

COMP-4400 FINAL PROJECT



University
of Windsor

TEAM MEMBERS

NAME: Shivani Pansara

STUDENT ID: 104874374

NAME: Aman Patel

STUDENT ID: 104956768

Main Aim

The Aim of this project is to implement the most popular clustering algorithm named Kmean clustering algorithm. We took this opportunity to not only implement this algorithm but also implemented Kmean++ algorithm to further optimize the output of Kmean Cluster (More on Kmean++ algorithm in the later part of this documentation)

BRIEF INTRODUCTION TO KMEANS ALGORITHM

Algorithm Kmeans:

Input: Dataset: $[[x_1, y_1, z_1], [x_2, y_2, z_2], \dots [x_N, y_N, z_N]]$ and K (Number of clusters)

Output: $[[[All\ Datapoints\ in\ cluster1], [All\ Datapoints\ in\ Cluster2], \dots [All\ Datapoints\ in\ ClusterK]]]$

begin:

1. Randomly Choose an initial K centroid from our datasets $C = \{cs_1, cs_2, cs_3, \dots, cs_K\}$.
2. while (Centroids to which each data points belongs to no longer change) do:
 - 2.1 For each $i \in \{1, 2, \dots, K\}$, assign each C_i as the set or collection of data points closer to c_i than they are to c_j where $j \neq i$. (We use Euclidean distance to find the distance between two vectors)
 - 2.2 Recalculate the centroids c_i based on the mean of the Data points in set C_i where $i \in \{1, 2, \dots, K\}$, i.e updated $c_i = (1/\text{length}(C_i)) * (\sum x)$, where $x \in C_i$
3. endwhile
4. return $[C_1, C_2, \dots, C_K]$

The overall idea of the Kmeans algorithm is:

1. Given a set of input data points and the number of clusters represented as K
2. Initialize centroids arbitrarily selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids i.e., assignment of data points to clusters isn't changing.

Why is Kmeans algorithm helpful?

Kmeans is a popular unsupervised clustering algorithm used in variety of applications such as image compression. This algorithm can be used to maximize the similarity of data points within clusters and minimize the similarity if data points in different clusters. It helps in giving a meaningful intuition of the structure of the data that we are dealing with.

For example, it is simple and flexible to use kmeans in a retail business. With this people can cluster large data sets in a short amount of time which is necessary when dealing with such kind of data. One other application is document clustering wherein after the application of the algorithm the similar documents are in the same clusters.

Advantages of K-means

- In unlabeled Data Sets where we often do not know how instances in a data set should be grouped, an algorithm like K-means is helpful.
- It is used to easily discriminate different clusters in a nonlinearly separable data like concentric circles.
- It is relatively simple to implement.

Disadvantages of K-means

- One of the biggest disadvantages of Kmeans is choosing the value of k manually.
- If the clusters have a complex geometric shape, Kmeans does a poor job in clustering the data.
- Different initial partitions can result in different final clusters.

Kmeans++

One of the disadvantages of the K-means algorithm is that it is sensitive to the way in which initialization step. In Kmeans algorithm initialization of initial K centers is done by choosing any K elements randomly where probability of choosing any data points among N points is uniform (i.e $1/N$). [This](#) paper proposed a variant that chooses initial K centers at random from the data points but weighs the data points according to their squared distance from the closest center already chosen. Based on this paper, this novel approach of choosing the centroids is both fast and simple and helps achieving guarantees that K-means cannot. (Our Implementation in no mean tries to re-prove the work already done by them. Proof for the above claim can be found in the linked research paper)

Following this idea of choosing the centers based on its distance from the already choose point, this is the algorithm we implemented:

Algorithm Kmeans++:

Input: Dataset, K

Output: List of initial K Centroids

begin

1. Randomly select the 1st Centroid. Add it to a List named *AlreadySelectedCentroids*
2. *DistanceMap* := [];
3. *for i* := 1 to k-1 *do*
 - for i* := 1 to |Dataset| *do*
 - current_min* := +infinity;
 - for j* := 1 to |AlreadySelectedCentroids| *do*
 - current_min* = min(*current_min*, euclidain_distance(DataSet[i], Centroids[j]));
 - rof*
 - Add *current_min* in *DistanceMap*;
 - rof*
 - S* := the index of largest element in *DistanceMap*
 - Add Centroids[S] to *AlreadySelectedCentroids*
4. *rof*
5. return *AlreadySelectedCentroids*

BRIEF DESCRIPTION OF OUR CODE

1. Initialization Step:

- For initialization we used two algorithms to compare the results. One is Kmean++ and another is the traditional algorithm using random initialization.

2. Clustering step:

- After the initialization step algorithm begin the clustering using begin_clustering function.
- With the list of centroids initially chosen (eventually with newly calculated centroids), the get_cluster_map function assign each datapoints into different clusters based on the Euclidean distance calculated using euclidean_diatance function.
- The get_cluster_map gives us the array of numbers depicting which datapoint belongs to which cluster.
- The datapoint nearest to a particular centroid is assigned to that cluster.
- Based on the list obtained from get_cluster_map, the datapoints are grouped into cluster using cluster_mapping and cluster_mapping_helper functions.
- The centroid is then recalculated using the centroids_calc function for each cluster obtained in the above step.
- Then again, the clustering step is invoked to perform the clustering using new centroids.
- This process goes on until the centroids to which each datapoint belongs to doesn't change.

3. Mapping Step:

- The above step after converging gives an array of numbers which shows that the datapoint at that index belongs to which cluster.
- So, in order to group the datapoints into cluster, cluster_mapping function is invoked which do the grouping and gives away the output.

SAMPLE INPUT-OUTPUT:

INPUT:

Algorithm takes the dataset containing all the datapoints, number of clusters and an output variable as an input.

For Example:

kmean([[1.0, 2.0], [2.0, 3.0], [3.0, 4.0], [4.0, 5.0], [5.0, 6.0]], 2, Clusters).

OUTPUT:

After entering the desired input in the required format, it gives away the datapoints partitioned in different clusters.

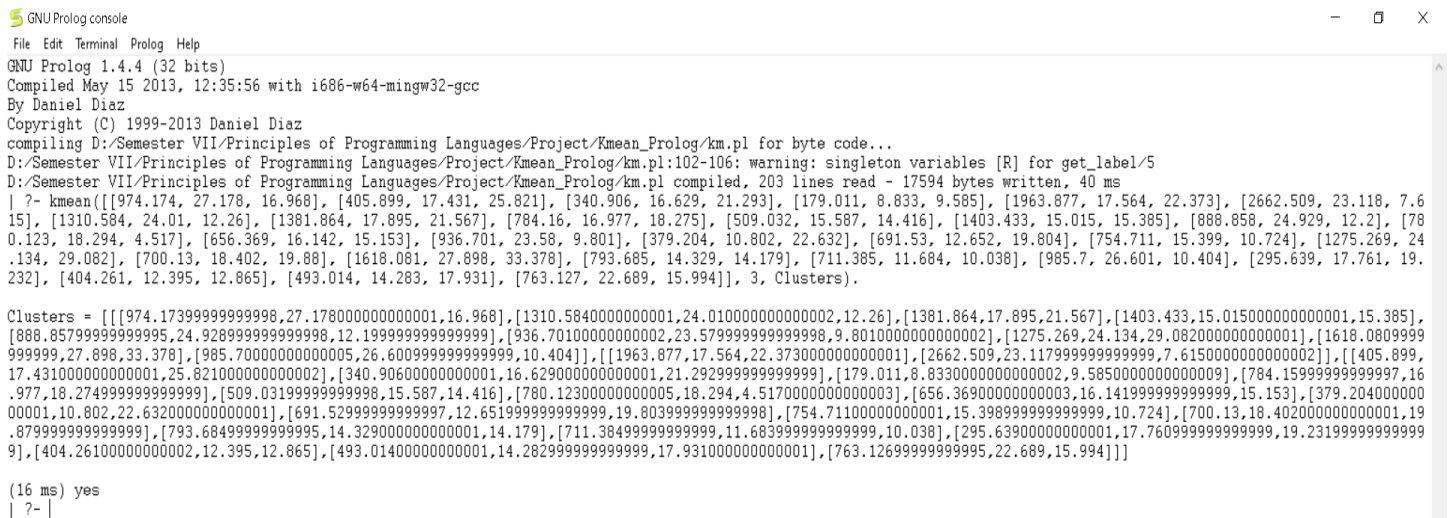
For Example:

Clusters = [[[1.0, 2.0], [2.0, 3.0], [3.0, 4.0]], [[4.0, 5.0], [5.0, 6.0]]]

TEST INPUT DATA:

We have tested this algorithm on a small 3D dataset. It is a carbon emission dataset in different states in India. The [original](#) file was in .docx format which was converted into [.csv](#) format containing only the x, y, z datapoints. Further, in order to convert the input compatible to the input format of our algorithm, a [python code](#) is used that takes datapoints from .csv file and converts it into the required input format.

[Here](#) is the screenshot of a sample run:



```
GNU Prolog console
File Edit Terminal Prolog Help
GNU Prolog 1.4.4 (32 bits)
Compiled May 15 2013, 12:35:56 with i686-w64-mingw32-gcc
By Daniel Diaz
Copyright (C) 1999-2013 Daniel Diaz
compiling D:/Semester VII/Principles of Programming Languages/Project/Kmean_Prolog/km.pl for byte code...
D:/Semester VII/Principles of Programming Languages/Project/Kmean_Prolog/km.pl:102-106: warning: singleton variables [R] for get_label/5
D:/Semester VII/Principles of Programming Languages/Project/Kmean_Prolog/km.pl compiled, 203 lines read - 17594 bytes written, 40 ms
| ?- kmean([[974.174, 27.178, 16.968], [405.899, 17.431, 25.821], [340.906, 16.629, 21.293], [179.011, 8.833, 9.585], [1963.877, 17.564, 22.373], [2662.509, 23.118, 7.6
15], [1310.584, 24.01, 12.26], [1381.864, 17.895, 21.567], [784.16, 16.977, 18.275], [509.032, 15.587, 14.416], [1403.433, 15.015, 15.385], [888.858, 24.929, 12.2], [78
0.123, 18.294, 4.517], [656.369, 16.142, 15.153], [936.701, 23.58, 9.801], [379.204, 10.802, 22.632], [691.53, 12.652, 19.804], [754.711, 15.399, 10.724], [1275.269, 24
.134, 29.082], [700.13, 18.402, 19.88], [1618.081, 27.898, 33.378], [793.685, 14.329, 14.179], [711.385, 11.684, 10.038], [985.7, 26.601, 10.404], [295.639, 17.761, 19.
232], [404.261, 12.395, 12.865], [493.014, 14.283, 17.931], [763.127, 22.689, 15.994]], 3, Clusters).

Clusters = [[[[974.1739999999999,27.178000000000001,16.968],[1310.5840000000001,24.010000000000002,12.26],[1381.864,17.895,21.567],[1403.433,15.015000000000001,15.385],
[888.8579999999999,24.928999999999998,12.199999999999999],[936.7010000000000,23.579999999999998,9.801000000000000],[1275.269,24.134,29.082000000000001],[1618.0809999
999999,27.898,33.378],[985.7000000000000,26.600999999999999,10.404]],[[1963.877,17.564,22.373000000000001],[2662.509,23.117999999999999,7.615000000000000]],[[405.899,
17.431000000000001,25.821000000000002],[340.9060000000000,21.292999999999999],[179.011,8.833000000000002,9.585000000000000],[784.1599999999997,16
.977,18.274999999999999],[509.0319999999998,15.587,14.416],[780.1230000000000,23.579999999999999,9.801000000000000],[656.3690000000000,16.141999999999999,15.153],[379.204000000
0000,10.802,22.632000000000001],[691.5299999999997,12.651999999999999,19.803999999999998],[754.7110000000001,15.398999999999999,10.724],[700.13,18.402000000000001,19
.879999999999999],[793.6849999999995,14.329000000000001,14.179],[711.3849999999999,11.683999999999999,10.038],[295.6390000000001,17.760999999999999,19.231999999999999
9],[404.2610000000000,12.395,12.865],[493.0140000000001,14.282999999999999,17.931000000000001],[763.1269999999995,22.689,15.994]]]
```

OBSERVATIONS

As mentioned before our algorithm uses Kmean++ algorithm for initialization of centroids. We compared our results obtained using traditional random initialization and Kmean++ algorithm that we have incorporated in our algorithm.

We observed that Kmean++ algorithm gives better cluster than the traditional random initialization.

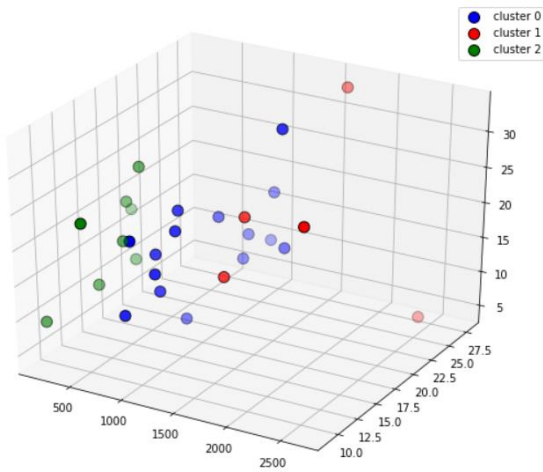


FIGURE 1: Random Initialization

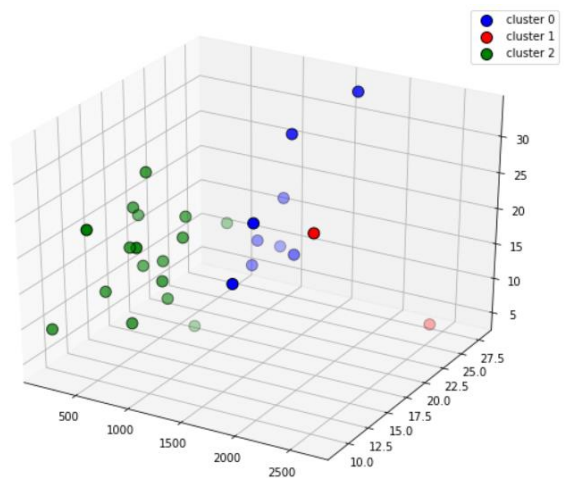


FIGURE 2: Kmean++ Algorithm

The **FIGURE 1** above is the clustering results obtained when random initialization is used. All the three clusters are congested and not very clear.

Whereas the **FIGURE 2** above is when Kmean++ algorithm is applied. The clusters obtained are comparatively clear and separated.