CLUSTERING REPORT

School or Department: Wu Han University

Grade and Specialty: Excellent engineer 2016

Name: Guo Yang, Kong Chuishun, Tang Rui

Advisor: Fangling Pu

April 8 2018

【**Abstract**】 In this lab, we pay attention to the clustering and the K-means clustering analysis. Clustering is the task of grouping a set of objects. It is also a main task of exploratory data mining, and a common technique for statistical data analysis. In the engineering, we use K-means clustering. And we will use a small python lab to turn out.

【**Key words**】 Python; Clustering analysis; K-means clustering; Data mining

Table of Contents

Chapter 1 Clustering analysis

1.1 definition

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics.

1.2 Algorithms

The following overview will only list the most prominent examples of clustering algorithms

1.2.1 Connectivity-based clustering (hierarchical clustering)

Connectivity-based clustering, also known as hierarchical clustering, is based on the core idea of objects being more related to nearby objects than to objects farther away. These algorithms connect "objects" to form "clusters" based on their distance. A cluster can be described largely by the maximum distance needed to connect parts of the cluster. At different distances, different clusters will form, which can be represented using a dendrogram, which explains where the common name "hierarchical clustering" comes from: these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distances.

And the following lab is this kind of clustering analysis

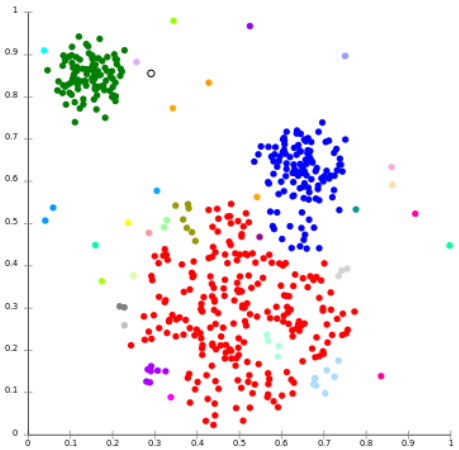


Figure 1.2.1

Single-linkage on Gaussian data. At 35 clusters, the biggest cluster starts fragmenting into smaller parts, while before it was still connected to the second largest due to the single-link effect.

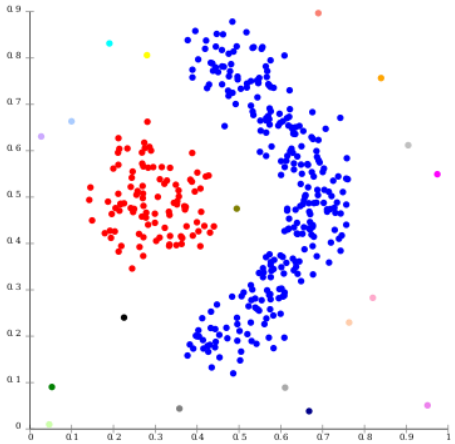


figure 1.2.2

Single-linkage on density-based clusters. 20 clusters extracted, most of which contain single elements, since linkage clustering does not have a notion of "noise".

1.2.2 Distribution-based clustering

The clustering model most closely related to statistics is based on distribution models. Clusters can then easily be defined as objects belonging most likely to the same distribution. A convenient property of this approach is that this closely resembles the way artificial data sets are generated: by sampling random objects from a distribution. One prominent method is known as Gaussian mixture models (using the expectation-maximization algorithm). Here, the data set is usually modeled with a fixed (to avoid overfitting) number of Gaussian distributions that are initialized randomly and whose parameters are iteratively optimized to better fit the data set. This will converge to a local optimum, so multiple runs may produce different results.

Distribution-based clustering produces complex models for clusters that can capture correlation and dependence between attributes.

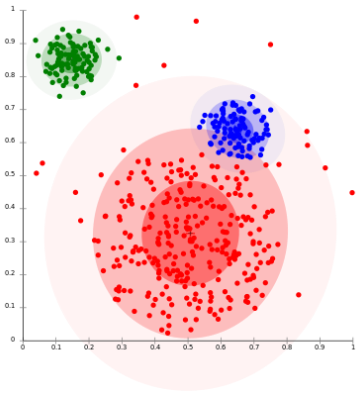


Figure 1.2.3 On Gaussian-distributed data, EM works well, since it uses Gaussians for modelling clusters

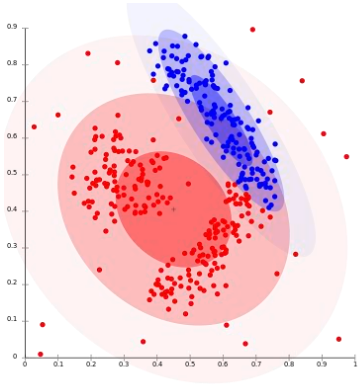


figure 1.2.4 Density-based clusters cannot be modeled using Gaussian distributions

1.2.3 Density-based clustering

In density-based clustering, clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas - that are required to separate clusters - are usually considered to be noise and border points.

The most popular density based clustering method is DBSCAN. In contrast to many newer methods, it features a well-defined cluster model called "density-reachability". Similar to linkage based clustering, it is based on connecting points within certain distance thresholds. However, it only connects points that satisfy a density criterion, in the original variant defined as a minimum number of other objects within this radius. A cluster consists of all density-connected objects (which can form a cluster of an arbitrary shape, in contrast to many other methods) plus all objects that are within these objects' range. Another interesting property of DBSCAN is that its complexity is fairly low - it requires a linear number of range queries on the database - and that it will discover essentially the same results (it is deterministic for core and noise points, but not for border points) in each run, therefore there is no need to run it multiple times.

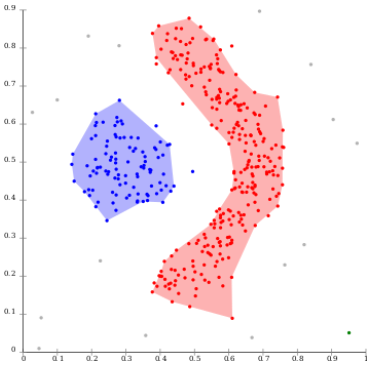


Figure1.2.5 Density-based clustering with DBSCAN.

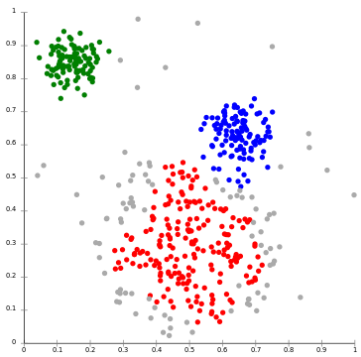


figure1.2.6 DBSCAN assumes clusters of similar density, and may have problems separating nearby clusters

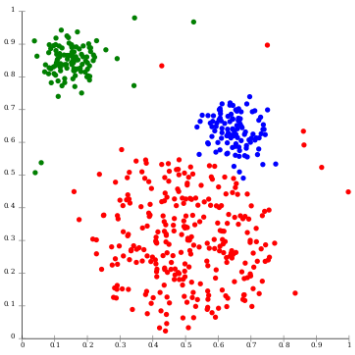


figure1.2.7 OPTICS is a DBSCAN variant that handles different densities much better

1.3 K-means clustering

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

Given a set of observations (**x1, x2, …, xn**), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k (≤ n) sets S = {*S1, S2, …, Sk*} so as to minimize the within-cluster sum of squares (WCSS) (i.e. variance). Formally, the objective is to find:

$$\arg_s\min\sum_1^k \sum_{Si}\|x-\mu i\|^2 \quad = \quad \arg_s\min\sum_1^k |Si| Var\ Si \qquad 1.3.1$$

where μ_i is the mean of points in *Si.* This is equivalent to minimizing the pairwise squared deviations of points in the same cluster:

$$\arg_s\min\sum_1^k \frac{1}{2|Si|}\sum_{Si}\|x-y\|^2 \qquad 1.3.2$$

The Equivalence can be deduced from identity

$$\sum_1^k ||x-\mu i||^2 = \sum_{Si} (x-\mu i)(\mu i - y) \qquad 1.3.3$$

Because the total variance is constant, this is also equivalent to maximizing the squared deviations between points in different clusters.

Chapter2 Code section

Code section 1: Main function

step 1: load data

```
dataSet = []
fileIn = open('testSet.txt')
for line in fileIn.readlines():
  lineArr = line.strip().split('\t')
  dataSet.append([float(lineArr[0]), float(lineArr[1])])
```

Test: step 1

```
#   print(dataSet)
#   print('\n')
"""
[[1.658985, 4.285136]]
[[1.658985, 4.285136], [-3.453687, 3.424321]]
[[1.658985, 4.285136], [-3.453687, 3.424321], [4.838138, -1.151539]]
[[1.658985, 4.285136], [-3.453687, 3.424321], [4.838138, -1.151539],
[-5.379713, -3.362104]]
```

step 2: clustering

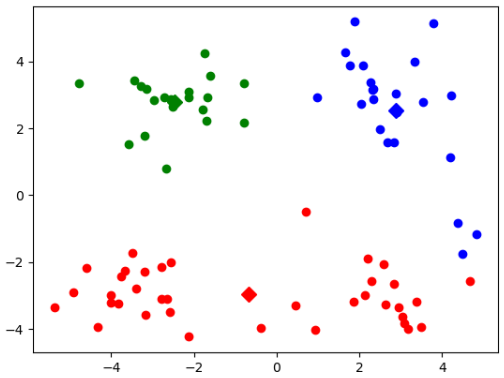
```
k = 10
dataSet = mat(dataSet)
centroids, clusterRepret = kmeans(dataSet, k)
```

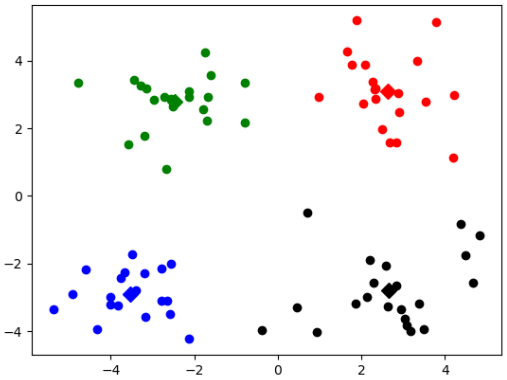
Test: step 2

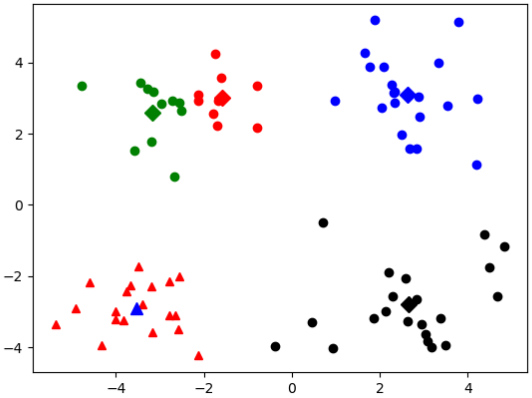
```
#print(dataSet)
#print('\n')
"""
[[ 1.658985  4.285136]
 [-3.453687  3.424321]
 ...
 ...
 [ 4.479332 -1.764772]
 [-4.905566 -2.91107 ]]
"""
```

step 3: show the result

```
print ("step 3: show the result...")
showCluster(dataSet, k, centroids, clusterRepret)
```







Code section 2: k-means function

step 1: calculate Euclidean distance

```
def euclDistance(vector1, vector2):
    return sqrt(sum(power(vector2 - vect
```

step 2: init centroids with random samples

```
def initCentroids(dataSet, k):
    Sample_nums, dim = dataSet.shape
    for i in range(k):
        index = int(random.uniform(0, Sample_nums))
    return centroids
```

step 3: k-means cluster

1. init centroids
2. find the centroid who is closest
3. update its cluster
4. update centroids

```
    centroids = initCentroids(dataSet, k)
    while clusterChanged:
        clusterChanged = False
        ## for each sample
        for i in range(Sample_nums):
            #print(i)
            minDist  = 100000.0
            minIndex = 0
            ## for each centroid
            ## step 2: find the centroid who is closest
            for j in range(k):
                distance = euclDistance(centroids[j, :], dataSet[i, :])
                if distance < minDist:
                    minDist  = distance
                    minIndex = j
                    print('distance(centroids['+str(minIndex)+',:],
dataSet['+ str(i) + ', :]) = ' + str(distance))
```

```
            ## step 3: update its cluster
            if clusetr_group[i, 0] != minIndex:
                clusterChanged = True
                clusetr_group[i, :] = minIndex, minDist**2
        ## step 4: update centroids
        for j in range(k):
            pointsInCluster = dataSet[nonzero(clusetr_group[:, 0].A == j)[0]]
            centroids[j, :] = mean(pointsInCluster, axis = 0)
    return centroids, clusetr_group
```

Chapter 3 Results and Conclusion

The result of the lab is shown as following. When K are different, the clustering results are different.

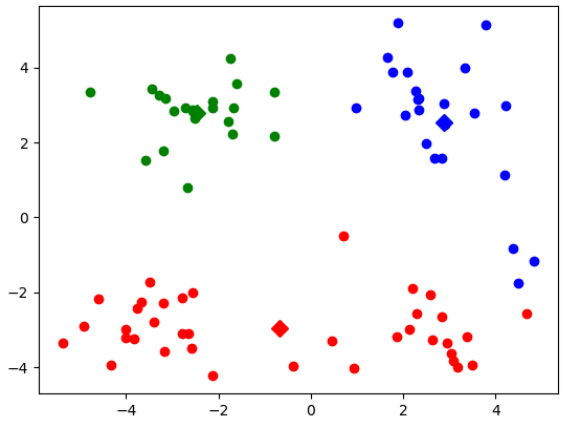


Figure 3.1

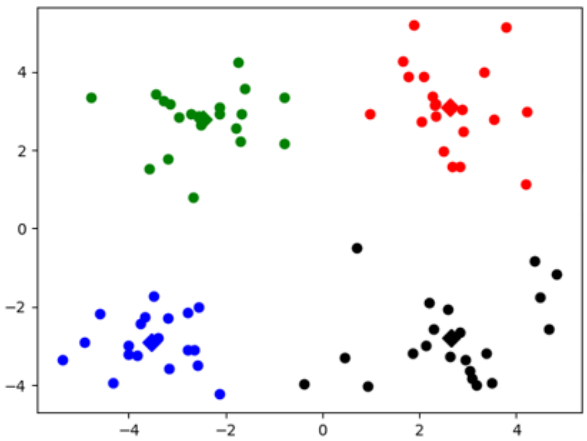


Figure 3.2

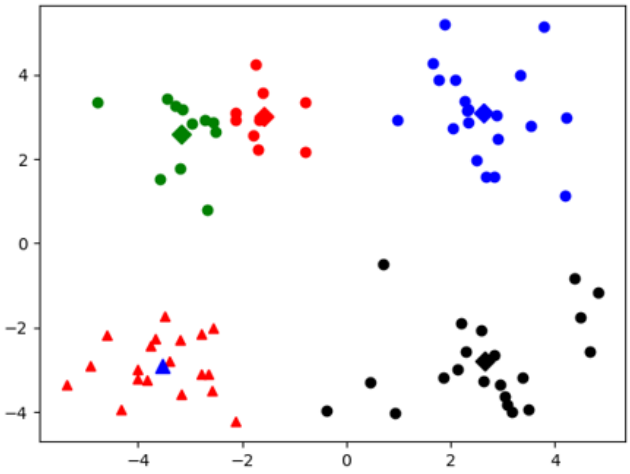


Figure 3.3

Appendix

Original codes are as following:

```
from numpy import *
import time
import matplotlib.pyplot as plt
import sys



# calculate Euclidean distance
def euclDistance(vector1, vector2):
    return sqrt(sum(power(vector2 - vector1, 2)))


# init centroids with random samples
def initCentroids(dataSet, k):
```

```
    numSamples, dim = dataSet.shape
    centroids = zeros((k, dim))
    for i in range(k):
        index = int(random.uniform(0, numSamples))
        centroids[i, :] = dataSet[index, :]
    return centroids


# k-means cluster
def kmeans(dataSet, k):
    numSamples = dataSet.shape[0]
    # first column stores which cluster this sample belongs to,
    # second column stores the error between this sample and its centroid
    clusterRepret = mat(zeros((numSamples, 2)))
    clusterChanged = True

    ## step 1: init centroids
    centroids = initCentroids(dataSet, k)

    while clusterChanged:
        clusterChanged = False
        ## for each sample
        for i in range(numSamples):
            #print(i)
            minDist    = 100000.0
            minIndex = 0
            ## for each centroid
            ## step 2: find the centroid who is closest
            for j in range(k):
                distance = euclDistance(centroids[j, :], dataSet[i, :])
                if distance < minDist:
```

```
                    minDist    = distance
                    minIndex = j

            ## step 3: update its cluster
            if clusterRepret[i, 0] != minIndex:
                clusterChanged = True
                clusterRepret[i, :] = minIndex, minDist**2

        ## step 4: update centroids
        for j in range(k):
            pointsInCluster = dataSet[nonzero(clusterRepret[:, 0].A == j)[0]]
            centroids[j, :] = mean(pointsInCluster, axis = 0)

    print ('Congratulations, cluster complete!')
    return centroids, clusterRepret

# show your cluster only available with 2-D data
def showCluster(dataSet, k, centroids, clusterRepret):
    numSamples, dim = dataSet.shape
    if dim != 2:
        print ("Sorry! I can not draw because the dimension of your data is not 2!")
        return 1

    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']
    if k > len(mark):
        print ("Sorry! Your k is too large! please contact Zouxy")
        return 1

    # draw all samples
    for i in range(numSamples):
```

```
        markIndex = int(clusterRepret[i, 0])
        plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])

mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']
# draw the centroids
for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize = 8)

plt.show()
```