INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

MACHINE LEARNING

ASSIGNMENT 2

K-MEANS CLUSTERING

Prepared by :

Samar Pratap Singh (19EE10069)

Sawale Saurabh Suresh (19CS10054)

Group No.: 47 (E)

Running the Code:

The .ipynb along with the Indian Liver Patient Dataset (ILPD).csv file is already attached in the zip folder, which can be downloaded and run on Google Colab.

The code is recommended to run on Colab to avoid problems with installation packages or version incompatibility due to outdated versions.

The view link code is written on the google colab and the link for the same is here : https://colab.research.google.com/drive/1VMgCk7rdKR0Y4jdu3mxPvJANoJoywhbH?usp=sharing

- ### Importing Dataset

    1. The dataset used is https://www.kaggle.com/jeevannagaraj/indian-liver-patient-dataset
    2. We have split the dataset as **80:20 splits as train and test set** respectively.

    3. Attribute Information:

        -- 1. Age (4 to 90)
        -- 2. gender (Male or Female)
        -- 3. tot_bilirubin (0.4 to 75)
        -- 4. Direct_bilirubin (0.1 to 19.7)
        -- 5. tot_proteins (63 to 2110)
        -- 6. albumin (10 to 2000)
        -- 7. ag_ratio (10 to 4929)
        -- 8. sgpt (2.7 to 9.6)
        -- 9. sgot (0.9 to 5.5)
        -- 10. alkphos (0.3 to 2.8)

- ### Splitting Dataset for training and validation:

    We split the data set into 80/20 splits for training and testing, respectively, using the **test_train_split** function.

- ### Data Cleaning :

    1. Some data were missing in **alphakos** so, for that we replaced the missing data with its mean value in the dataset.
    2. **Gender** was in the form of string, so we made it as an integer for further operations by representing ***Male with 1 and Female with 0***.

3. Finally, we normalized our data for all further operations(as some features weren't comparable) using : normalized_X = $(X - X_{min})/ (X_{max} - X_{\min})$

Functions used in the project:

1. **test_train_split :** this function splits the dataset as 80/20 split.
2. **Initialize_centroids :** it takes the dataset and returns:
    a. a set of centroids, with the rows selected for centroids.
    b. New X and Y(training sets) dataframes with these centroids deleted from this dataframe for further applications.

    These are returned based on the **init** parameter. If it is default, the centroids are randomly initialized and by kmeans++ otherwise.
3. **kmeans :** The main function performing the K-means algorithm over the centroids obtained after the centroids are initialized from the *Initialize_centroids* function according to "random" or the "kmeans++" way.
4. **kmeanspp :** This function implements the kmeans++ initialization process by taking the dataframe and "K" as argument and returns the set of Centroids and the *k_rows* (rows from which we selected centroids).
5. **For all the Cluster-Performance calculations Sklearn's library functions are used:**
    a. **NMI :** `normalized_mutual_info_score()`
    b. **ARI :** `sklearn.metrics.adjusted_rand_score()`
    c. **Homogeneity :** `sklearn.metrics.homogeneity_score()`
    d. **Silhouette Index :** `sklearn.metrics.silhouette_score()`

For all the plotting functions, we have used python's Matplotlib function.
For min, max functions, we have directly used Python's min() and max() functions.

- ## K-means clustering implementation and algorithm :

  <u>Implementation :</u>

  This algorithm will regroup **n** data points into the K number of clusters. So given a large amount of data, we need to cluster this data into K clusters.
  We have written the function which implements the k-means algorithm, where "k" is given, and we perform k-means (for maximum of 100 iterations or until the centroids don't change in further iterations)

  Algorithm:
  Randomly choose k examples as initial centroids
  for i from 0 to max_iteration (here 100)
  update the centroids with the new centroids, which is the mean of all points assigned to a centroid's cluster(from 1 to k)

Assign X_train (i.e rows of the training dataset) points to different clusters based on the euclidean distance to form the new centroids.

break when the clusters don't change anymore to avoid unnecessary further computations

In Code implementation : The centroid initialization takes place which initializes the centroids based on whether we want random or kmeans++ initialization(implemented using kmeanspp). The centroids and data frames are then used to implement the actual kmeans algorithm to get the final list of centroids of the final clusters found.

- ## Calculating Clustering performance :
  ## <u>Implementation</u>

  - We have used k_means++ method for initialization of centroids.

  - with available ground truth:;
    - We have calculated the NMI (Normalized Mutual Information) score, ARI (Adjusted Rand Index), and homogeneity
  - without the ground truth:
    - We have calculated the silhouette score.

  - **NMI** :: Normalized Mutual Information (NMI) :
    It is a normalization of the Mutual Information (MI) score to scale the results between 0 (no mutual information) and 1 (perfect correlation). It is an external measure because we need the class labels of the instances to determine the NMI. Higher the NMI, better is the clustering.

  - **ARI :** Adjusted Rand index :
    It measures the consensus between the true, pre-existing observation labels and the labels predicted as an output of the clustering algorithm. Since the Rand index measures labeling similarity on a 0-1 bound scale, with one equaling perfect prediction labels.

  - **Silhouette score** : Silhouette score is used to evaluate the quality of clusters created using clustering algorithms such as K-Means in terms of how well the different clusters are distinguished from each other. It doesn't need ground truth for calculation of the cluster quality.

<u>Result :</u>

Let, the user Enter the value of K as 6. After applying the K-means algorithm by random initialization of centroids and Calculation of cluster performance, here is what we get:

```
Enter value of k :6
NMI_value = 0.07396833983946738
Silouette score = 0.223016098745844 63
Homogenity score = 0.12172184988028785
ARI_value = 0.07257700725891088
```

Analysis of result:

- The low values of NMI, ARI and homogeneity score shows the low similarity between the predicted labels and that of the true value of Outcome.
- Low score of the Silhouette score shows that the labelling the clusters are overlapping and are not very dense and separated. This value of Silhouette score shows that the model itself couldn't find very well-defined clusters, as silhouette index doesn't use ground truth.

- **Finding the most suitable "K" :**

Implementation :

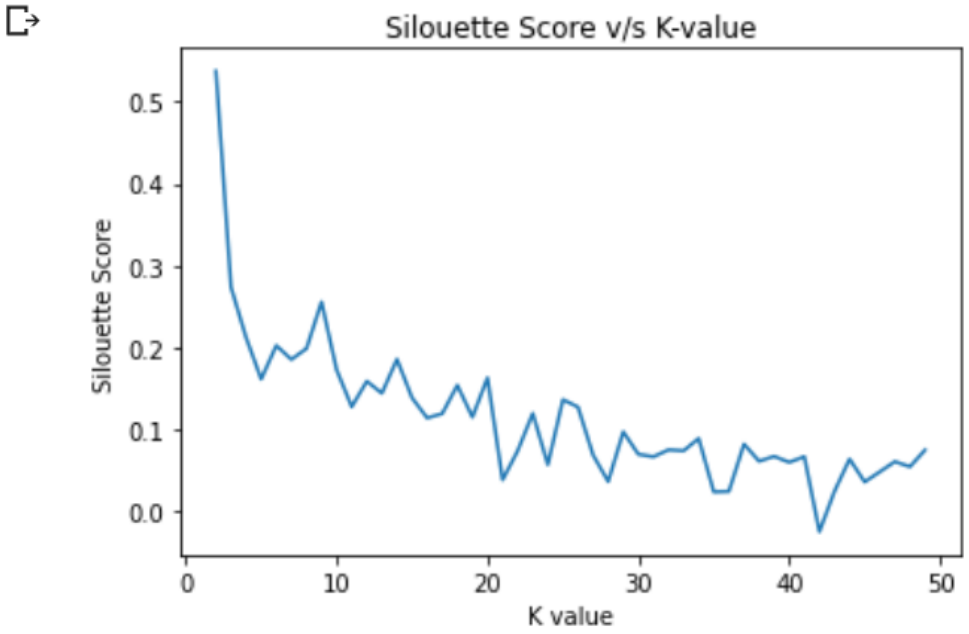
- We have used **silhouette index** to check for the performance for the k-means algorithm and choose the best k for our dataset.
- We plotted graphs silhouette score vs K-value (varying k from 2 to 50) to see which value of K was best suited for our dataset. A better value of Silhouette score tells us that the data can be modeled into well separated and identifiable clusters.

Result :

For the 1st few values, this is what we get:

```
k = 2
Silouette score = 0.537105763873247
k = 3
Silouette score = 0.2825742452063157
k = 4
Silouette score = 0.22045796366889447
k = 5
Silouette score = 0.14827417755543568
k = 6
Silouette score = 0.2617418836236739
k = 7
Silouette score = 0.2068601028650108
k = 8
Silouette score = 0.1038431476651345
k = 9
Silouette score = 0.12362837116792982
k = 10
Silouette score = 0.13931367452204776
```

- **Silhouette score vs k value plot:**



From the above graph, we can see that k-value = 2 gives the best value of silhouette score.

Analysis of result:

- We have used only silhouette score to determine the best suited value of K for our dataset and this set of centroids. As NMI, Homogeneity and ARI

all depend on available ground truth for their evaluation and silhouette index doesn't depend on ground truth index.
    - We only have to rely on how well our model clusters the given data as the basis for identifying the best value of K, as available ground truth is mostly not available in real life where we apply k-means clustering algorithm.
    - Silhouette score varies from [-1, 1], if score is 1 cluster dense and well separated from other cluster, if score is 0 clusters are overlapping, if score is negative then samples might have assigned wrong clusters
    - Which means silhouette score is high (close to 1) when the cluster is dense and well separated from other cluster.
    - For given dataset as k -value = 2 gives the best silhouette score. We can assume that k = 2 give much better score that other values of k

- **Finding whether there is any change in clustering outcome if you initialize K cluster centroids with random points in different execution. Heuristic used for initialization instead of random : "Kmeans++"**

  <u>Implementation:</u>

  TEST-A
  {
    1. select K random points from the original data
    2. split remaining data into 2 parts randomly (80:20) ratio
    3. apply k-means on training data with K-random points selected in step-1
    4. label test data using k-centroids.
    5. use some metric(NMI,ARI, etc.) in test data to understand clustering accuracy
    6. repeat 2-5 at least 50 times and report average metric

  Repeat 2-to 5 at least 50 times and report average metric

    - We have calculated the NMI/ARI/Homogeneity/Silhouette values with both ways(random and k-means++) of initialization of k centroids
    - We have taken k = 2, since it is the best value we are getting(Can take other values also if we want for analysis).
    - Furthermore, we have put the values of NMI/ARI/Homogeneity/Silhouette for plotting of their mean values over the 50 random k points and 50 Kmeans++ points

  }

So, we have done the **TEST-A** for both random and Kmeans++ initialization processes and then compare these two by the clustering performance.

Since, initially we pick K centroids randomly each time we run K-means clustering algorithm. It leads to different clusters being formed each time we run the algorithm, and so the final cluster centroids are different, thus affecting the clustering performance differently. Hence, The performance of the algorithm tends to become more random, since it may find good clusters at times, but at times, it may not.

Problem with random initialization:

In random initialization, if two initial centroids are very near, it would take a lot of iterations for the algorithm to converge and so something need to be done in order to make sure that initial centroids are far apart from each other.

Why we used K-means++ :

The objective of the KMeans++ initialization is that chosen centroids should be far from one another. The first cluster center is chosen uniformly at random from the data points that are being clustered, after which each subsequent cluster center is chosen from the remaining data points with probability proportional to its squared distance from the point's closest existing cluster center.

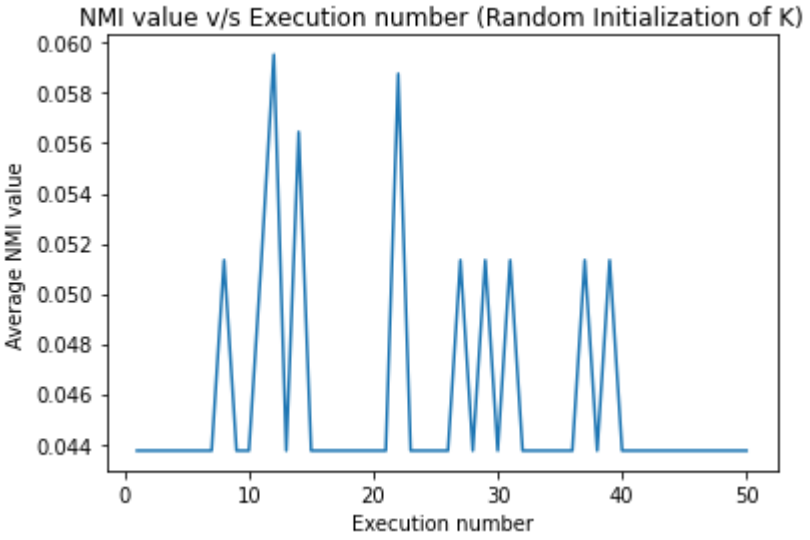
The algorithm followed is:

- `# Randomly select the first cluster center from the data points and append it to the centroid matrix:`
- `# Loop for choosing other centroids:`
    - `# For each data point, we calculate the square of Euclidean distance and append the minimum dist. to the Distance: "D" array.`
    - `# Calculate probabilities of choosing the particular data point as the next centroid by dividing the Distance array(D) elements with the sum of Distance array and then take the cumulative PD.`
    - `# choose a random no. between 0 and 1.`
    - `# choose the index of the Cumulative PD which is just greater than the chosen random number`
    - `# Assign the data point corresponding to this selected index`
    - `# Add this point in our Centroids`
- `# Finally, return the Centroids and the selected rows`

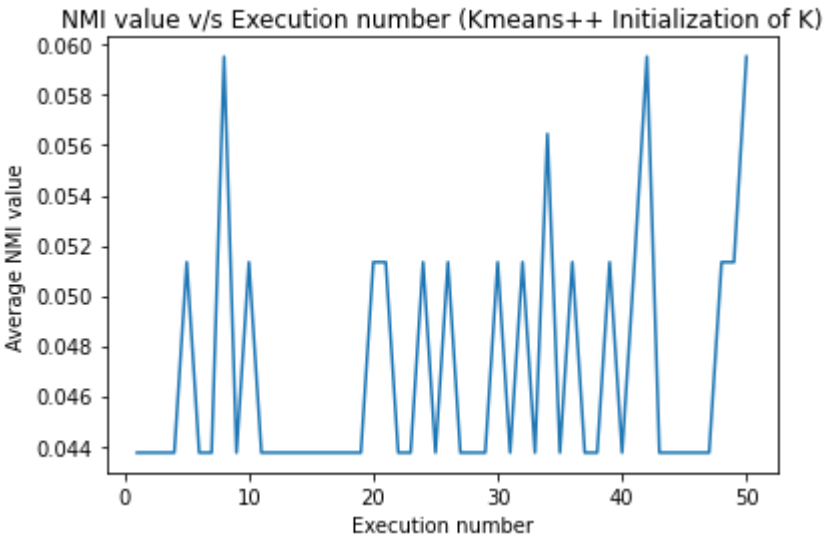
Results :

- **NMI values vs Execution number:**

  Dispersion :
  random_max_NMI = 0.05953283021330544
  random_min_NMI = 0.04374193068377192
  random_mean_NMI = 0.04567584670131092



  Dispersion :
  kmeanspp_max_NMI = 0.05953283021330544
  kmeanspp_min_NMI = 0.04374193068377192
  kmeanspp_mean_NMI = 0.04691840746604097



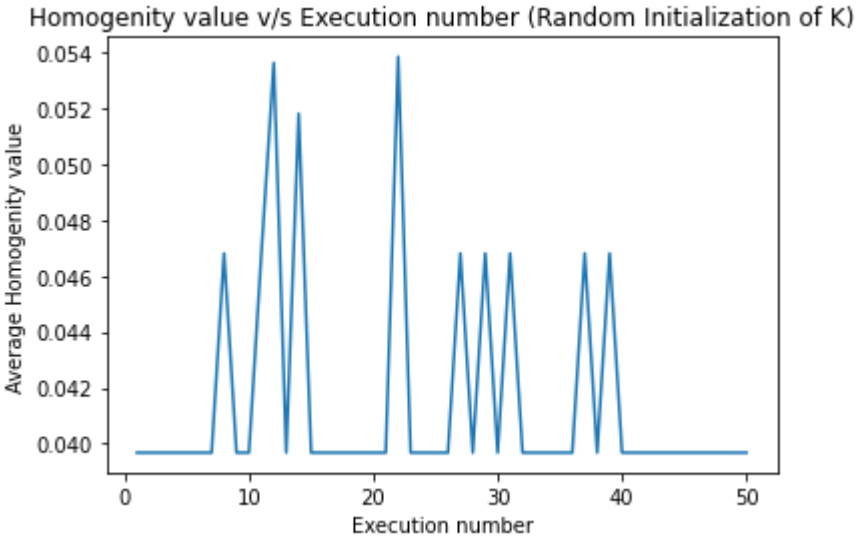
- **Homogeneity value vs Execution number:**

Dispersion :

random_max_Homogenity = 0.05386635231218007

random_min_Homogenity = 0.03967308820753225

random_mean_Homogenity = 0.04147929887518107

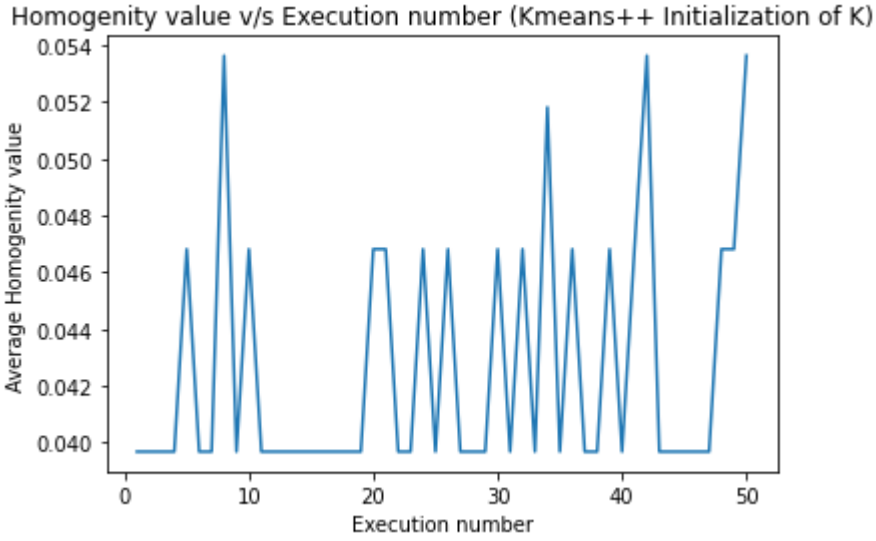


Dispersion :

kmeanspp_max_Homogenity = 0.053642681787040285

kmeanspp_min_Homogenity = 0.03967308820753225

kmeanspp_mean_Homogenity = 0.04261134045425048



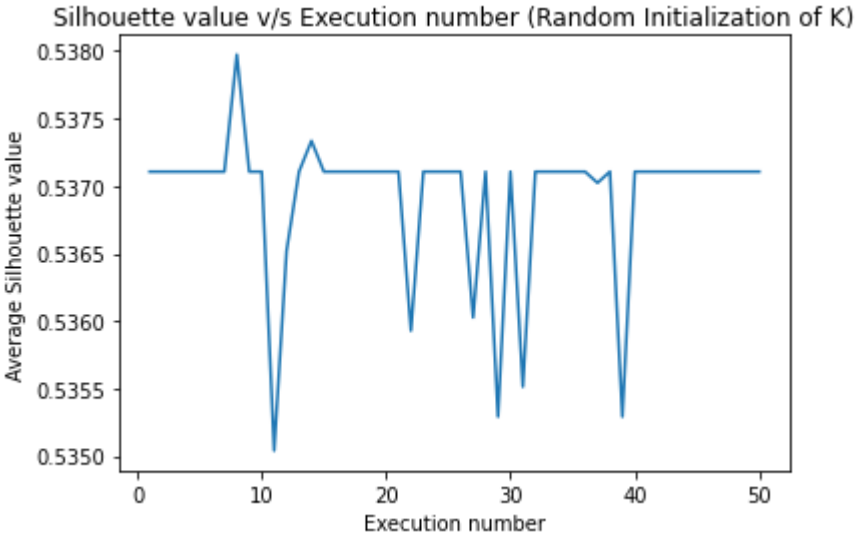
- **Silhouette values vs Execution number:**

Dispersion :

random_max_Silhouette = 0.5379723290013212

random_min_Silhouette = 0.5350405386374878

random_mean_Silhouette = 0.5369234376860865

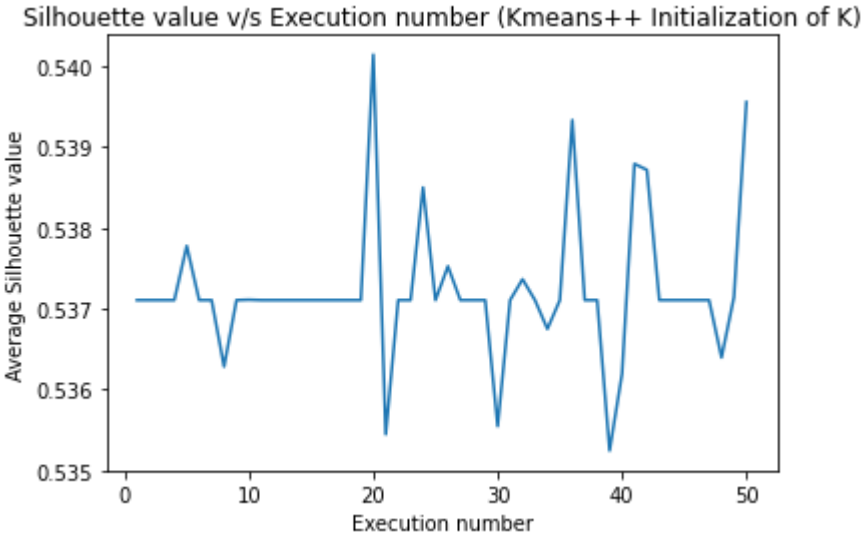


Dispersion :

kmeanspp_max_Silhouette = 0.5401420662758872

kmeanspp_min_Silhouette = 0.5352445889900314

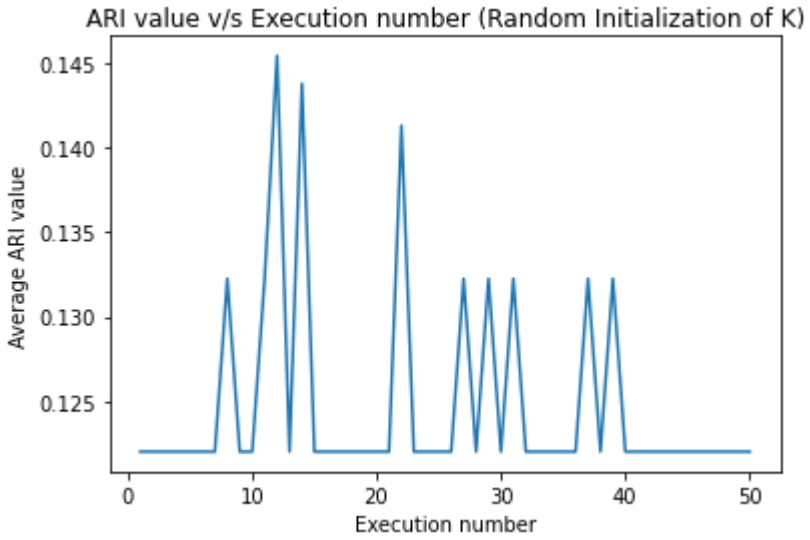
kmeanspp_mean_Silhouette = 0.5372240370423559



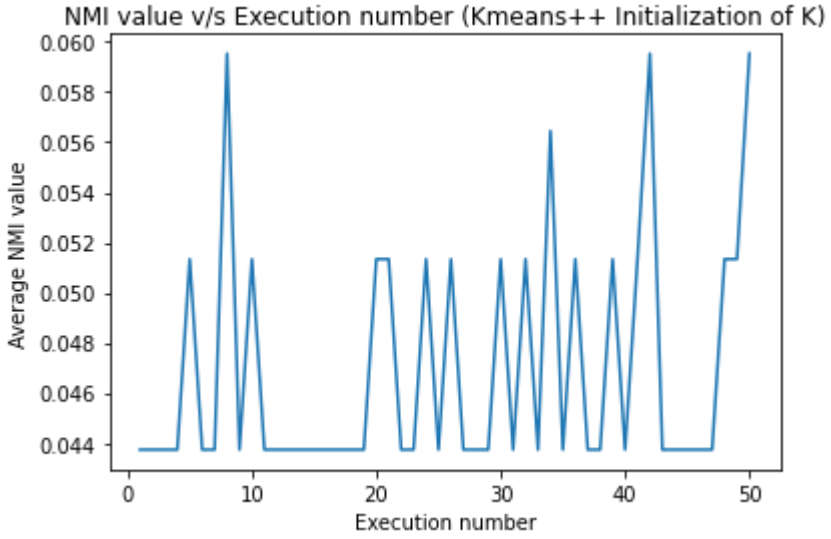
- **ARI values vs Execution number:**

  Dispersion :

random_max_ARI = 0.14547794917836554
random_min_ARI = 0.12200217478787241
random_mean_ARI = 0.12472952163060995



Dispersion :
kmeanspp_max_ARI = 0.14547794917836554
kmeanspp_min_ARI = 0.12200217478787241
kmeanspp_mean_ARI = 0.12651193022377719



Analysis:

- From the above graphs, it is clear that for both random initialization and Kmeans, the evaluation metrics change values with every execution.

- Although, it is important to note that the variation in values of cluster performances is quite less.
- When K-means++ instead of random initialization, the clusters tend to become more distinguished.
- The reason behind this is that the heuristic chooses the data point with the maximum chance of forming a new cluster.