post02-changhua-yu

Changhua Yu

12/27/2017

An introduction to basic clustering algorithms and their implementations in R

Intro:

> In the Stat133 lesson, we've learned a powerful unsupervised data processing method, the Principal component analysis, that effectively executes dimensional reduction on complicated data for better visualization and generalization. However, through PC analysis, some of the information contained in the dataset is lost as a trade off for clarity, and what if the information lost is important? Is there a way that does not require the reduction of dimensionality, but yield a clearer visualization for the data? The answer is yes, and there are algorithms that perform grouping of each observation in the dataset based on the similarity of features but without the dimension of features reduced.

> In this post, I will introduce the algorithms and the implementations in R of two prevalent clustering methods: k-means clustering and hierarchical clustering; moreover, I will try to compare their pros and cons in various conditions.

Motivation:

> I was fascinated by the statistical methods that aid the genomic sequencing and characterization of cell sub-populations during my internship at Yale last summer. I ended up completing my first post on the exploration of PCA by applying it to cell data processing. During my research on the previous topic, I've realized that clustering is also an indispensable part of unsupervised learning and bear the advantage over PCA in that no information is lost. Thus I decided to explore more on this topic due to my personal interests as well as its potential to help all the classmates in further projects.

Audience:

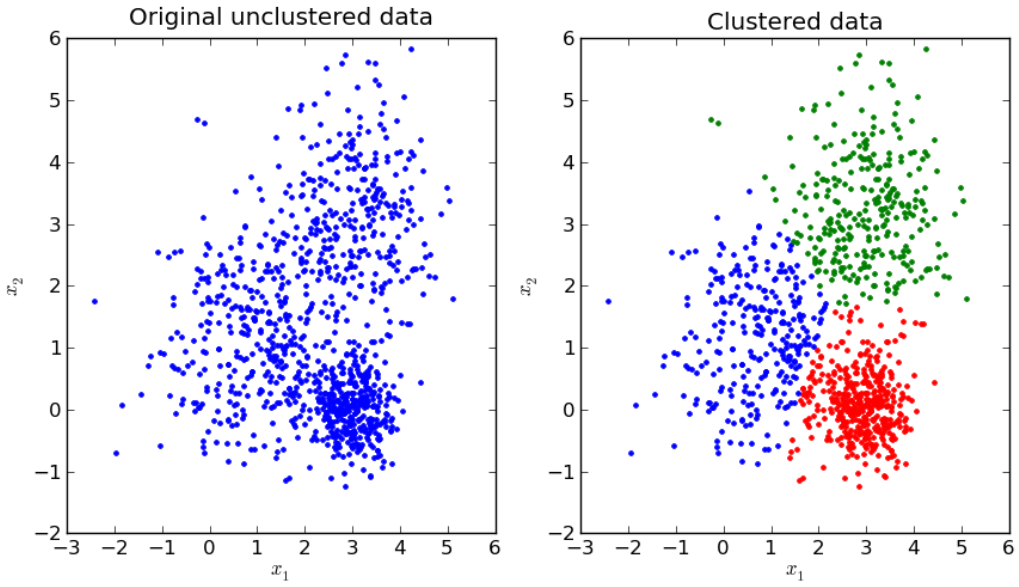
> The post is designed for everyone in the current stat133 course, in that it contains the materials on basic R indexing, string and regular expression manipulation, and user-defined functions, and any command out of the scope of the current content of the course will be fully explained. Moreover, the mathematical algorithms of the two clustering methods will be explained without any prerequisites on linear algebra and advanced calculus.

Introduction to the Methodology and Algorithm

K-means Clustering: Algorithm

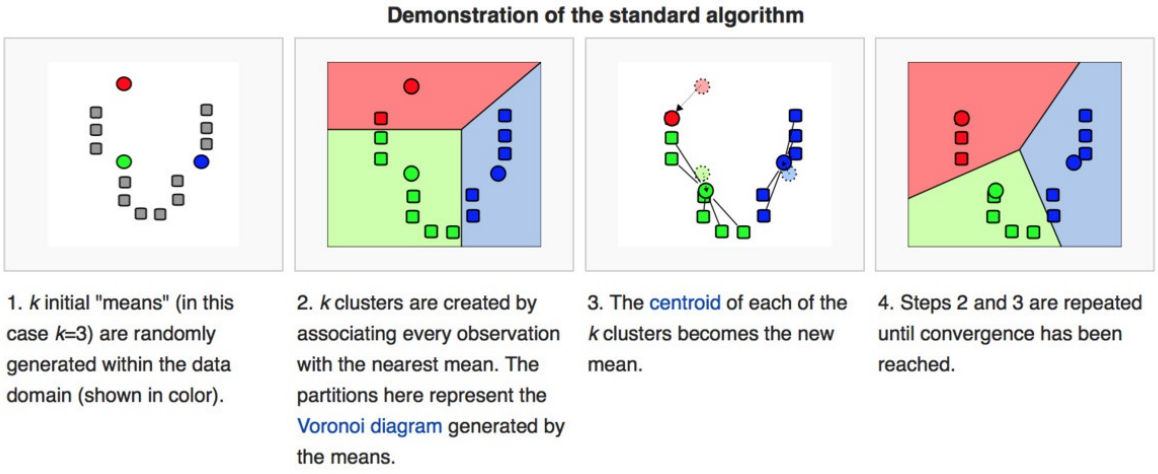
> The K-means Clustering is an iterative and straightforward approach that assigns the data points (observations) in a set to K specific groups based on their similarity, which is commonly measured by the Euclidian distance between the data points. For example, the squared Euclidian distance between two observations P and Q that contain three features would be (p1-q1)^2 + (p2-q2)^2 + (p3-q3).

> The K-means Clustering method is constrained by two principals: 1. first, each observation belongs to at least one of the clusters; 2. second, no observation belongs to more than one group.



Demonstration of K-means Clusterings Principle, source: Arkanath Pathak, Trends Of Code

> We have to find an arrangement that for each of the K groups, the variation within the group is minimized. Provided that the square Euclidean distance is used to measure the variation, we have to minimize the variation between each pair within a group. If the dataset is large and K is also relatively large, the calculation to derive the exactly solution is complicated and computationally unfavorable. However, there's a simple way to approximate a local optimum solution following the procedure provided below: 1. Randomly assign a number of 1-K for each of the observations in the set as the initial assignment
> 2. Calculate the centroid of each group by the means of each feature
> 3. Iterate through process of reassigning each observation to the cluster that has the centroid closest to the observation point, and then recalculate the centroid for each group
> 4. Stop when either not reassignment occurs again or the maximum iteration depth is reached



Demonstration of K-means Clusterings Principle source:https://brilliant.org/wiki/k-means-clustering/

> Through such reallocation of centroid and corresponding groups, a local optimum point can be eventually reached. However, it should be noted that the choice of the initial assignments will affect the approximated local optimum, so we should always run multiple times and then select the most reasonable clustering results.

> So now that we have understand algorithm behind K-means Clustering, I'm going to try to build a user-defined version of K-means Clustering function and test it on a set of simple data.

K-means Clustering: User-built demonstration

```
# now let's try to create a k-means clustering function by ourselves

#first, set a seed to ensure reproducibility
set.seed(2)

#Load the necessary packages
library(dplyr)
```

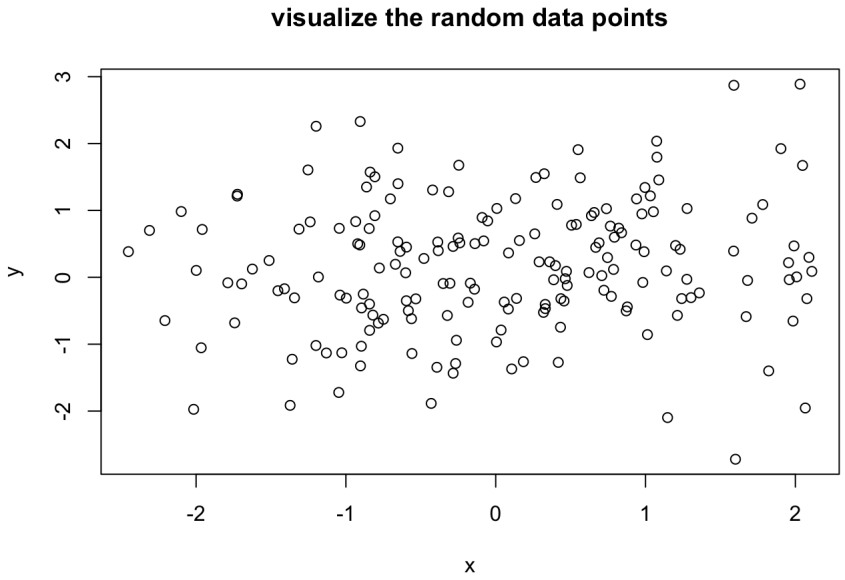
```
## Warning: package 'dplyr' was built under R version 3.4.2
```

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

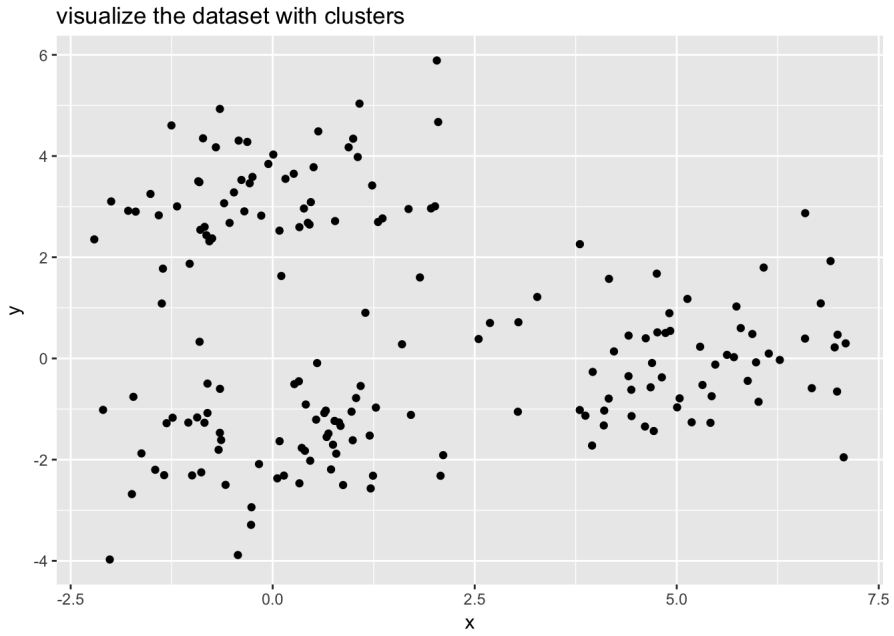
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

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

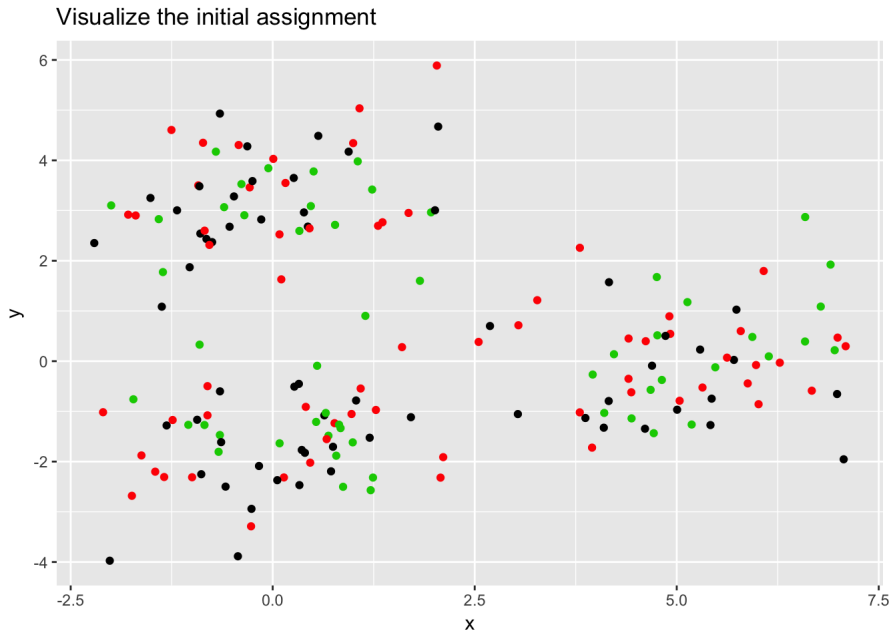
```
library(ggplot2)

#To create a set of data that has apparent groups for testing, we first generate a normally distributed random set with 60 points
dat_without_group = matrix(rnorm(180*2), ncol=2)

#visualize the random data points
plot(dat_without_group,xlab = 'x',ylab = 'y',main = 'visualize the random data points' )
```




```
#then create the clusters by shifting the x or y values for the first, the middle and the last 20 data points
dat_without_group[1:60,1] = dat_without_group[1:60,1]+5
dat_without_group[61:120,2] = dat_without_group[61:120,2]+3
dat_without_group[121:180,2] = dat_without_group[121:180,2]-2

#Visualize the new dataset that clearly contain 3 clusters of points
dat_with_group = data.frame(x = dat_without_group[,1],y = dat_without_group[,2])
ggplot(dat_with_group,aes(x,y)) + geom_point()+ggtitle('visualize the dataset with clusters')
```




```
# Assume that we know it is going to have three clusters, we use K = 3, and then assign n = 1,2,3 randomly for eac
h points
initial_assignment = sample(1:3, size = 180, replace = TRUE)

#Bind the assigned group numbers to each point
dat_with_group = mutate(dat_with_group,group = initial_assignment)

#Visualize the initial assignment
kmeans_1 = ggplot(dat_with_group,aes(x,y)) + geom_point(color = dat_with_group$group) + ggtitle('Visualize the ini
tial assignment')
kmeans_1
```



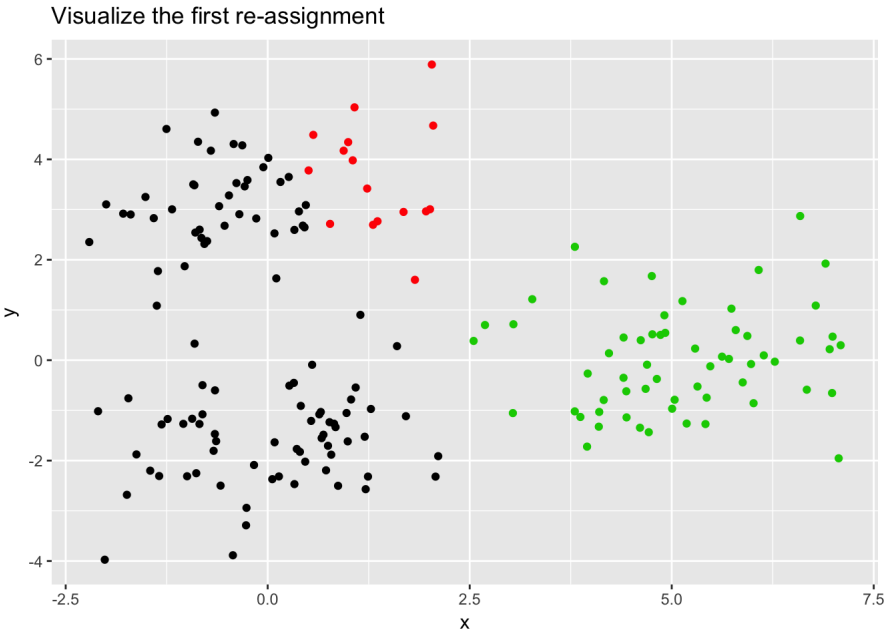

```
# Then, we may calculate the current centroids by calculating the mean of x and y values
centroid_calc = function(dat){
  centriod = list()
  # separate the value of each group
  group1 = filter(dat,dat$group == 1)
  group2 = filter(dat,dat$group == 2)
  group3 = filter(dat,dat$group == 3)

  #Calculate the means for x and y to render the centriod of the three initial groups
  centriod$cent_1 = data.matrix(summarise(group1,x = mean(x),y = mean(y)))
  centriod$cent_2 = data.matrix(summarise(group2,x = mean(x),y = mean(y)))
  centriod$cent_3 = data.matrix(summarise(group3,x = mean(x),y = mean(y)))
  centriod
}
# Display the result of centriods in a list
first_centriods = centroid_calc(dat_with_group)
first_centriods
```

```
## $cent_1
##              x         y
## [1,] 1.251058 0.3647394
##
## $cent_2
##              x         y
## [1,] 1.853167 0.6017789
##
## $cent_3
##              x         y
## [1,] 2.000168 0.5456517
```

```
# Define a function that reassign each point to a new group by minimizing the distance between centriod and each p
oints
reassign = function(centriod, dat){
  # retrieve the coordinates of centriods
  x1 = centriod[[1]][1]
  y1 = centriod[[1]][2]
  x2 = centriod[[2]][1]
  y2 = centriod[[2]][2]
  x3 = centriod[[3]][1]
  y3 = centriod[[3]][2]

  # create a new data frame that contains the distance from each centriods
  dist = data.frame(
  dist_1 = (dat$x - x1)^2+(dat$y - y1)^2,
  dist_2 = (dat$x - x2)^2+(dat$y - y2)^2,
  dist_3 = (dat$x - x3)^2+(dat$y - y3)^2)
  dist
}

# get the distance between each point the three centriods
dist_first = reassign(first_centriods,dat_with_group)
head(dist_first,5)
```

```
##      dist_1    dist_2    dist_3
## 1 10.080486  7.726151  6.906214
## 2 18.120766 14.573512 13.409546
## 3 28.482055 22.461105 21.070339
## 4  9.095438  7.070140  6.307499
## 5 13.491476  9.407240  8.523953
```

```
# reassign the group number for the set
numbering = function(distance, dat){
  # create an empty vector to store the new assignment
  numbers = c(rep(0,60))

  # redeterminine the closest centriod
  for (i in 1:nrow(distance)) {
    current_row = distance[i,]
    numbers[i] = which.min(current_row)
  }
  dat$group = numbers
  dat
}

# Now we get the reassigned groups by iterate one time of K-means-Clustering method
dat_with_group_2 = numbering(dist_first,dat_with_group)

# Let's visualize the clustering results
kmeans_2 = ggplot(dat_with_group_2,aes(x,y)) + geom_point(color = dat_with_group_2$group) + ggtitle('Visualize the
first re-assignment')
kmeans_2
```



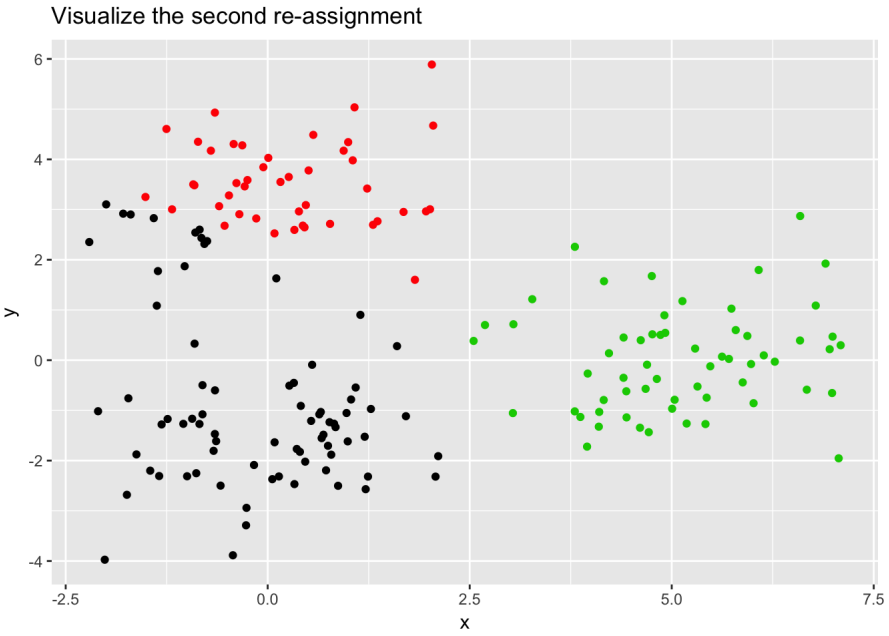

```
# We can see that there's still digressed grouping, so now we iterate another time
second_centriods = centroid_calc(dat_with_group_2)
second_centriods
```

```
## $cent_1
##              x        y
## [1,] -0.215176 0.309284
##
## $cent_2
##             x        y
## [1,] 1.334318 3.655056
##
## $cent_3
##             x           y
## [1,] 5.096811 0.003058197
```

```
dist_second = reassign(second_centriods,dat_with_group_2)
head(dist_second,5)
```

```
##     dist_1   dist_2    dist_3
## 1 20.44214 29.61960 2.0555341
## 2 31.62898 39.00333 1.6079441
## 3 46.28797 38.24587 2.3746024
## 4 18.76127 29.33830 2.7930697
## 5 26.42261 22.53308 0.3241256
```

```
# Now we got the reassigned groups by iterate one time of K-means-Clustering method
dat_with_group_3 = numbering(dist_second,dat_with_group_2)

# Let's visualize the clustering results
kmeans_3 = ggplot(dat_with_group_3,aes(x,y)) + geom_point(color = dat_with_group_3$group) + ggtitle('Visualize the second re-assignment')
kmeans_3
```




```
# We can see that there's still digressed grouping, so now we iterate another time
third_centriods = centroid_calc(dat_with_group_3)
third_centriods
```

```
## $cent_1
##                x          y
## [1,] -0.1730839 -0.8503128
##
## $cent_2
##               x        y
## [1,] 0.2755721 3.528868
##
## $cent_3
##              x           y
## [1,] 5.096811 0.003058197
```

```
dist_third = reassign(third_centriods,dat_with_group_3)
head(dist_third,5)
```

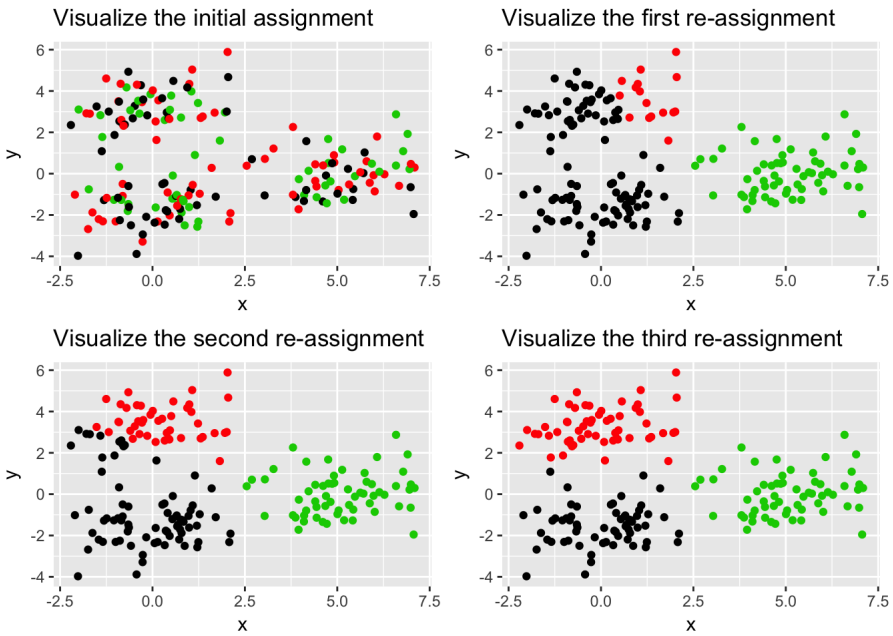
```
##     dist_1   dist_2    dist_3
## 1 18.31806 35.43682 2.0555341
## 2 28.87688 47.05274 1.6079441
## 3 47.25396 49.68358 2.3746024
## 4 16.42252 34.63567 2.7930697
## 5 27.88145 30.47695 0.3241256
```

```
# Now we got the reassigned groups by iterate one time of K-means-Clustering method
dat_with_group_4 = numbering(dist_third,dat_with_group_3)

# Let's visualize the clustering results
kmeans_4 = ggplot(dat_with_group_4,aes(x,y)) + geom_point(color = dat_with_group_4$group) + ggtitle('Visualize the third re-assignment')

#install the grid and gri_extra package for put all for plots altogether
#install.packages('gridExtra') implement this line, but skip this for Rmd knitting issue
library('gridExtra')
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

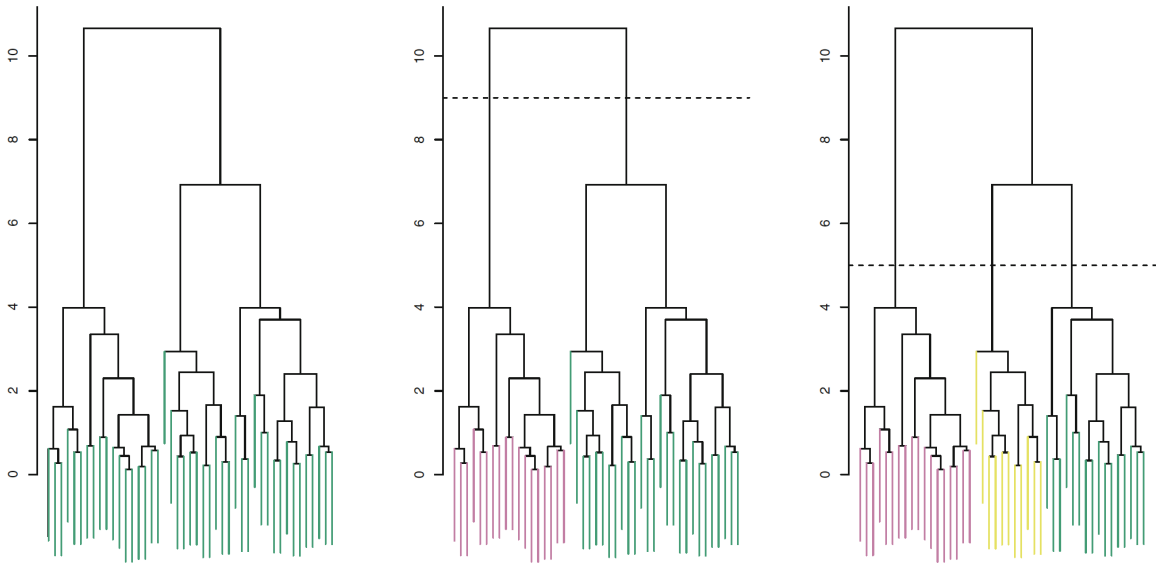
```
grid.arrange(kmeans_1,kmeans_2,kmeans_3,kmeans_4)
```



> We can tell from the grid plot above, that the local optimum is gradually reached as the iterations of the clustering process proceeds, and we will eventually reach the point where no change is needed. The larger the dataset, the greater iteration depth is required.

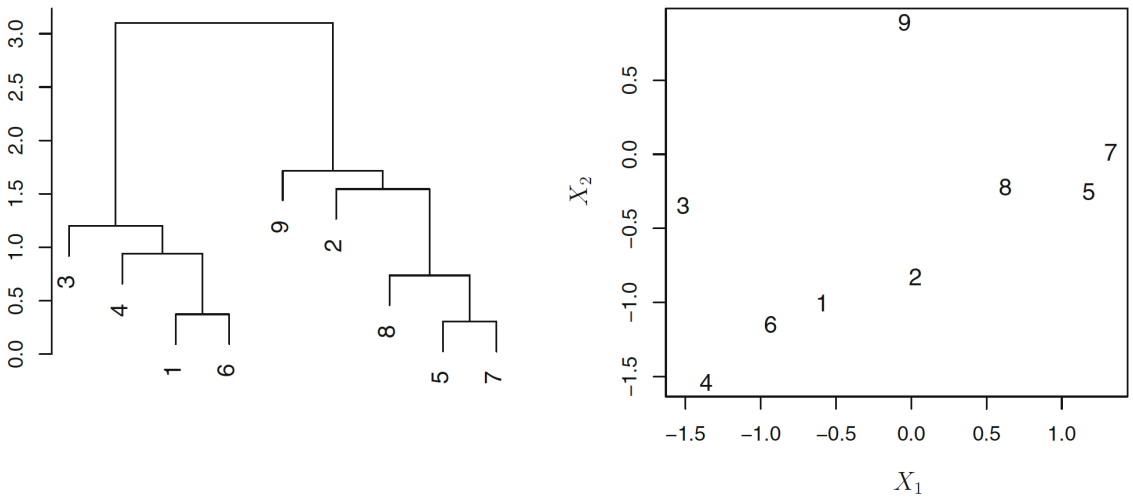
Hierarchical Clustering: Algorithm

> Now we are going to shift our focus to Hierarchical Clustering. The main difference between those two methods is that Hierarchical Clustering does not require a pre-assigned number K as it takes a bottom-up approach to perform the clustering by generating a dendrogram



Demonstration of Hierarchical Clusterings Principle source: The machine learning book

> As the figure depicts, the bottom leafs are the individual observations in the set, and as the branches move up, the leafs that share the most similarity are grouped into single branches. The vertical axis, which is the height of the fusion, thus represents the extent of similarity: the earlier the two leaves are fused together, the more similarity they share.



Demonstration of hierarchical clustering process source: The machine learning book

> The second figure illustrates a simplified version of the hierarchical clustering process for understanding the algorithm. As the dissimilarity is again measured by the Euclidean distance between the two points, the Euclidean distances of every two original observations are derived, and then the two closest two points are fused together into a new branch. Then, iterate through the process by computing the new pairwise inter-cluster dissimilarities among the rest 9 clusters (a branch and 8 points), until all the leafs converge to the largest root.

> The hierarchical clustering is intriguing in that we can obtain any number of clusters by cutting at different height along the vertical axis, and theoretically a cluster at a lower level is always a sub-branch of the parent branch above.

> However, you may ask: how can we calculate the similarity between an individual observation and a branch that contains several points? The answer is that there's no single correct answer, and it depends a lot the type of data you are processing. In the examples below, we will implement the 'complete' method that generate relatively conservative and balanced dendrogram. The 'complete' linkage calculate all the pair-wise Euclidean distances of all the possible combinations of observations between two clusters, and then use the maximal pair-wise dissimilarity as the dissimilarity between those two clusters.

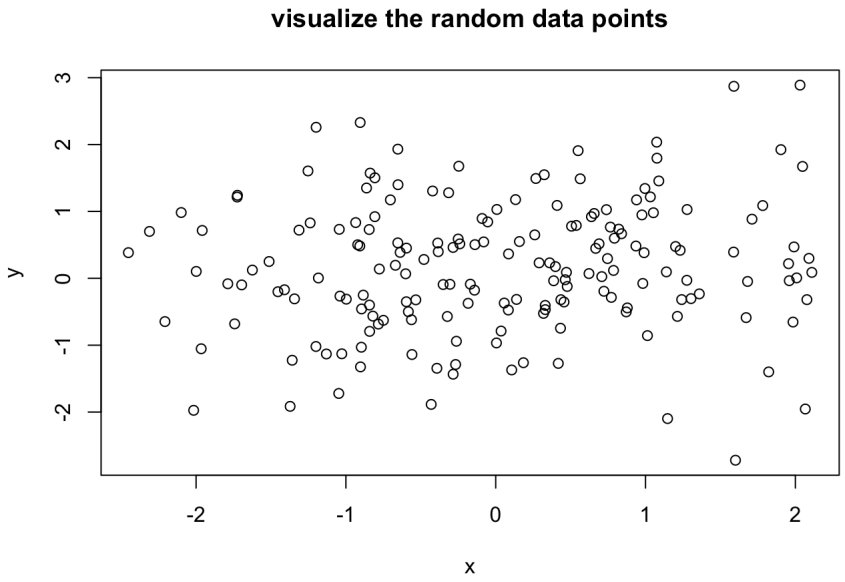
> As the hierarchical clustering relies on the commands that compute pair-wise distances matrix and plot the dendrogram, it is hard to create a user-defined version; rather, the built-in implementation is clear enough to aid our understanding.

R built-in command for clustering methods and their application

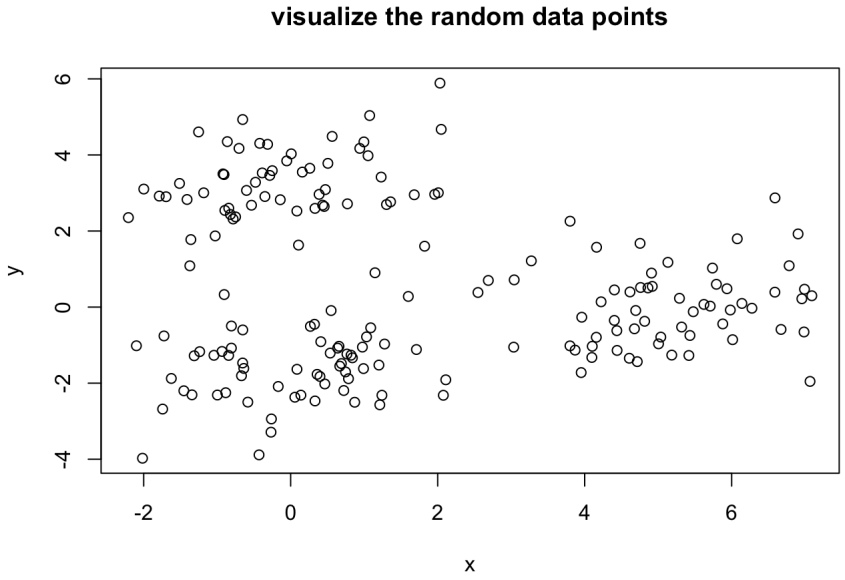
R built-in command for K-means clustering

> In this section, we will use the randomized data with clusters that I created before in the previous section, but apply the kmeans() built in function to achieve the same outputs. The kmeans() function take the following inputs 1. x: a numeric matrix or a data frame with all numeric values that contain the original data 2. centers: The pre-assigned number of group K 3. inter.max: The maximal iteration number had the local optimum not been reached 4. nstart: The number different initial assignements (As we dicussed before, different initial assignments may yield different results, and we should always try multiple time to get the best trial)

```
#To create a set of data that has apparent groups for testing, we first generate the same normally distributed ran
dom set with 60 points
set.seed(2)
dat_without_group = matrix(rnorm(180*2), ncol=2)
#visualize the random data points
plot(dat_without_group,xlab = 'x',ylab = 'y',main = 'visualize the random data points' )
```



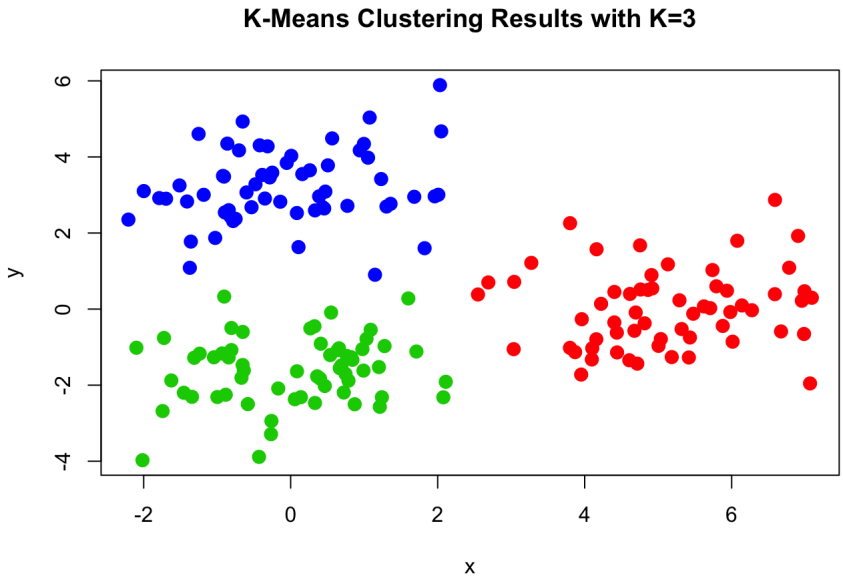

```
#then create the clusters by shifting the x or y values for the first, the middle and the last 20 data points
dat_without_group[1:60,1] = dat_without_group[1:60,1]+5
dat_without_group[61:120,2] = dat_without_group[61:120,2]+3
dat_without_group[121:180,2] = dat_without_group[121:180,2]-2
#visualization of the clustered data
plot(dat_without_group,xlab = 'x',ylab = 'y',main = 'visualize the random data points' )
```



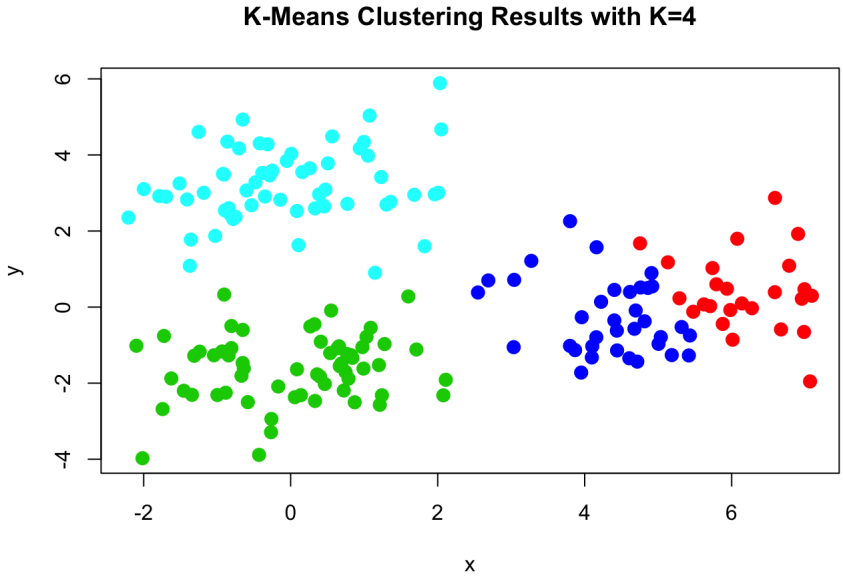

```
# We then implement the kmeans function with the cluster number 3 and 20 different trials with unique initial assi
gnments
k_out=kmeans(dat_without_group,3,nstart=20)

#The cluster assignments of the 150 observations are contained in k_out$cluster
head(k_out$cluster,5)
```

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

```
#Visualize the grouping result
plot(dat_without_group, col=(k_out$cluster+1), main="K-Means Clustering Results with K=3", xlab ="x", ylab="y", pc
h =20, cex =2)
```




```
# Now, suppose that we do not know the exact number of clusters, what if we set K=4?
k_out_four=kmeans(dat_without_group,4,nstart=20)

#Plotting out the result of using K=4 with 20 trials of initial assignments
plot(dat_without_group, col=(k_out_four$cluster+1), main="K-Means Clustering Results with K=4", xlab ="x", ylab="y
", pch =20, cex =2)
```



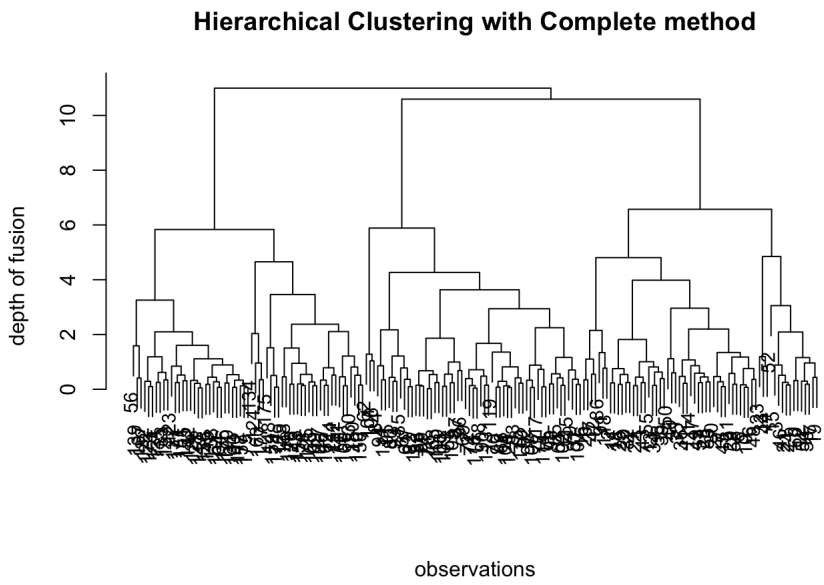
> From the example above, we can tell that the K-mean clustering is not a very robust method, in that slighltly changing the number K will yield very different results.

R built-in command for hierarchical clustering

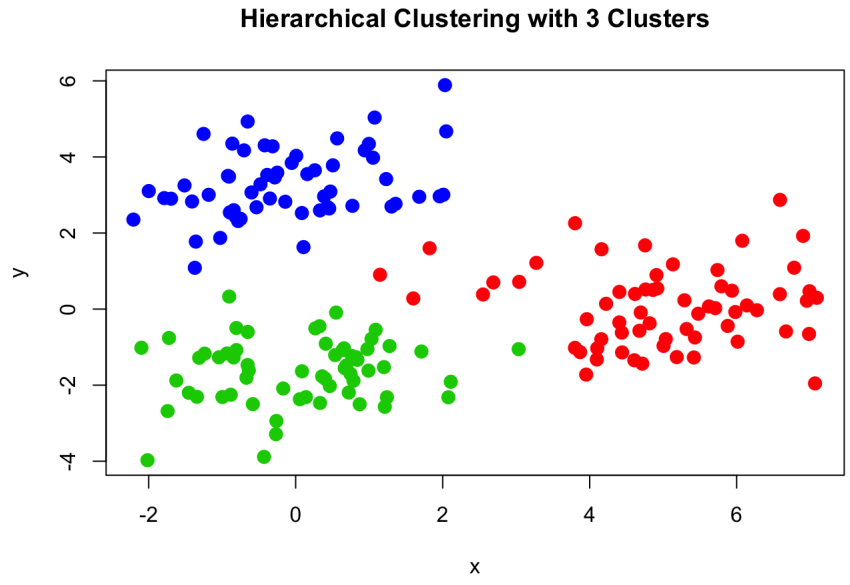
> In this section, we will use the randomized data with clusters that I created before in the previous section, but apply the hclust() and dist() built in function to achieve the hierarchical clustering. The dist() function takes in a numeric matrix, and returns the Euclidean distance matrix between each of the rows, and the hclust() function takes the dissimmilarity matrix created by the dist() and returns a list of outputs for further dendrogram visualization.

```
# So now let's go over the R build in implementation of Hierarchical Clustering

# we will still use the dat_without_group numeric matrix, and first apply the dist function
pair_Distance = dist(dat_without_group)
```

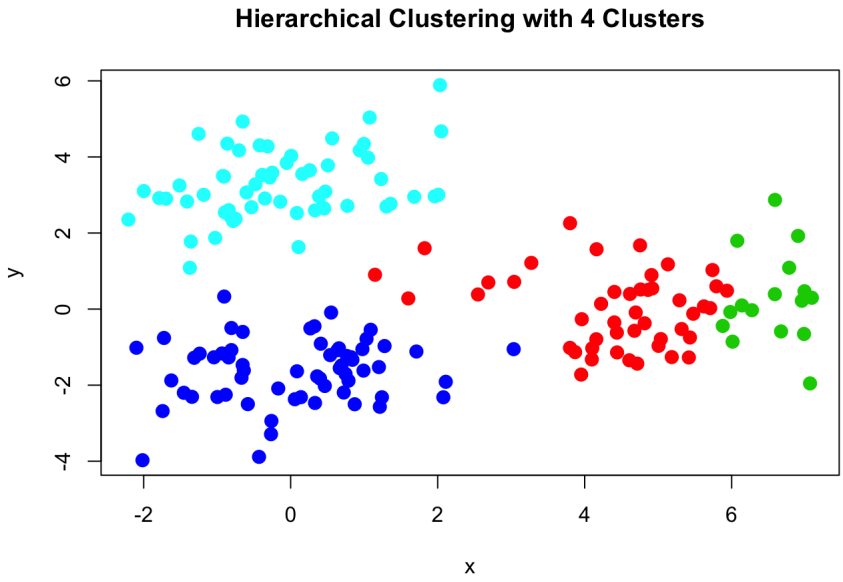
```
# Then, use the pairDistance matrix as the input for hclust(), as described by the introduction, we will apply the
method called 'complete' that use the maximal inter-pair distance as the dissimilarity measurement
hc_complete=hclust(pair_Distance, method="complete")

# Then, we can generate the whole dendrogram with the output hc_complete by simply call plot() function
plot(hc_complete,main="Hierarchical Clustering with Complete method", ylab = 'depth of fusion',xlab="observations"
, sub="",
cex =.9)
```




```
# To get the group labelling number for each of the point at a given cut vertical distance, we apply the function
cutree() that takes hc_complete and a specified number of clusters and outputs the labelling for each of the origi
nal observations
group_labelling = cutree(hc_complete, 3)

# plot the orginal data with the group_labellin as color
plot(dat_without_group, col=(group_labelling+1), main="Hierarchical Clustering with 3 Clusters", xlab ="x", ylab="
y", pch =20, cex =2)
```




```
# we can apply the cutree function again to plot the clustering result with four clusters
group_labelling_4 = cutree(hc_complete, 4)

plot(dat_without_group, col=(group_labelling_4+1), main="Hierarchical Clustering with 4 Clusters", xlab ="x", ylab
="y", pch =20, cex =2)
```

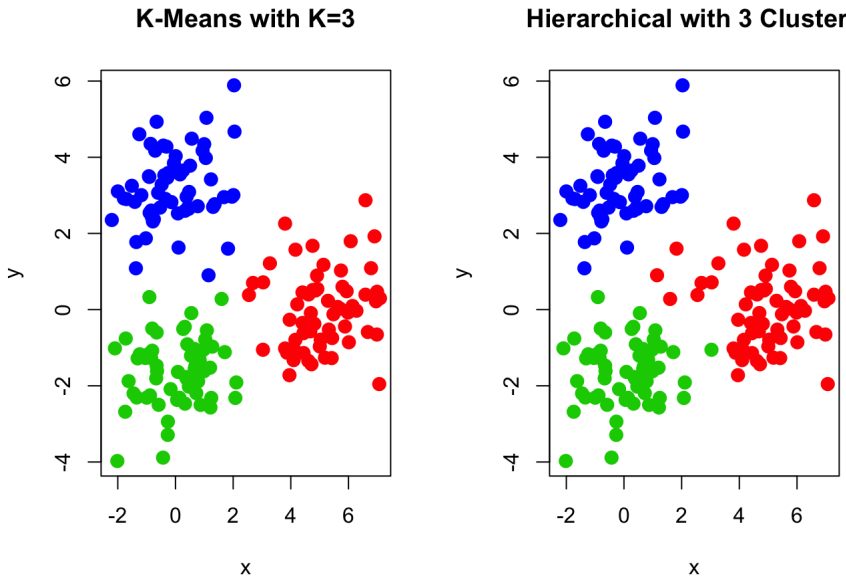


We can see from the above implementation that the Hierarchical Clustering is also not very robust, in that the number of clusters really affect the grouping result; however, most of the grouping arrangement stays the same, in that we can cut the dendrogram from any distances.

Discussion: A Comparison of the two methods

First of all, we can see that the grouping results for our randomly created data by apply different approaches are generally the same for a given number of cluster.

```
# Let's align the results together to aid the comparison

# Align the two graphs in a row
par(mfrow=c(1,2))

#K-means Clustering Plot created before
plot(dat_without_group, col=(k_out$cluster+1), main="K-Means with K=3", xlab ="x", ylab="y", pch =20, cex =2)

#Hierarchical Clustering Plot created before
plot(dat_without_group, col=(group_labelling+1), main="Hierarchical with 3 Clusters", xlab ="x", ylab="y", pch =20
, cex =2)
```



As we've seen in the previous examples, there are pros and cons for each of the algorithms. For the K-means Clustering method, we must decide how many clusters we need beforehand. In contrast, the Hierarchical Clustering method can avoid this problem theoretically, as a dendrogram can be used to generate arbitrary number of clusters by cutting at different levels that are nested. Nevertheless, in reality such assumption might

not true.

For example, if same number of males and females are chosen from three different countries, and the number from each country is also equivalent. If we apply hierarchical clustering to cut the depths with 3 clusters, it should yield the results by grouping people with the same nationalities. If we then apply another round of fusion, it will yield the result of by fusion two countries into a broader branch. However, the most reasonable grouping standard should be gender, and so is deviated from the nested structure. In this case, a K-means Clustering will yield more reasonable result than Hierarchical clustering.

As the clustering algorithms force to fit each of the observation onto a unique cluster, they are not very robust in that a small portion of change in the data may lead to different very different grouping arrangements. Thus, we always need to preprocess the data used to eliminate anomalies that generate noise and to validate the grouping results within a real context

Conclusion

As a conclusion, we have explored two major clustering algorithms: K-means Clustering and Hierarchical Clustering. Specifically, we understand how the algorithms through building up our user-defined functions, apply a randomly created example with clusters to get familiar with the R built-in implementations, and analyze the advantages and disadvantages. Clustering holds the potential in the characterization of functional genes, the evolutionary phylogenetic information, and numerous other machine learning fields. We should always seek new implementations of those methods in interdisciplinary projects, but also should validate and analyze the clustering results in a given context.

References

1. G. James et al., An Introduction to Statistical Learning: with Applications in R, Springer Texts in Statistics , DOI 10.1007/978-1-4614-7138-7

2. Understanding data science: clustering with k-means in R: http://cowlet.org/2013/12/23/understanding-data-science-clustering-with-k-means-in-r.html
3. Rob Kabacoff, R in Action, Manning publishing house.
4. Deza, Elena; Deza, Michel Marie (2009). Encyclopedia of Distances. Springer. p. 94.
5. K-means Clustering Algorithm: Know How It Works:https://www.edureka.co/blog/k-means-clustering-algorithm/
6. k-Means Clustering: https://madlib.apache.org/docs/v1.10/group__grp__kmeans.html
7. Stanford CS221 K means: http://stanford.edu/~cpiech/cs221/handouts/kmeans.html
8. Nathan Landman, Hannah Pang, Eli Ross, k-Means Clustering:https://brilliant.org/wiki/k-means-clustering/
9. Stanford NLP Group:Single-link and complete-link clustering https://nlp.stanford.edu/IR-book/html/htmledition/single-link-and-complete-link-clustering-1.html
10. Data Mining Map - Hierarchical Clustering:http://www.saedsayad.com/clustering_hierarchical.html
11. Arkanath Pathak, Trends Of Code:http://trendsofcode.net/kmeans/