = FALSE
    if (kc$cluster[row_max_length] == 3 && kc$cluster[row_max_width] ==1){
      type1compare = kc
    }
  } else {
    if (kc$cluster[row_max_length] == 3 && kc$cluster[row_max_width] ==1){
      type2compare = kc
    }
  }
}
#print(type1compare)
#print(type2compare)


resultvector = type1compare$cluster != type2compare$cluster
moved = newiris[resultvector,]
pairs(moved, main = "Points that changed cluster")




notmoved = newiris[!resultvector,]
pairs(notmoved, main = "Points that didn't change cluster" )
```

References

Arthur, D. and Vassilvitskii, S. (2006) *K-means++: The advantages of careful seeding*. Stanford.

Datacamp (2020) *RDocumentation kmeans*. Available at: https://www.rdocumentation.org/packages/stats/versions/3.6.2/topics/kmeans.

Morissette, L. and Chartier, S. (2013) 'The k-means clustering technique: General considerations and implementation in mathematica', *Tutorials in Quantitative Methods for Psychology*. Citeseer, 9(1), pp. 15–24.