**Clustering R-implementations**

**Partitioning Clustering (K-means)**

**K-means**

The k-means algorithm is very simple and basically consists of two steps. It is initialized by a random choice of cluster centres, e.g. a random selection of objects in the data set or random values within the range for each variable. The steps are:

1. Pick an initial set of K centroids (this can be random or any other means)
2. For each data point, assign it to the member of the closest centroid according to the given distance function
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.
4. Output the centroids.

In this part, we want to show how to use K-means in R through few examples (**codes are provided in blue colour**). **Make sure that you have installed the appropriate packages before calling them via the library command.**

**IRIS Dataset (Our first simple k-means code)**

The dataset is the Iris dataset; this dataset contains data on flowers from three different species of Iris (i.e. three classes/groups): setosa, versicolor and virginica. Each observation contains 4 variables, the petal width, petal length, sepal width and sepal length. This dataset is labelled since it contains the species of the flower (i.e. we know the desired output). Let’s see if the unsupervised k-means algorithm can detect the species on its own (i.e. without utilising the desired output knowledge)! Iris data is considered as a classic benchmark dataset.

library(datasets)

We can show the iris data with this command, just type "iris" for show the all data

head(iris)

+/- Output

## Sepal.Length Sepal.Width Petal.Length Petal.Width Species

## 1 5.1 3.5 1.4 0.2 setosa

## 2 4.9 3.0 1.4 0.2 setosa

## 3 4.7 3.2 1.3 0.2 setosa

## 4 4.6 3.1 1.5 0.2 setosa

## 5 5.0 3.6 1.4 0.2 setosa

## 6 5.4 3.9 1.7 0.4 setosa

or we can use command "names" to show the label/column names

names(iris)

+/- Output

## [1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"

## [5] "Species"

In this example, we assign the data from column 1-4 (features) to variable x, and the class column to variable y. Remember, the first 4 input variables are the attributes/features for this specific dataset. The 5th input represents the output class.

x = iris[,-5] # here the x now contains only the first 4 columns, as the 5th has been omitted.

y = iris$Species #the $ sign is associated with the label of that particular column

Create kmeans model with this command: (You need to put the number how many cluster you want, in this case we can use 3 because we already now in iris data we have 3 classes)

kc <- kmeans(x,3)

type "kc" for show summary

kc

+/- Output

## K-means clustering with 3 clusters of sizes 62, 38, 50

##

## Cluster means:

## Sepal.Length Sepal.Width Petal.Length Petal.Width

## 1 5.902 2.748 4.394 1.434

## 2 6.850 3.074 5.742 2.071

## 3 5.006 3.428 1.462 0.246

##

## Clustering vector:

## [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

## [36] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

## [71] 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2

## [106] 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2

## [141] 2 2 1 2 2 2 1 2 2 1

##

## Within cluster sum of squares by cluster:

## [1] 39.82 23.88 15.15

## (between\_SS / total\_SS = 88.4 %)

##

## Available components:

##

## [1] "cluster" "centers" "totss" "withinss"

## [5] "tot.withinss" "betweenss" "size" "iter"

## [9] "ifault"

After we know the result, we need to know how many error and missing data, so we need to compare the clustering result with the species/classes iris data. We use table for comparison

table(y,kc$cluster)

# table command provides us the “famous” confusion matrix which is important in order to extract information regarding the acquired accuracy.

+/- Output

##

## y 1 2 3

## setosa 0 0 50

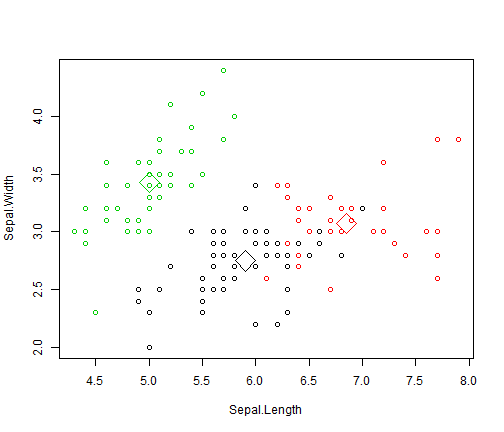
## versicolor 48 2 0

## virginica 14 36 0

For plotting we can use plot function. In this case we plot the Sepal length as x-axis and Sepal Width as y-axis, you may choose different.

plot(x[c("Sepal.Length", "Sepal.Width")], col=kc$cluster)

points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=23, cex=3)



You can see, that although “green” samples are linear separable from the other two classes, this cannot be verified for the other two (“black” & “red”).

The format of the K-means function in R is kmeans(*x*, *centers*) where *x* is a numeric dataset (matrix or data frame) and *centers* is the number of clusters to extract. The function returns the cluster memberships, centroids, sums of squares (within, between, total), and cluster sizes. Since K-means cluster analysis starts with k randomly chosen centroids, a different solution can be obtained each time the function is invoked. Use the set.seed() function to guarantee that the results are reproducible. Additionally, this clustering approach can be sensitive to the initial selection of centroids. The kmeans() function has an nstart option that attempts multiple initial configurations and reports on the best one. For example, adding nstart=25 will generate 25 initial configurations. This approach is often recommended.

**Example 2 (Variation from the previous):**

Since the initial cluster assignments are random, let us set the seed to ensure reproducibility.

library(datasets)

head(iris)

set.seed(20)

irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)

irisCluster

In the above code, rather than taking four variables as inputs, we have selected only two (3rd & 4th). See that before the “,” in the [,3:4] I have nothing. This means I select all the rows!

K-means clustering with 3 clusters of sizes 46, 54, 50

Cluster means:

Petal.Length Petal.Width

1 5.626087 2.047826

2 4.292593 1.359259

3 1.462000 0.246000

Clustering vector:

[1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

[35] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

[69] 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1

[103] 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1

[137] 1 1 2 1 1 1 1 1 1 1 1 1 1 1

Within cluster sum of squares by cluster:

[1] 15.16348 14.22741 2.02200

(between\_SS / total\_SS = 94.3 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size" "iter"

[9] "ifault"

Since we know that there are 3 species involved, we can ask the algorithm to group the data into 3 clusters, and since the starting assignments are random, we specify nstart = 20. This means that R-code will try 20 different random starting assignments and then select the one with the lowest within cluster variation. We can see the cluster centroids, the clusters that each data point was assigned to, and the within cluster variation. Let us compare the clusters with the species.

table(irisCluster$cluster, iris$Species)

setosa versicolor virginica

1 0 2 44

2 0 48 6

3 50 0 0

As we can see, the data belonging to the setosa species got grouped into cluster 3, versicolor into cluster 2, and virginica into cluster 1. The algorithm wrongly classified two data points belonging to versicolor and six data points belonging to virginica.

**IRIS Dataset (k-means code that includes also some tools for “finding” the proper number of clusters)**

**NbClust** is an R Package for determining the relevant number of clusters in a dataset. It provides 30 indices which determine the number of clusters in a data set and it offers also the best clustering scheme from different results to the user. In addition, it provides a function to perform k-means and hierarchical clustering with different distance measures and aggregation methods. Any combination of validation indices and clustering methods can be requested in a single function call. This enables the user to simultaneously evaluate several clustering schemes while varying the number of clusters, to help determining the most appropriate number of clusters for the data set of interest. So, you need to install the related package in you RStudio environment.

install.packages("NbClust")

|  |
| --- |
|  |

You can then load it in memory as

library(NbClust)

To load the Iris dataset in R type at the console

data(iris)

To see the dataset column headings and datatype

str(iris)

head(iris)

We need to remove the column ‘Species’ from this dataset because if we don’t, then we will get an error when executing the NbClust method

iris$Species=NULL

Now apply the NbClust method as given but first set the seed function to any value so that your result is reproducible.

set.seed(26)

clusterNo=NbClust(iris,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",index="all")

“index=all” means test for all 30 indices with the given parameters. You will then get a brief summary of the results as shown below

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Among all indices:

\* 2 proposed 2 as the best number of clusters

\* 13 proposed 3 as the best number of clusters

\* 5 proposed 4 as the best number of clusters

\* 1 proposed 6 as the best number of clusters

\* 2 proposed 10 as the best number of clusters

\*\*\*\*\* Conclusion \*\*\*\*\*

\*According to the majority rule, the best number of clusters is 3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Try also

clusterNo=NbClust(iris,distance="manhattan", min.nc=2,max.nc=15,method="kmeans",index="all")

and / or

clusterNo=NbClust(iris,distance="maximum", min.nc=2,max.nc=15,method="kmeans",index="all")

**Elbow Method**

We can also plot the Within Cluster Sum of Squares and the number of clusters to find the location of a bend or a knee in the plot which is considered as an indicator of the appropriate number of clusters.

First, calculate the within sum of squares (withinss) of different numbers of clusters:

k = 2:10

set.seed(42)

WSS = sapply(k, function(k) {kmeans(iris[1:4], centers=k)$tot.withinss})

You can then use a line plot to plot the within sum of squares with a different number of k

plot(k, WSS, type="l", xlab= "Number of k", ylab="Within sum of squares")

If you wish to have consistency with the variable names, and replicate example 1, here is the equivalent code:

library(NbClust)

library(datasets)

head(iris)

names(iris)

x = iris[,-5]

y = iris$Species

#set.seed(26)

clusterNo=NbClust(x,distance="euclidean", min.nc=2,max.nc=15,method="kmeans",index="all")

#clusterNo=NbClust(iris,distance="manhattan", min.nc=2,max.nc=15,method="kmeans",index="all")

#clusterNo=NbClust(iris,distance="maximum", min.nc=2,max.nc=15,method="kmeans",index="all")

k = 2:10

#set.seed(42)

WSS = sapply(k, function(k) {kmeans(x, centers=k)$tot.withinss})

plot(k, WSS, type="l", xlab= "Number of k", ylab="Within sum of squares")