**K-Means Clustering in R**

Learn all about clustering and, more specifically, k-means in this R Tutorial, where you'll focus on a case study with Uber data.

Clustering is an unsupervised learning technique. It is the task of grouping together a set of objects in a way that objects in the same cluster are more similar to each other than to objects in other clusters. Similarity is an amount that reflects the strength of relationship between two data objects. Clustering is mainly used for exploratory data mining. It is used in many fields such as machine learning, pattern recognition, image analysis, information retrieval, bio-informatics, data compression, and computer graphics.

In this tutorial, you will see:

* You'll first take a look at the different [types of clustering](https://www.datacamp.com/community/tutorials/k-means-clustering-r#types): hard and soft clustering
* Next, you'll study the [types of clustering methods](https://www.datacamp.com/community/tutorials/k-means-clustering-r#types_algo), such as connectivity-, centroid-, distribution- and density-based clustring.
* You will then learn about the [k-means clustering](https://www.datacamp.com/community/tutorials/k-means-clustering-r#kmeans) algorithm, an example of centroid-based clustering. You will work on a case study to see the working of k-means on the Uber dataset using R. The dataset is freely available and contains raw data on Uber pickups with information such as the date, time of the trip along with the longitude-latitude information. You can apply clustering on this dataset to identify the different boroughs within New York.

Make sure to check out DataCamp's [Unsupervised Learning in R](https://www.datacamp.com/courses/unsupervised-learning-in-r) course. The course dives into the concepts of unsupervised learning using R. You will see the k-means and hierarchical clustering in depth. You will also learn about Principal Component Analysis (PCA), a common approach to dimensionality reduction in Machine Learning.

**Clustering: Types**

Clustering can be broadly divided into two subgroups:

* Hard clustering: in hard clustering, each data object or point either belongs to a cluster completely or not. For example in the Uber dataset, each location belongs to either one borough or the other.
* Soft clustering: in soft clustering, a data point can belong to more than one cluster with some probability or likelihood value. For example, you could identify some locations as the border points belonging to two or more boroughs.

**Clustering Algorithms**

Clustering algorithms can be categorized based on their cluster model, that is based on how they form clusters or groups. This tutorial only highlights some of the prominent clustering algorithms.

* Connectivity-based clustering: the main idea behind this clustering is that data points that are closer in the data space are more related (similar) than to data points farther away. The clusters are formed by connecting data points according to their distance. At different distances, different clusters will form and can be represented using a dendrogram, which gives away why they are also commonly called "hierarchical clustering". These methods do not produce a unique partitioning of the dataset, rather a hierarchy from which the user still needs to choose appropriate clusters by choosing the level where they want to cluster. They are also not very robust towards outliers, which might show up as additional clusters or even cause other clusters to merge.
* Centroid-based clustering: in this type of clustering, clusters are represented by a central vector or a centroid. This centroid might not necessarily be a member of the dataset. This is an iterative clustering algorithms in which the notion of similarity is derived by how close a data point is to the centroid of the cluster. k-means is a centroid based clustering, and will you see this topic more in detail later on in the tutorial.
* Distribution-based clustering: this clustering is very closely related to statistics: distributional modeling. Clustering is based on the notion of how probable is it for a data point to belong to a certain distribution, such as the Gaussian distribution, for example. Data points in a cluster belong to the same distribution. These models have a strong theoritical foundation, however they often suffer from overfitting. Gaussian mixture models, using the expectation-maximization algorithm is a famous distribution based clustering method.
* Density-based methods search the data space for areas of varied density of data points. Clusters are defined as areas of higher density within the data space compared to other regions. Data points in the sparse areas are usually considered to be noise and/or border points. The drawback with these methods is that they expect some kind of density guide or parameters to detect cluster borders. DBSCAN and OPTICS are some prominent density based clustering.

**One algorithm to rule them all**

Now that you have seen various types of clustering algorithms, the big question is: "how can you identify the correct algorithm to use?"

Well, sorry but there is no ONE algorithm to rule them all. False alarm!!

"Clustering is in the eye of the beholder!"

Clustering is an subjective task and there can be more than one correct clustering algorithm. Every algorithm follows a different set of rules for defining the 'similarity' among data points. The most appropriate clustering algorithm for a particular problem often needs to be chosen experimentally, unless there is a mathematical reason to prefer one clustering algorithm over another. An algorithm might work well on a particular dataset but fail for a different kind of dataset.

**K-Means Clustering**

In this section, you will work with the Uber dataset, which contains data generated by Uber for the city on New York. Uber Technologies Inc. is a peer-to-peer ride sharing platform. Don't worry if you don't know too much about Uber, all you need to know is that the Uber platform connects you with (cab)drivers who can drive you to your destiny. The data is freely available on [Kaggle](https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city/data). The dataset contains raw data on Uber pickups with information such as the date, time of the trip along with the longitude-latitude information.

New York city has five boroughs: Brooklyn, Queens, Manhattan, Bronx, and Staten Island. At the end of this mini-project, you will apply k-means clustering on the dataset to explore the dataset better and identify the different boroughs within New York. All along, you will also learn the various steps that you should take when working on a data science project in general.

**Problem Understanding**

There is a lot of information stored in the traffic flow of any city. This data when mined over location can provide information about the major attractions of the city, it can help us understand the various zones of the city such as residential areas, office/school zones, highways, etc. This can help governments and other institutes plan the city better and enforce suitable rules and regulations accordingly. For example, a different speed limit in school and residential zone than compared to highway zones.

The data when monitored over time can help us identify rush hours, holiday season, impact of weather, etc. This knowledge can be applied for better planning and traffic management. This can at a large, impact the efficiency of the city and can also help avoid disasters, or at least faster redirection of traffic flow after accidents.

However, this is all looking at the bigger problem. This tutorial will only concentrate on trying to solve the problem of identifying the five boroughs of New York city using k-means algorithm, so as to get a better understanding of the algorithms, all along learning to tackle a data science problem.

**Understanding The Data**

You only need to use the Uber data from 2014. You will find the following .csv files in the Kaggle link mentioned above:

* uber-raw-data-apr14.csv
* uber-raw-data-may14.csv
* uber-raw-data-jun14.csv
* uber-raw-data-jul14.csv
* uber-raw-data-aug14.csv
* uber-raw-data-sep14.csv

This tutorial makes use of various libraries. Remember that when you work locally, you might have to install them. You can easily do so, using install.packages().

Let's now load up the data:

# Load the .csv files

apr14 <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/uber-tlc-foil-response/master/uber-trip-data/uber-raw-data-apr14.csv")

may14 <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/uber-tlc-foil-response/master/uber-trip-data/uber-raw-data-may14.csv")

jun14 <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/uber-tlc-foil-response/master/uber-trip-data/uber-raw-data-jun14.csv")

jul14 <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/uber-tlc-foil-response/master/uber-trip-data/uber-raw-data-jul14.csv")

aug14 <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/uber-tlc-foil-response/master/uber-trip-data/uber-raw-data-aug14.csv")

sep14 <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/uber-tlc-foil-response/master/uber-trip-data/uber-raw-data-sep14.csv")

Let's bind all the data files into one. For this, you can use the bind\_rows() function under the dplyr library in R.

library(dplyr)

data14 <- bind\_rows(apr14, may14, jun14, jul14, aug14, sep14)

So far, so good! Let's get a summary of the data to get an idea of what you are dealing with.

summary(data14)

Date.Time Lat Lon Base

Length:4534327 Min. :39.66 Min. :-74.93 B02512: 205673

Class :character 1st Qu.:40.72 1st Qu.:-74.00 B02598:1393113

Mode :character Median :40.74 Median :-73.98 B02617:1458853

Mean :40.74 Mean :-73.97 B02682:1212789

3rd Qu.:40.76 3rd Qu.:-73.97 B02764: 263899

Max. :42.12 Max. :-72.07

The dataset contains the following columns:

* Date.Time : the date and time of the Uber pickup;
* Lat: the latitude of the Uber pickup;
* Lon: the longitude of the Uber pickup;
* Base: the TLC base company code affiliated with the Uber pickup.

**Data Preparation**

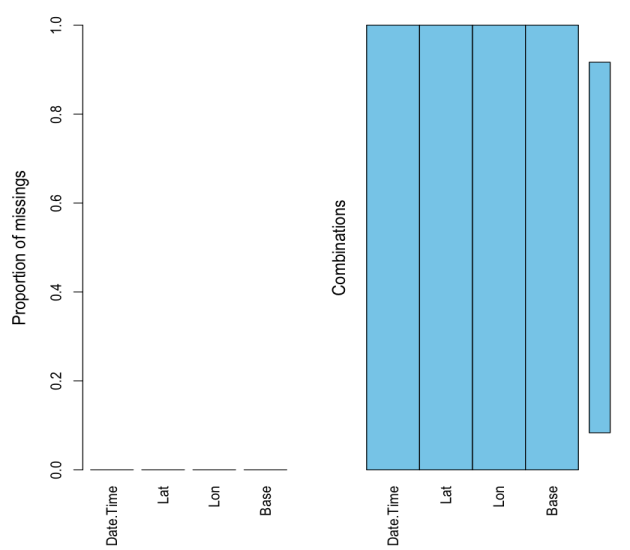
This step consists of cleaning and rearranging your data so that you can work on it more easily. It's a good idea to first think of the sparsity of the dataset and check the amount of missing data.

# VIM library for using 'aggr'

library(VIM)

# 'aggr' plots the amount of missing/imputed values in each column

aggr(data14)



As you can see, the dataset has no missing values. However, this might not always be the case with real datasets and you will have to decide how you want to deal with these values. Some popular methods include either deleting the particular row/column or replacing with a mean of the value.

You can see that the first column is Date.Time. To be able to use these values, you need to separate them. So let's do that, you can use the lubridate library for this. Lubridate makes it simple for you to identify the order in which the year, month, and day appears in your dates and manipulate them.

library(lubridate)

# Separate or mutate the Date/Time columns

data14$Date.Time <- mdy\_hms(data14$Date.Time)

data14$Year <- factor(year(data14$Date.Time))

data14$Month <- factor(month(data14$Date.Time))

data14$Day <- factor(day(data14$Date.Time))

data14$Weekday <- factor(wday(data14$Date.Time))

data14$Hour <- factor(hour(data14$Date.Time))

data14$Minute <- factor(minute(data14$Date.Time))

data14$Second <- factor(second(data14$Date.Time))

#data14$date\_time

data14$Month

Let's check out the first few rows to see what our data looks like now....

head(data14, n=10)

| **Date.Time** | **Lat** | **Lon** | **Base** | **Year** | **Month** | **Day** | **Weekday** | **Hour** | **Minute** | **Second** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2014-04-01 00:11:00 | 40.7690 | -73.9549 | B02512 | 2014 | 4 | 1 | 3 | 0 | 11 | 0 |
| 2014-04-01 00:17:00 | 40.7267 | -74.0345 | B02512 | 2014 | 4 | 1 | 3 | 0 | 17 | 0 |
| 2014-04-01 00:21:00 | 40.7316 | -73.9873 | B02512 | 2014 | 4 | 1 | 3 | 0 | 21 | 0 |
| 2014-04-01 00:28:00 | 40.7588 | -73.9776 | B02512 | 2014 | 4 | 1 | 3 | 0 | 28 | 0 |
| 2014-04-01 00:33:00 | 40.7594 | -73.9722 | B02512 | 2014 | 4 | 1 | 3 | 0 | 33 | 0 |
| 2014-04-01 00:33:00 | 40.7383 | -74.0403 | B02512 | 2014 | 4 | 1 | 3 | 0 | 33 | 0 |
| 2014-04-01 00:39:00 | 40.7223 | -73.9887 | B02512 | 2014 | 4 | 1 | 3 | 0 | 39 | 0 |
| 2014-04-01 00:45:00 | 40.7620 | -73.9790 | B02512 | 2014 | 4 | 1 | 3 | 0 | 45 | 0 |
| 2014-04-01 00:55:00 | 40.7524 | -73.9960 | B02512 | 2014 | 4 | 1 | 3 | 0 | 55 | 0 |
| 2014-04-01 01:01:00 | 40.7575 | -73.9846 | B02512 | 2014 | 4 | 1 | 3 | 1 | 1 | 0 |

Awesome!

For this case study, this is the only data manipulation you will require for a good data understanding as well as to work with k-means clustering.

Now would be a good time to divide your data into training and test set. This is an important step in every data science project, it is done to train the model on the training set, determine the values of the parameters required and to finally test the model on the testing set. For example, when working with clustering algorithms, this division is done so that you can identify the parameters such as k, which is the number of clusters in k-means clustering. However, for this case study, you already know the number of clusters expected, which is 5 - the number of boroughs in NYC. Hence, you shall not be working the traditional way but rather, keep it primarily about learning about k-means clustering.

Have a look at DataCamp's [Python Machine Learning: Scikit-Learn Tutorial](https://www.datacamp.com/community/tutorials/machine-learning-python) for a project that guides you through all the steps for a data science (machine learning) project using Python. You will also work with k-means algorithm in this tutorial.

Now before diving into the R code for the same, let's learn about the k-means clustering algorithm...

**K-Means Clustering with R**

K-means clustering is the most commonly used unsupervised machine learning algorithm for dividing a given dataset into k clusters. Here, k represents the number of clusters and must be provided by the user. You already know k in case of the Uber dataset, which is 5 or the number of boroughs. k-means is a good algorithm choice for the Uber 2014 dataset since you do not know the target labels making the problem unsupervised and there is a pre-specified k value.

Here you are using clustering for classifying the pickup points into various boroughs. The general scenario where you would use clustering is when you want to learn more about your dataset. So you can run clustering several times, investigate the interesting clusters and note down some of the insights you get. Clustering is more of a tool to help you explore a dataset, and should not always be used as an automatic method to classify data. Hence, you may not always deploy a clustering algorithm for real-world production scenario. They are often too unreliable, and a single clustering alone will not be able to give you all the information you can extract from a dataset.

The basic idea behind k-means clustering consists of defining clusters so that the total intra-cluster variation (known as total within-cluster variation) is minimized. There are several k-means algorithms available. However, the standard algorithm defines the total within-cluster variation as the sum of squared distances Euclidean distances between items and the corresponding centroid:

W(Ck)=∑xi∈Ck(xi−μk)2W(Ck)=∑xi∈Ck(xi−μk)2

where:

* xixi is a data point belonging to the cluster CkCk
* μkμk is the mean value of the points assigned to the cluster CkCk

Each observation xixi is assigned to a given cluster such that the sum of squares distance of the observation to their assigned cluster centers μkμk is minimized.

Let's go through the steps more systematically:

1. Specify k - the number of clusters to be created.
2. Select randomly k objects from the dataset as the initial cluster centers.
3. Assign each observation to their closest centroid, based on the Euclidean distance between the object and the centroid.
4. For each of the k clusters recompute the cluster centroid by calculating the new mean value of all the data points in the cluster.
5. Iteratively minimize the total within sum of square. Repeat Step 3 and Step 4, until the centroids do not change or the maximum number of iterations is reached (R uses 10 as the default value for the maximum number of iterations).

The total within sum of square or the total within-cluster variation is defined as:

∑kk=1W(Ck)=∑kk=1∑xi∈Ck(xi−μk)2∑k=1kW(Ck)=∑k=1k∑xi∈Ck(xi−μk)2

This is the summation of all the clusters over the sum of squared Euclidean distances between items and their corresponding centroid.

Now that you have seen the theory, let's implement the algorithm and see the results!

You can use the kmeans() function in R. k value will be set as 5. Also, there is a nstart option that attempts multiple initial configurations and reports on the best one within the kmeans function. Seeds allow you to create a starting point for randomly generated numbers, so that each time your code is run, the same answer is generated.

set.seed(20)

clusters <- kmeans(data14[,2:3], 5)

# Save the cluster number in the dataset as column 'Borough'

data14$Borough <- as.factor(clusters$cluster)

# Inspect 'clusters'

str(clusters)

List of 9

$ cluster : int [1:4534327] 3 4 4 3 3 4 4 3 4 3 ...

$ centers : num [1:5, 1:2] 40.7 40.8 40.8 40.7 40.7 ...

..- attr(\*, "dimnames")=List of 2

.. ..$ : chr [1:5] "1" "2" "3" "4" ...

.. ..$ : chr [1:2] "Lat" "Lon"

$ totss : num 22107

$ withinss : num [1:5] 1386 1264 948 2787 1029

$ tot.withinss: num 7414

$ betweenss : num 14692

$ size : int [1:5] 145109 217566 1797598 1802301 571753

$ iter : int 4

$ ifault : int 0

- attr(\*, "class")= chr "kmeans"

The above list is an output of the kmeans() function. Let's see some of the important ones closely:

* cluster: a vector of integers (from 1:k) indicating the cluster to which each point is allocated.
* centers: a matrix of cluster centers.
* withinss: vector of within-cluster sum of squares, one component per cluster.
* tot.withinss: total within-cluster sum of squares. That is, sum(withinss).
* size: the number of points in each cluster.

Let's plot some graphs to visualize the data as well as the results of the k-means clustering well.

library(ggmap)

NYCMap <- get\_map("New York", zoom = 10)

ggmap(NYCMap) + geom\_point(aes(x = Lon[], y = Lat[], colour = as.factor(Borough)),data = data14) +

ggtitle("NYC Boroughs using KMean")

The boroughs (clusters) formed is matched against the real boroughs. The cluster number corresponds to the following boroughs:

1. Bronx
2. Manhattan
3. Brooklyn
4. Staten Island
5. Queens

As you can see, the results are pretty impressive. And now, that you have used k-mean to categorize the pickup point and have additional knowledge added to the dataset. Let's try to do something with this new found knowledge. You can use the borough information to check out Uber's growth within the boroughs for each month. Here's how...

library(DT)

data14$Month <- as.double(data14$Month)

month\_borough\_14 <- count\_(data14, vars = c('Month', 'Borough'), sort = TRUE) %>%

arrange(Month, Borough)

datatable(month\_borough\_14)

Let's get a graphical view of the same...

library(dplyr)

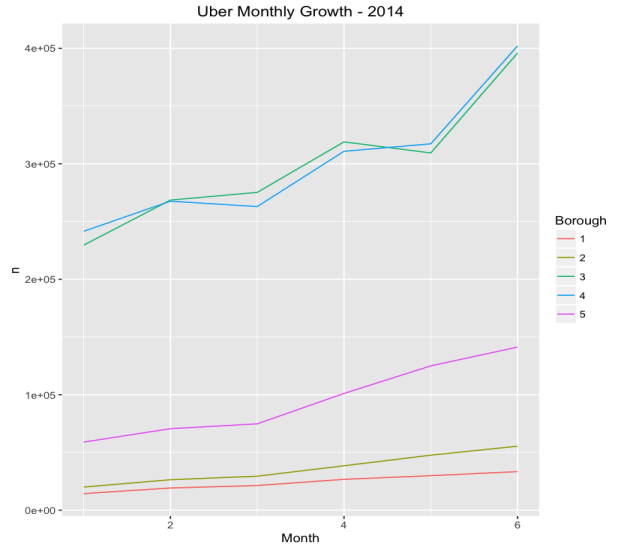
monthly\_growth <- month\_borough\_14 %>%

mutate(Date = paste("04", Month)) %>%

ggplot(aes(Month, n, colour = Borough)) + geom\_line() +

ggtitle("Uber Monthly Growth - 2014")

monthly\_growth



As you have seen, k-means is a pretty good clustering algorithm. But, it has some drawbacks. The biggest disadvantage is that it requires us to pre-specify the number of clusters (k). However, for the Uber dataset, you had some domain knowledge available that told you the number of boroughs in New York City. This might not always be the case with real world datasets. Hierarchical clustering is an alternative approach that does not require a particular choice of clusters. An additional disadvantage of k-means is that it is sensitive to outliers and different results can occur if you change the ordering of the data.

k-means is a lazy learner where generalization of the training data is delayed until a query is made to the system. Which means k-means starts working only when you trigger it to, thus lazy learning methods can construct a different approximation or result to the target function for each encountered query. It is a good method for online learning but it requires a possibly large amount of memory to store the data, and each request involves starting the identification of a local model from scratch.

# K Means Clustering in R Example

Summary: The **kmeans()** function in R requires, at a minimum, numeric data and a number of centers (or clusters).  The cluster centers are pulled out by using **$centers**.  The cluster assignments are pulled by using **$cluster.**  You can evaluate the clusters by looking at **$totss** and **$betweenss**.

**Tutorial Time**: 30 Minutes

R comes with a default K Means function, kmeans().  It only requires two inputs: a matrix or data frame of all numeric values and a number of centers (i.e. your number of clusters or the K of k means).

|  |  |
| --- | --- |
| 1  2  3 | kmeans(x, centers, iter.max = 10, nstart = 1,   algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",   "MacQueen"), trace=FALSE) |

* X is your data frame or matrix.  All values must be numeric.
  + If you have an ID field make sure you drop it or it will be included as part of the centroids.
* Centers is the K of K Means.  centers = 5 would results in 5 clusters being created.
  + You have to determine the appropriate number for K.
* iter.max is the number of times the algorithm will repeat the cluster assignment and moving of centroids.
* nstart is the number of times the initial starting points are re-sampled.
  + In the code, it looks for the initial starting points that have the lowest within sum of squares (withinss).
  + That means it tries “nstart” samples, does the cluster assignment for each data point “nstart” times, and picks the centers that have the lowest distance from the data points to the centroids.
* trace gives a verbose output showing the progress of the algorithm.

## K Means Algorithms in R

The out-of-the-box K Means implementation in R offers three algorithms (Lloyd and Forgy are the same algorithm just named differently).

The default is the Hartigan-Wong algorithm which is often the fastest.  This [StackOverflow answer](http://stackoverflow.com/questions/20446053/k-means-lloyd-forgy-macqueen-hartigan-wong" \t "_blank) is the closest I can find to showing some of the differences between the algorithms.

Research Paper References:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | Forgey, E. (1965). “Cluster Analysis of Multivariate Data:  Efficiency vs. Interpretability of Classification”. In: Biometrics.    Lloyd, S. (1982). “Least Squares Quantization in PCM”.  In: IEEE Trans. Information Theory.    Hartigan, J. A. and M. A. Wong (1979). “Algorithm AS  136: A k-means clustering algorithm”. In: Applied Statistics  28.1, pp. 100–108.    MacQueen, J. B. (1967). “Some Methods for classification  and Analysis of Multivariate Observations”. In: Berkeley  Symposium on Mathematical Statistics and Probability |

## kmeans() R Example

Let’s take an example of clustering customers from a wholesale customer database.  You can download the data I’m using from the Berkley UCI Machine Learning Repository [here](https://archive.ics.uci.edu/ml/datasets/Wholesale+customers).

Let’s start off by reading in the data (Note: You may have to use setwd() to change your directory to wherever you’re storing your data).  After reading in the data, let’s just get a quick summary.

|  |  |
| --- | --- |
| 1  2 | data <-read.csv("Wholesale customers data.csv",header=T)  summary(data) |

There’s obviously a big difference for the top customers in each category (e.g. Fresh goes from a min of 3 to a max of 112,151).  Normalizing / scaling the data won’t necessarily remove those outliers.  Doing a log transformation might help.   We could also remove those customers completely.  From a business perspective, you don’t really need a clustering algorithm to identify what your top customers are buying.  You usually need clustering and segmentation for your middle 50%.

With that being said, let’s try removing the top 5 customers from each category.  We’ll use a custom function and create a new data set called data.rm.top

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | top.n.custs <- function (data,cols,n=5) { #Requires some data frame and the top N to remove  idx.to.remove <-integer(0) #Initialize a vector to hold customers being removed  for (c in cols){ # For every column in the data we passed to this function  col.order <-order(data[,c],decreasing=T) #Sort column "c" in descending order (bigger on top)  #Order returns the sorted index (e.g. row 15, 3, 7, 1, ...) rather than the actual values sorted.  idx <-head(col.order, n) #Take the first n of the sorted column C to  idx.to.remove <-union(idx.to.remove,idx) #Combine and de-duplicate the row ids that need to be removed  }  return(idx.to.remove) #Return the indexes of customers to be removed  }  top.custs <-top.n.custs(data,cols=3:8,n=5)  length(top.custs) #How Many Customers to be Removed?  data[top.custs,] #Examine the customers  data.rm.top<-data[-c(top.custs),] #Remove the Customers |

Now, using data.rm.top, we can perform the cluster analysis.  Important note: We’ll still need to drop the Channel and Region variables.  These are two ID fields and are not useful in clustering.

|  |  |
| --- | --- |
| 1  2  3  4 | set.seed(76964057) #Set the seed for reproducibility  k <-kmeans(data.rm.top[,-c(1,2)], centers=5) #Create 5 clusters, Remove columns 1 and 2  k$centers #Display&nbsp;cluster centers  table(k$cluster) #Give a count of data points in each cluster |

Now we can start interpreting the cluster results:

* Cluster 1 looks to be a heavy Grocery and above average Detergents\_Paper but low Fresh foods.
* Cluster 3 is dominant in the Fresh category.
* Cluster 5 might be either the “junk drawer” catch-all cluster or it might represent the small customers.

A measurement that is more relative would be the withinss and betweenss.

* k$withinss would tell you the sum of the square of the distance from each data point to the cluster center.  Lower is better.  Seeing a high withinss would indicate either outliers are in your data or you need to create more clusters.
* k$betweenss tells you the sum of the squared distance between cluster centers.  Ideally you want cluster centers far apart from each other.

**It’s important to try other values for K**.  You can then compare withinss and betweenss.  This will help you select the best K.   For example, with this data set, what if you ran K from 2 through 20 and plotted the total within sum of squares?  You should find an “elbow” point.  Wherever the graph bends and stops making gains in withinss you call that your K.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | rng<-2:20 #K from 2 to 20  tries <-100 #Run the K Means algorithm 100 times  avg.totw.ss <-integer(length(rng)) #Set up an empty vector to hold all of points  for(v in rng){ # For each value of the range variable   v.totw.ss <-integer(tries) #Set up an empty vector to hold the 100 tries   for(i in 1:tries){   k.temp <-kmeans(data.rm.top,centers=v) #Run kmeans   v.totw.ss[i] <-k.temp$tot.withinss#Store the total withinss   }   avg.totw.ss[v-1] <-mean(v.totw.ss) #Average the 100 total withinss  }  plot(rng,avg.totw.ss,type="b", main="Total Within SS by Various K",   ylab="Average Total Within Sum of Squares",   xlab="Value of K") |

This plot doesn’t show a very strong elbow.  Somewhere around K = 5 we start losing dramatic gains.  So I’m satisfied with 5 clusters.

You now have all of the bare bones for using kmeans clustering in R.

Here’s the full code for this tutorial.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | data <-read.csv("Wholesale customers data.csv",header=T)  summary(data)  top.n.custs <- function (data,cols,n=5) { #Requires some data frame and the top N to remove  idx.to.remove <-integer(0) #Initialize a vector to hold customers being removed  for (c in cols){ # For every column in the data we passed to this function  col.order <-order(data[,c],decreasing=T) #Sort column "c" in descending order (bigger on top)  #Order returns the sorted index (e.g. row 15, 3, 7, 1, ...) rather than the actual values sorted.  idx <-head(col.order, n) #Take the first n of the sorted column C to  idx.to.remove <-union(idx.to.remove,idx) #Combine and de-duplicate the row ids that need to be removed  }  return(idx.to.remove) #Return the indexes of customers to be removed  }  top.custs <-top.n.custs(data,cols=3:8,n=5)  length(top.custs) #How Many Customers to be Removed?  data[top.custs,] #Examine the customers  data.rm.top <-data[-c(top.custs),] #Remove the Customers  set.seed(76964057) #Set the seed for reproducibility  k <-kmeans(data.rm.top[,-c(1,2)], centers=5) #Create 5 clusters, Remove columns 1 and 2  k$centers #Display cluster centers  table(k$cluster) #Give a count of data points in each cluster  rng<-2:20 #K from 2 to 20  tries<-100 #Run the K Means algorithm 100 times  avg.totw.ss<-integer(length(rng)) #Set up an empty vector to hold all of points  for(v in rng){ # For each value of the range variable  v.totw.ss<-integer(tries) #Set up an empty vector to hold the 100 tries  for(i in 1:tries){  k.temp<-kmeans(data.rm.top,centers=v) #Run kmeans  v.totw.ss[i]<-k.temp$tot.withinss#Store the total withinss  }  avg.totw.ss[v-1]<-mean(v.totw.ss) #Average the 100 total withinss  }  plot(rng,avg.totw.ss,type="b", main="Total Within SS by Various K",  ylab="Average Total Within Sum of Squares",  xlab="Value of K") |