**PROJECT 1 REPORT**

**Submitted by:**

**Adiba Arif**

**Mustafa Bharmal**

Q1) Describe the 2 assignment methods you implemented and discuss the Big O for each one (K-Means). Note that you do not need to discuss the round robin assignment strategy, which everyone must implement.

Ans1) 2 assignment methods:

1. Pillaring(): In this method, we first find the mean of the list (i.e. mean of all the points) and then calculate the distance from each point to the mean and then the point with the maximum distance is the first centroid. Then again we calculate the distance from each point to the 1st centroid and save these distances. We then find the point with the furthest distance from the 1st centroid and this becomes the 2nd centroid. We continue doing this until all K centroids have been designated.

Big O:

Variables used

p = number of points

k = no of clusters

For random function:

* random() takes O(k) to assign random points to clusters
* k\_means() takes:
* O(p\*k) in best case, in this case for each point we find distance from point and each centroid. The points are placed in correct cluster in the first iteration only.
* O((p\*k)^2) in worst case, in this case for each point we find the distance from point and each centroid. The points can change from one cluster to another (max is k).
* assignColor() takes O(p+k)
* printClusters() takes O(k)
* squaredDistError() takes O(p+k)

Hence, for random k means, the overall complexity is

* O((k)+(p\*k)+(p+k)+k+(p+k)) = O(p\*k) in best case
* O((k)+((p\*k)^2)+(p+k)+k+(p+k)) = O((p\*k)^2) in worst case

1. random(): In this method, we randomly choose points from the input list as centroids depending upon the given number of clusters (if the user enters 6 clusters, then 6 centroids will be formed).

Big O:

Variables used

p = number of points

k = no of clusters

For pillar function:

* pillar() takes O(k\*p), for each cluster we calculate pillar by traversing all points
* k\_means() takes:
* O(p\*k) in best case, in this case for each point we find distance from point and each centroid and the points are placed in correct cluster in first iteration only
* O((p\*k)^2) in worst case, in this case for each point we find distance from point and each centroid and the points can change from one cluster to another (max is k).
* assignColor() takes O(p+k)
* printClusters() takes O(k)
* squaredDistError() takes O(p+k)

Hence, for pillar k means, the overall complexity is

* O((k\*p)+(p\*k)+(p+k)+k+(p+k)) = O(p\*k) in best case
* O((k\*p)+((p\*k)^2)+(p+k)+k+(p+k)) = O((p\*k)^2) in worst case

Conclusion: pillar() takes more execution time as its pillar() is O(p\*k) and that of random() is O(k), but the overall complexity is the same.

Q2) Analyze your results when using different data sets-

1. When using K-Means, did any of the 3 assignment strategies produce better clusters? Why or why not?

* We measured the “goodness” of the clusters, by calculating the Sum of square errors as well as check the number of iterations (but mainly we see the error rate). Error rate was calculated for all three strategies using 5 different data sets. On an average, the error rates were approximately same for all strategies with a minute difference but going by the definition of producing better clusters, we concluded that depending on the data set which we are using, the algorithm which generates the minimum error rate as compared to the others is the best strategy. From the table we can see that in two datasets, Random strategy produced better clusters as its error rate was less as compared to the other strategy. On the other hand, in two datasets we see that the Pillaring strategy produced better clusters. So, the strategy which minimum error rate will produce the best cluster.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Data sets | | Round-robin | Random | Pillaring | Minimum Error Rate |
| Data set 1 | Error rate | 34.064 | 33.840 | 33.618 | Pillaring |
| # of iterations | 12 | 5 | 10 |
| Data set 2 | Error rate | 33.016 | 34.175 | 33.644 | Round-robin |
| # of iterations | 7 | 9 | 9 |
| Data set 3 | Error rate | 33.121 | 32.683 | 32.297 | Pillaring |
| # of iterations | 7 | 4 | 7 |
| myData400 (400 points with 6 clusters) | Error rate | 37.229 | 34.074 | 34.294 | Random |
| # of iterations | 18 | 22 | 22 |
| DBSCANdata (30 points with 6 clusters) | Error rate | 29.905 | 26.619 | 28.315 | Random |
| # of iterations | 4 | 4 | 6 |

1. With data sets that produced weak clusters with one algorithm, were you able to see that the other algorithm performed better? Were there data sets that did not produce strong clusters with either algorithm? Be sure you include a summary of the data set(s) you used to test your code.

* We used DBSCANdata dataset and performed both the algorithms on it i.e. K means and DBSCAN. By running K-Means on it (K=6 i.e. number of clusters), we got the following graph showing 6 clusters. This algorithm produced clusters which were strong as well as weak.

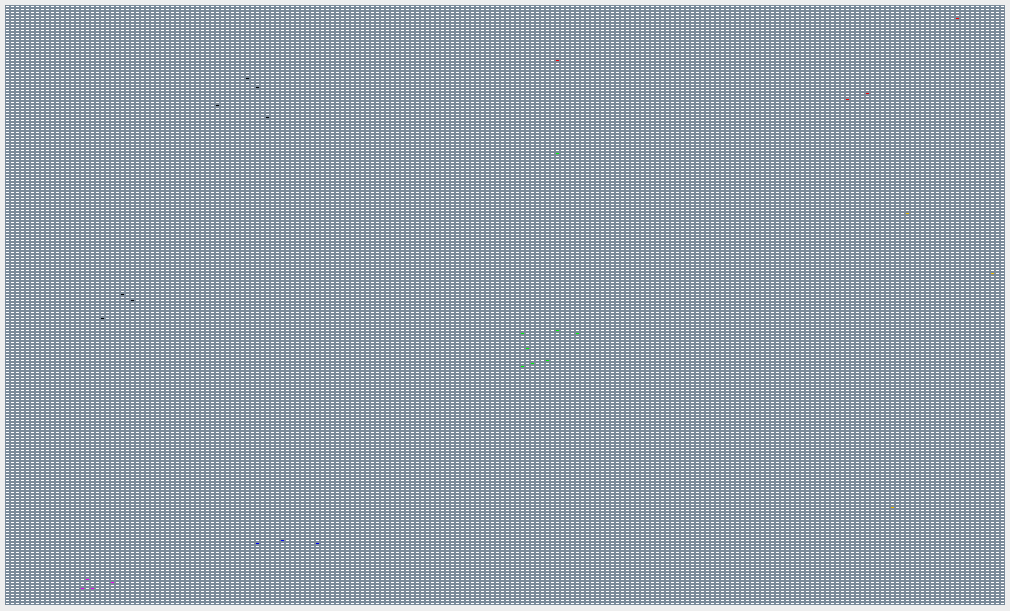


Figure 1: K-Means

* On the other hand by running DBSCAN on it (2 as minimum number of points and 50 as distance), we got the following graph showing 6 clusters. This algorithm produced clusters which are strong i.e. close to the central core and ignores/eliminates the noise points (A noise point is any point that is not a core point or a border point).
* The DBSCAN algorithm does not form unwanted connections between clusters. The assumption behind this algorithm is that the probability of picking a point near the central core of the distribution is much higher as the probability distribution is lighter near the edges. In DBSCAN the data points that are spread out are ignored and focus is on the dense parts which are present at the central cores of the cluster. Hence, it performs better as compared to K-Means.

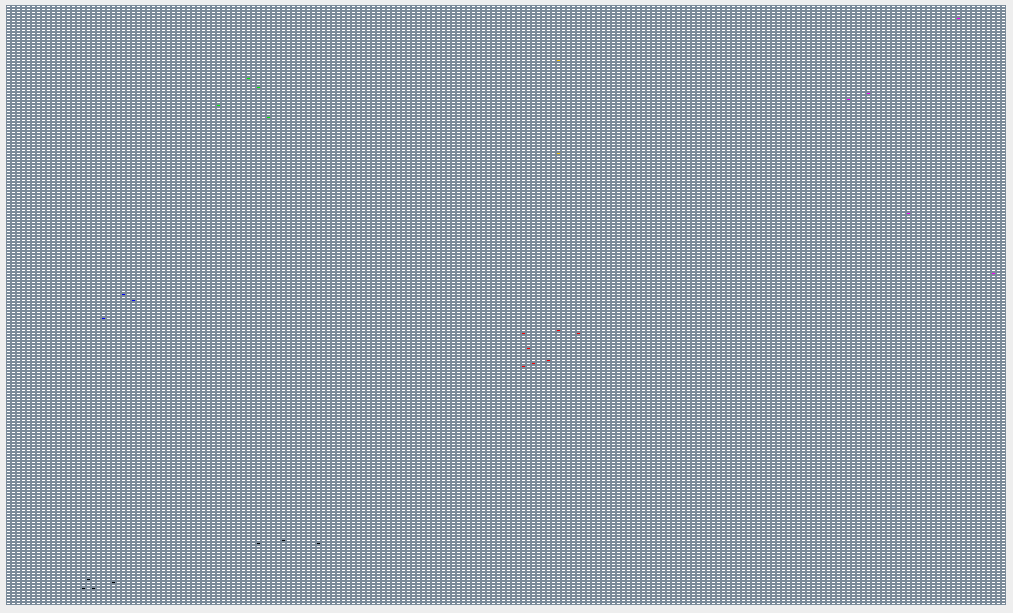


Figure 2: DBSCAN

Summary of datasets:

We used two datasets in order to test our code. The first one is myData400.txt, this data set was randomly generated (wrote a code which randomly generated the points). It consisted of 400 points and was tested with 6 clusters in K means. The outcomes were proper i.e. we could get the proper clusters and we could easily distinguish them.

The other data set we used is DBSCANdata.txt, this data set contains 30 points and the clusters were known before the execution of the algorithm. This data set was created to test DBSCAN algorithm (points were generated in such a way that they formed clusters and few were taken as outliers). When the algorithm was executed on the dataset, we got the actual outcome similar to the predicted one.