K-Means Clustering

Term Project

in

Data Analytics course (CS40003)

by

Chandan Ritvik

(16CS30010)

Under instruction of

Dr. Debasis Samanta

Department of Computer Science & Engineering

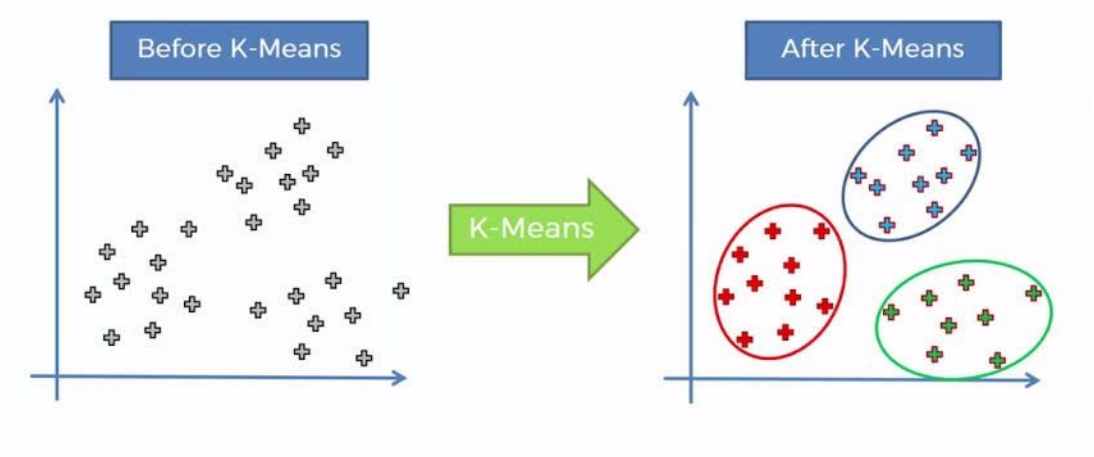
Indian Institute of Technology, Kharagpur

November 2019

Theory

Among the various Data Analytics techniques which are taught in this course, K-Means clustering is one the most important unsupervised methods. It is used to get an intuition about the structure of the data. In this case, we are provided with data pertaining to cancer.

If the task is successful, then would be able to find out whether the tumours are benign or malignant, just by using different attribute values like 'Bare Nuclei', 'Mitoses',etc.. This task is performed by identifying subgroups (clusters) such that data points in the same subgroup are very similar while data points in different clusters are very different. They are separated by hard boundaries.



Prior labelled data as such is not required for this task, but in the dataset ('CANCER.txt'), we are given labelled data. In the algorithm that we used for K-Means clustering, these labels are neglected initially while calculating clusters, but finally the 2 labels ('benign' & 'malignant') are found to be matching with each of the one clusters. Hence, we used labelled classes for Validation but not for Training clusters.

The first attribute named 'id' has been completely dropped in the following analysis. Instead, index of data-point is used as reference whenever required.

Pandas dataframe has been used to read the input file. Then this dataframe was used to construct a 2D list. **Euclidean distance** has been used as the distance metric in this k-means algorithm. We take into account all the 9 attributes while calculating distances.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

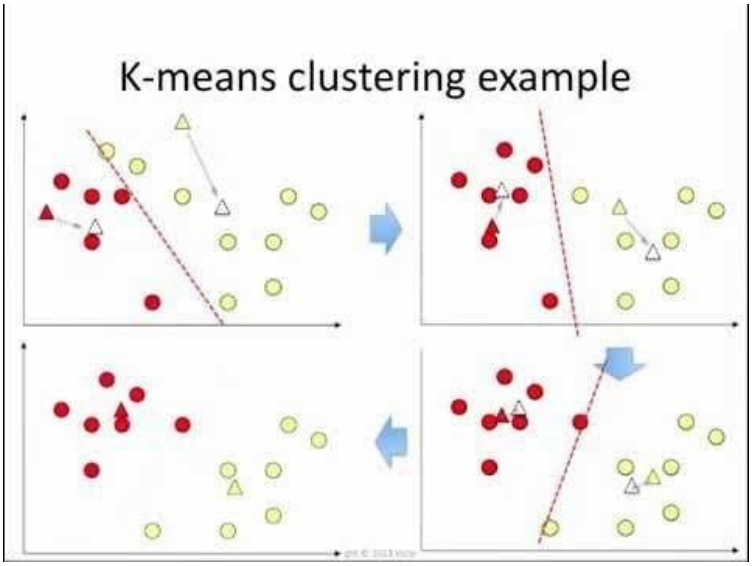
The Algorithm takes as input - k (number of clusters) & dataset.

This clustering algorithm itself is run 5 times to randomly choose initial clusters again & again. Out of the 5 clusters thus obtained, the best one is chosen as final cluster.

For each of the chosen initial clusters, we perform 10 iterations to arrive at the 'final clusters'. The algorithm easily converges in 10 iterations for this dataset. Each iteration improves the clusters' quality. The iteration itself has 2 steps:

1. **Build k clusters** (subgroups) using the k initial cluster centroids, by assigning each data-point to it's closest centroid

2. **Calculate k cluster centroids** by averaging the data-points in each cluster.



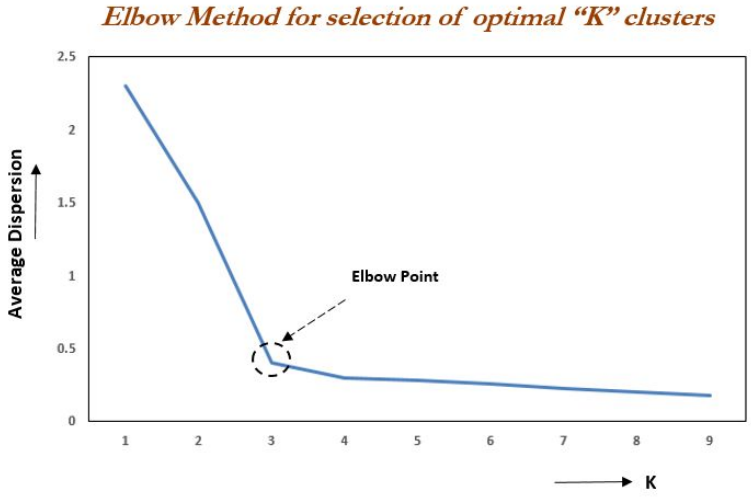
For performing 1st step in each iteration, we would have to calculate sum of squared distances between all data points and all centroids. There are 9 dimensions in data. Hence this would have Time Complexity of O(9*n*k). The second step would just take O(9*n).

Since initial clusters are chosen 5 times randomly and each one take 10 iterations to find final clusters. the time complexity of the algorithm for a particular value of 'k' becomes O(5*10*9*n*k) (or) **O(450*n*k)**.

There are 16 data-points with missing attribute values. It has been observed that in all these cases, an attribute called 'Bare Nuclei' is only missing. The missing parts ('**?**') in these data-points, is overlooked while calculating Euclidean distances. But after the training is complete, the centroids are used to calculate the approximate missing values for all of these 16 data-points.

To **decide the best value of 'k'** for this dataset (CANCER.txt), we have run this same algorithm for different values of k & then evaluated cluster qualities for each one, using silhouette method & elbow method.

Elbow method - we calculate 'SSE' (Sum of Squared Error) for each 'k'. Then we plot it on a graph taking 'SSE' in y-coordinate & 'k' on x-coordinate. The SSE generally decreases with increasing 'k'. When an 'elbow'(i.e., a point, such that,the decrease after this point is much less than the decrease before this point), is observed in the graph, it is taken as the optimal value of 'k' for the given dataset.



Silhouette analysis - For a given k, we have already computed the best clusters. Now, for each data-point:

C(i) - Cluster assigned to the ith data point
|C(i)| – Number of data points in C(i)

a(i) gives a measure of how nicely the ith data point is assigned to it's own cluster

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

b(i) gives the average dissimilarity to the closest cluster

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j)$$

s(i) - Silhouette coefficient is calculated as follows:

$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

s(i) close to 1 → the point is well clustered
s(i) close to 0 → the point is poorly clustered

By taking average of s(i) for all points, we get silhouette score for the given 'k'. The 'k' value which has **highest silhouette score is chosen as optimal value of 'k'**.

Implementation:

```
import pandas as pd
from matplotlib import pyplot as plt
import math
import random

def gen_matrix(df):
#Convert Dataframe to 2D list
 lst=[]
 for ind,row in df.iterrows():
    templst=[]
    for j in range(9):
     templst.append(df.iloc[ind,j])
    lst.append(templst)
 return lst

def present(lst,a):
# Check if 'a' is present in list
 for x in lst:
    if(x==a):
       return True
 return False

def gen_random(mat,k):
# Generate k random centroids
 len1=len(mat)
 c1=[]
 used=[]
 for _ in range(k):
    r=random.randrange(0,699)
#Check repetition
    while(present(used,r)):
       r=random.randrange(0,699)
    c1.append(mat[r])
    used.append(r)
 return c1

def eucl(a,b):
# Calculate Euclidean distance
 sq=0
 for i in range(9):
    if(a[i]=='?' or b[i]=='?'):
    # Neglect missing values
       continue
    sq=sq+pow(int(a[i])-int(b[i]),2)
 return math.sqrt(sq)

def classify(clus,tup):
# Classify a data-point
# into one of k clusters
 close_ind=-1
 dist,dist_min=0,999999
 for i in range(len(clus)):
    dist=eucl(clus[i],tup)
    if dist<dist_min:
       close_ind=i
       dist_min=dist
 return close_ind
```

```
def avg(group,mat):
# Find centroids by averaging
 clus=[]
 for i in range(len(group)):
    if(len(group[i])==0):
     continue
    a=0
    sum_feat=[0]*9
    for t in group[i]:
      for val in range(9):
        if(mat[t][val]=='?'):
         a+=1
         continue
        sum_feat[val]+=int(mat[t][val])
    for val in range(9):
      if(val==5):
       sum_feat[val]/=(len(group[i])-a)
      else:
       sum_feat[val]/=len(group[i])
    clus.append(sum_feat)
 return clus
def k_means(init_clus,mat):
# The Main Algorithm
 clus=init_clus
 for _ in range(10): #10 iterations
    group=[[] for x in
range(len(init_clus))]
    for i in range(len(mat)):
      num=classify(clus,mat[i])
      group[num].append(i)
    clus=avg(group,mat)
 return clus,group
# 'group'-> 2D list of clusters
```

```
def silhouette_score(clus,mat,group):
 a=[0]*len(mat)
 b=[0]*len(mat)
 for lst in group:   #Update a[i]
   for i in lst:
     dist=0
     for j in lst:
       if(j==i):
         continue
       dist+=eucl(mat[i],mat[j])
     dist/=len(lst)-1
     a[i]=dist

 for lst in group:   #Update b[i]
   for i in lst:
    dist_clus=[]
    for lst2 in group:
     if(lst==lst2 or len(lst2)==0):
      continue
     dist=0
     for j in lst2:
      dist+=eucl(mat[i],mat[j])
     dist_clus.append(dist/len(lst2))
    b[i]=min(dist_clus)

 sum=0
 for i in range(len(mat)):
 # Calculate average s[i]
   if(a[i]==0 and b[i]==0):
     continue
   sum+=((b[i]-a[i])/max(a[i],b[i]))
 return sum/len(mat)
```

```
def elbow_plot(mat):
  sselst=[]
  klst=[i for i in range(1,9)]
  for k in klst:
    init_clus=gen_random(mat,k)
res_cl,group=k_means(init_clus,mat)
    sseklst=[]
    for t in range(6):
      sse=0
      for i in range(k):
        for j in group[i]:
          sse+=(eucl(mat[j],res_cl[i])**2)
      sseklst.append(sse)
    sselst.append(min(sseklst))
  plt.plot(klst,sselst)
  plt.show() #Plot Here

def roundit(clus):
# Round-off all values
    for i in range(len(clus)):
        for j in range(9):
            clus[i][j]=round(clus[i][j],2)
    return clus
```

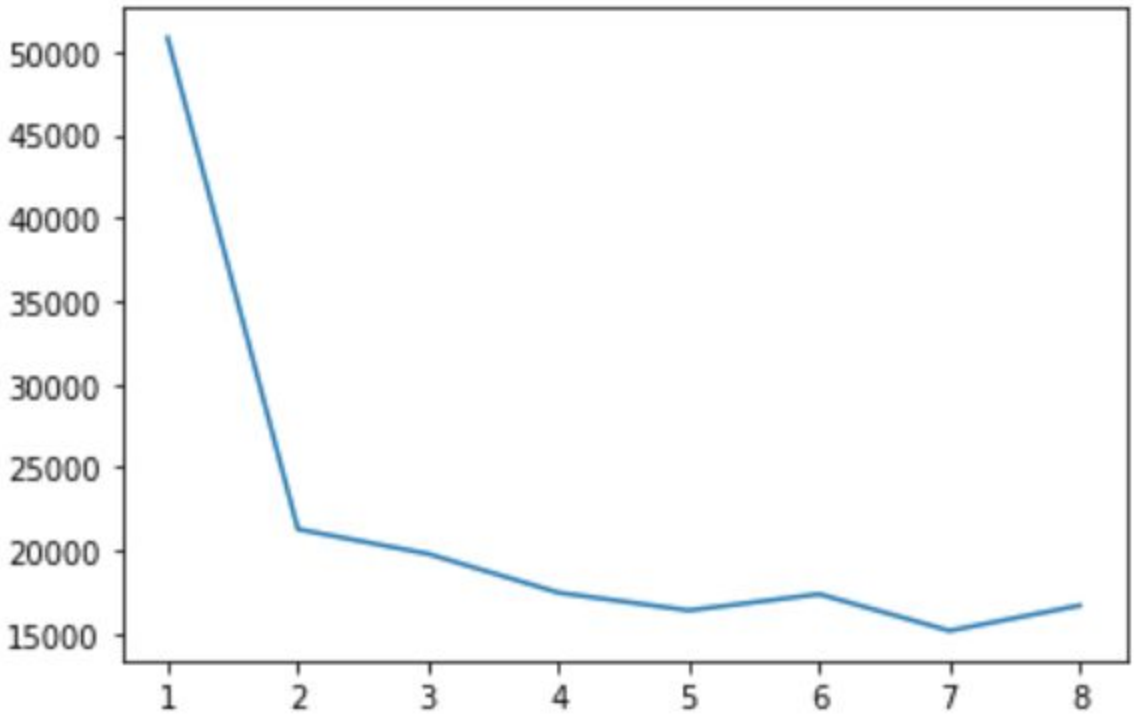
```
if __name__== "__main__":
  df = pd.read_csv('CANCER.txt',
sep=",", header=None)
  df.columns = ["id", "a", "b", "c", "d",
          "e", "f", "g", "h", "i","class"]
  df=df.drop(['id'],axis=1)

  mat=gen_matrix(df)
  # elbow_plot(mat)
  k=2
  init_clus=gen_random(mat,k)
  res_cl,group=k_means(init_clus,mat)
  print(roundit(res_cl))

print(silhouette_score(res_clr,mat,group))
```

Experimental Results:

Elbow method: The graph is plotted using matplotlib in python.



X-coordinate → k
Y-coordinate → SSE

Thus, it is clearly observable that **k=2** is the elbow point.

Silhouette Analysis: Silhouette scores are as follows:

K	2	3	4	5	6	7
Score	0.5917	0.4915	0.4771	0.2418	0.2445	0.1805

Again, Silhouette score is maximum for **k=2**.

Accuracy of Classification:

Further analysis would be performed with k=2. Since, we are already provided with class labels, accuracy can be easily calculated (for k=2) as follows:

	Benign	Malignant
Cluster - 1 **(2.92, 1.24, 1.36, 1.29, 2.03, 1.28, 2.05, 1.2, 1.06)**	444	8
Cluster - 2 **(7.15, 6.6, 6.6, 5.59, 5.39, 7.64, 5.98, 5.92, 2.55)**	14	233

It can concluded that:

cluster - 1 corresponds to 'Benign'
cluster - 2 corresponds to 'Malignant'

Training set Accuracy = (444+233)/699 * 100% = **96.85 %**

For the sake of using **Hold-Out method** several times, a validation set of size around 100 is kept aside each time. So, in these cases, training was performed only with around 600 data-points. The average accuracy is computed.

Testing accuracy (or) Validation set accuracy = 94.37%

Filling up missing attribute values: To make best guess of missing values in 'Bare Nuclei', we try to reduce the euclidean distance from the data-point to its respective class cluster, by assigning a suitable value.

Out of 16 data-points, 14 of them belong to 'benign' & 2 to 'malignant'.

For 'benign', we assign 1.28 to 'Bare Nuclei' ('?' → 1.28)
For 'malignant', we assign 7.64 to 'Bare Nuclei' ('?' → 7.64)