IMPLEMENTING CLUSTERING ALGORITHMS

Question 1: Brief Explanation of k-means Clustering Algorithm

The K-means algorithm is a popular unsupervised machine learning algorithm that aims to cluster data points based on their features or attributes. It is a representative-based algorithm that groups together data points that are similar. The algorithm quantifies the objective function of clustering by computing the sum of squares of Euclidean distances between data points and their closest representatives. In other words, the algorithm tries to minimize the sum of distances between data points and their respective centroids.

$$Dist(\overline{X_i}, \overline{Y_j}) = ||\overline{X_i} - \overline{Y_j}||_2^2.$$

By iteratively updating the centroids and reassigning data points to the closest centroid, the K-means algorithm creates clusters that are more homogeneous within themselves and more distinct from other clusters. It is a simple and effective algorithm that is widely used in various fields such as image segmentation, document clustering, and market segmentation. By using K-means, businesses can identify groups of customers with similar preferences, allowing them to create targeted marketing strategies. Researchers can also use K-means to group similar data points together, allowing them to analyze patterns and trends in the data.

Pseudo Code for the K-Means Algorithm

```
- function K-mean(Dataset, K, maxIter):
  // Initialization phase
  randomly_choose K points from Dataset as cluster representatives Y1, ..., Yk
  - repeat for i = 1 to maxIter:

    // Assignment phase
    - create empty clusters C1, ..., Ck
    - for each point X in the Dataset:
      find the closest cluster representative Yj based on squared Euclidean distance
      assign X to the cluster with representative Yj
      add X to the corresponding cluster Cj

    // Optimization phase
    - for j = 1 to k:
      compute the mean of the points in cluster Cj as the new cluster representative Yj

    // Check convergence
    - if no points have changed clusters:
      break // converged
  - return cluster representatives Y1, ..., Yk
```

Question 2: Brief Explanation of k-means ++ Clustering Algorithm

The K-means++ algorithm is an improvement over the traditional K-means algorithm that addresses the issue of initial centroid selection. By choosing the first centroid randomly and subsequent centroids based on the probability of selecting a data point proportional to its squared distance from the nearest centroid, the K-means++ algorithm avoids the problem of poorly placed initial centroids, resulting in better clustering results. It is widely used in various applications such as image segmentation, natural language processing, and data mining where clustering is required.

Pseudo Code for the K-Mean ++ Algorithm

```
function kmeans++(Dataset, K, Maxiter):
  // Initialization phase
  randomly_choose 1 point from Dataset as the first cluster representative Y1
  for i = 2 to K:
    for each point X in the Dataset:
      calculate the distance D(X) to its nearest cluster representative
    choose the next cluster representative Yi with probability proportional to D(X)^2
  set the initial cluster representatives to Y1, ..., Yk
  repeat for i = 1 to Maxiter:

    // Assignment phase
    create empty clusters C1, ..., Ck
    for each point X in the Dataset:
      find the closest cluster representative Yj based on squared Euclidean distance
      assign X to the cluster with representative Yj
      add X to the corresponding cluster Cj

    // Optimization phase
    for j = 1 to k:
      compute the mean of the points in cluster Cj as the new cluster representative Yj

    // Check convergence
    if no points have changed clusters:
      break // converged
  return cluster representatives Y1, ..., Yk
```

Question 3: Brief Explanation of Bisecting k-Means Hierarchical Clustering Algorithm

The bisecting k-means algorithm is a top-down hierarchical clustering method that recursively splits each node into two children using a 2-means algorithm. The algorithm selects the node to be split based on different growth strategies such as splitting the heaviest node or the node with the smallest distance from the root, which leads to balancing either the cluster weights or the tree height. To split a node, the algorithm uses randomized trial runs and selects the split that has the best impact on the overall clustering objective. Different variants of the bisecting k-means algorithm provide flexibility in balancing cluster weights or tree height, making it widely applicable in various fields.

Pseudo Code for the Bisecting k-Means Hierarchical Clustering Algorithm

```
function Bisecting K-mean (dataset, s):
  // Initialization phase
  create a tree T with a single node containing the whole dataset D
  set current number of clusters k = 1
  // Repeat until k == s
  while k < s:
    // Select a leaf node L with the largest sum of squared distance
    max_dist = -1
    for each leaf node L in T:
      compute the sum of squared distance ∑ dist(X, Y)**2 for cluster L
      if sum_of_squares > max_dist:
        max_dist = sum_of_squares
        selected_leaf_node = L
    // Split the selected leaf node L into 2 clusters L1, L2 using k-means algorithm
    cluster_representatives = k-mean(selected_leaf_node.dataset, 2, maxIter)
    L1 = a new node with cluster_representatives[1] as its representative and its dataset consisting of the points assigned to that cluster in the k-means algorithm
    L2 = a new node with cluster_representatives[2] as its representative and its dataset consisting of the points assigned to that cluster in the k-means algorithm
    // Add L1, L2 as children of L in T
    remove selected_leaf_node from T
    add L1, L2 as children of selected_leaf_node in T
    // Update the number of clusters
    k = k + 1
  // Return the leaf clusters
  return all leaf nodes in T
```

Question 4, 5 & 6: Running Algorithms for the Instance

	bisecting	kmean ++	kmean
0	0.152528	0.152528	0.152528
1	0.142493	0.141088	0.150064
2	0.136785	0.104731	0.147565
3	0.073847	0.103035	0.103086
4	0.074581	0.088564	0.099814
5	0.047996	0.093792	0.126276
6	0.065499	0.093855	0.080584
7	0.026984	0.100453	0.082719

table 1: Showing Silhouette score for the respective algorithms

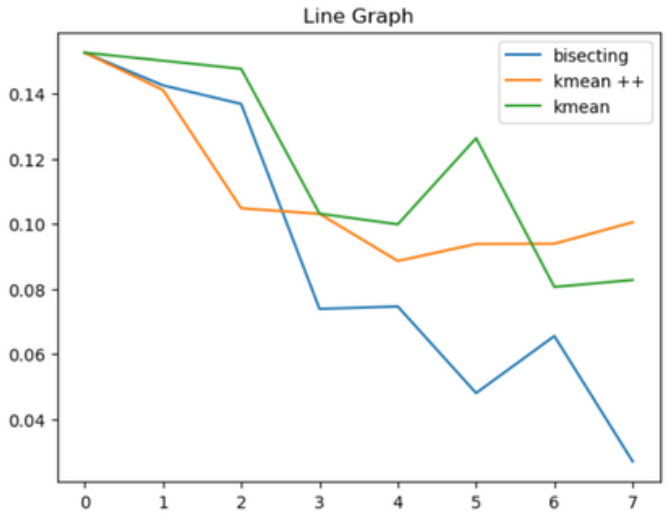

fig 1: Showing line chat of the respective Silhouette score

The *table 1* shows the Silhouette coefficients for different numbers of clusters (k) using three different clustering algorithms: bisecting k-means, k-means++, and traditional k-means. The Silhouette coefficient is a measure of how well-defined the clusters are, with higher values indicating better-defined clusters.

As we can see, the Silhouette coefficient generally decreases as the number of clusters increases, which suggests that it becomes harder to find well-defined clusters as the data is split into more groups. However, there are some exceptions, such as the jump from k=2 to k=3 for k-means, where the coefficient actually increases slightly.

Additionally, comparing the performance of the three algorithms, k-means++ generally outperforms the other two algorithms, especially for larger values of k. However, the results may vary depending on the specific dataset being analyzed.

Question 7: Running Algorithms for the Instance

label	kmean clusters	kmean ++ clusters	bisecting kmean clusters
radicchio	2	2	2
rhubarb	2	2	2
turnip	2	2	2
radish	2	2	2
courgette	2	2	2
pumpkin	2	2	2
potato	2	2	2
quandong	1	1	1
sunchokes	2	2	2
zucchini	2	2	2

From the *table 1* showing the Silhouette coefficient for each set of clusters, we can observe that the Bisecting K-means algorithm achieved the highest Silhouette coefficient of 0.152528 for k=2, which means that the data points are well-clustered and have high intra-cluster similarity and low inter-cluster similarity. However, for larger values of k, the Silhouette coefficient decreases for all three algorithms, indicating that the clusters become less distinct and less representative of the data. Looking at the table showing the words and their respective clusters for the last 10 results of the three algorithms, we can see that all three algorithms consistently assign the same labels to the same words, which means that they are achieving consistent results.

Reference

1. Müller, A. and Guido, S. (2016). Introduction to Machine Learning with Python. O'Reilly Media.
2. Raschka, S. (2015). Python Machine Learning. Packt Publishing.
3. Murphy, K.P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press.
4. Grus, J. (2019). Data Science from Scratch: First Principles with Python. O'Reilly Media.
5. Scikit-learn documentation: https://scikit-learn.org/stable/
6. Arthur, D. and Vassilvitskii, S. (2007). "K-means++: The Advantages of Careful Seeding". Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 1027-1035.
7. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R. and Wu, A.Y. (2002). "An Efficient k-Means Clustering Algorithm: Analysis and Implementation". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 881-892.