**Activity: Concluding Conquest of Arrakis!**

In the Lab you have come across the (Lisan-Al-Gaib) algorithm. Let's recall some terminologies used. We had N spice points provided as input in a .txt file. We began with a guess estimate of the number of spice centers to be K. We used an iterative algorithm to cluster the entire set of spice points into spice clusters which are distributed around a spice center.

The K spice centers had indexing 0,1,2...K-1

The spice points lying in these clusters have label as the indexing to the spice center

**NOTE: This Question has 3 parts**

1. **Part 1 - Already done for you**
2. **Part 2 - Fill the necessary functions – autograder provided**
3. **Part 3 - Generate plots – no autograder provided – compare against plots shown in this pdf.**

**PART 1: INITIALISATION HEURISTIC** (Already **DONE** for you)

You might have noticed that the algorithm depends on the initialisation of the spice centers. For example a bad choice of initialisation can lead to the final clustering as in demo/default.png. To overcome this, we propose a different initialisation heuristic. This part has been already implemented for you, check the function `initialise\_spice\_centers` in submission.py.

The heuristic is described below:

1. Maintain a spice center set T, initially empty.

2. Add a random spice point to the set T.

3. Loop until size of T is K:

a. For every other spice point other than the spice centers chosen so far, calculate the distance to the nearest spice center, call this distance spice distance.

b. The spice point with the largest spice distance is added to the set T.

4. Return the set of spice centers.

The above heuristic adds one spice center at a time into the set T. The spice center is chosen such that it is farthest from the spice centers chosen so far. This heuristic is expected to perform better than random initialisation.

**Part 2: CLUSTERING METRICS[5 marks]**

To evaluate the final clustering output of the algorithm, we propose two metrics/scores namely

1. LAG score : To be implemented in function **calculate\_lag\_score() [1 mark]**
2. Silhouette score : To be implemented in function **calculate\_silhouette\_score() [4 marks]**

**NOTE: To obtain full marks, implementation should use only numpy functions and no for/while anywhere. We perform a naive check for presence of “for” and “while” in the code snippets inside the functions. So do not write these keywords even within comments! You are also not allowed to create new function definitions, or use eval or exec. Every presence of a loop keyword has a penalty of 1.5 marks, creation of new functions, or using eval, exec or any manipulation has straightaway 0 for this question.**

Before moving ahead, recall that after clustering, the spice points are clustered into K sets, with each set/cluster having a spice center. All spice points within a cluster are assigned the index of the spice center. So, every spice point within the same cluster has the same label.

**Assumption**: Every cluster has at least one spice point.

**LAG score**: Sum of square of distances between every spice point to nearest spice center.

Parameter Arguments : spicepoints, centers

Hint: One possible method to find this is calculate all distances between every spice point and every spice center and obtain a NxK matrix, use this to find the minimum for each spice point and return the sum.

**Silhouette Score**: Mean of silhouette of every spice point.

Parameter Arguments : spicepoints, centers, labels

For a spice point i, the silhouette(i) is defined as {b(i) - a(i)}/max(a(i),b(i))

where,

1. a(i) is the mean intra-cluster distance i.e., mean distance to a spice point within the same cluster
2. b(i) is the mean neighbor-cluster distance i.e., mean distance to a spice point within the nearest neighboring cluster. The notion of near is given based on distance of the spice point to the spice center of the clusters. [Obviously the nearest cluster to the spice point is the cluster label in which it lies, we need to choose the second nearest cluster label points as the nearest neighboring cluster.]

**Explanation**:

Let’s say we have spice points P1, P2, P3, P4, P5, P6 with labels: 0,0,1,1,0,2

Let distance(euclidean distance) between two points x,y be denoted by d(x,y)

a(P1) = mean(d(P1,P2), d(P1, P5)) // P2,P5 both are in the same cluster as P1 since label is same for three of them

a(P2) = mean(d(P2,P1),d(P2,P5))

a(P3) = mean(d(P3,P4))

a(P6) = 0 // since P6 is the only point in the cluster with label 2

Let the cluster centers be C0,C1,C2

To calculate b(P1) we first need to know d(P1,C0), d(P1, C1), d(P1, C2). The least would obviously be C0 as the label of P1 is 0. Let the second least is d(P1,C1).

b(P1) = mean(d(P1,P3), d(P1,P4))

**Hint**:

We break down the implementation into 3 parts:

1. Calculating an (N,) size np.array **a** which is the a(i) for all N spice points. **[2 points]**
2. Calculating an (N,) size np.array **b** which is the b(i) for all N spice points. **[2 points]**
3. Calculating the silhouette score: (This part is implemented)

**Part 3: PLOTTING SCORES FOR DIFFERENT INITIAL GUESSES OF K[1 mark]**

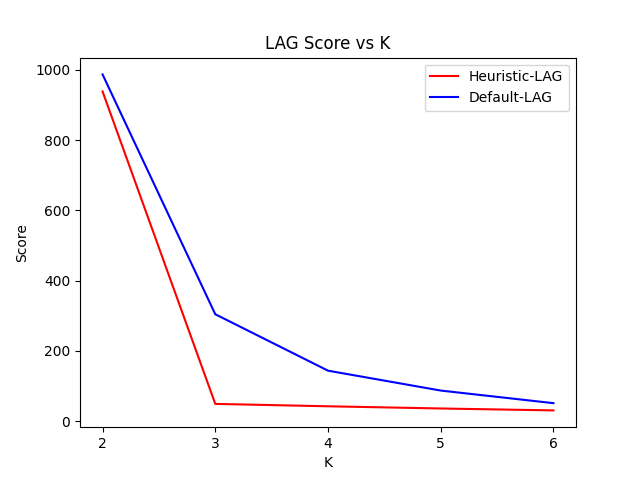
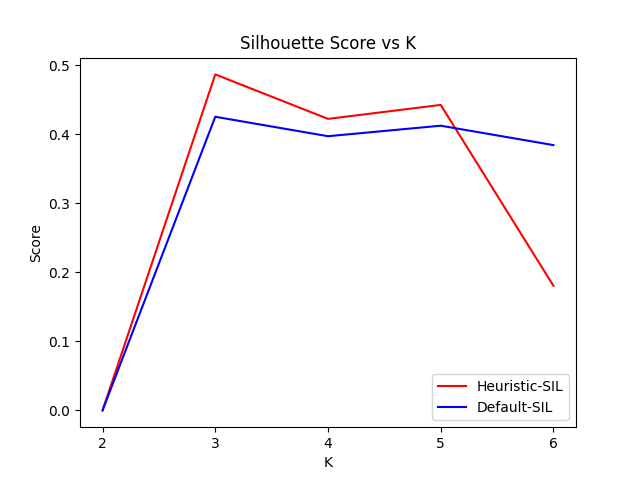
Refer to the function guess\_K(data\_path, max\_K, N\_iter). Code for simulating for N\_iter turns for values of K ranging from 2 to max\_K has been implemented for you. The scores for the default setting of K-Means, which was random initialisation, is stored in lag\_scores\_default and silhouette\_scores\_default, whereas the scores for the heuristic we proposed are stored in lag\_scores, silhouette\_scores.

Write code for generating 2 separate plots, for **LAG scores[0.5 marks]** and **silhouette scores[0.5 marks]** in **LAG\_scores.png** and **SIL\_scores.png** respectively.

The function code will not be evaluated, but the generated plots must be submitted for direct evaluation. You are supposed to run this function yourself and generate the plots to be submitted.

The plots must be generated for data\_path = “spicepoints.csv”, max\_K = 6, N\_iter = 500. Note, it may take some time(~10-15 secs) to generate plots.

Following is the scatter plot for the data in spicepoints.csv

The LAG\_scores.png and SIL\_scores.png look as follows:

The values may differ slightly from the above reference. Ensure you plot the design as shown, red line for scores of heuristic initialisation and blue lines for default. The label on the x-axis is ‘K’ and on the y-axis is ‘Score’. The title and legend of plots must be named as shown.

**Files to Submit: submission.py, LAG\_scores.png, SIL\_scores.png**

**FINAL marks: max(0, total marks including penalty)**