CS-233 Theoretical Exercise 13

May 2024

1 K-means Clustering

Question 1: Are the following statements true?

1. The standard way of initializing K-means is by setting $\mu_1 = \cdots = \mu_K$ to be equal to a vector of zeros.
2. For some datasets, the "right" or "correct" value of K (the number of clusters) can be ambiguous, and hard even for a human expert looking carefully at the data to determine.
3. One way to reduce the problem of K-means getting stuck in a bad local optimum is to try using multiple random initializations.
4. Since K-means is an unsupervised learning algorithm, it cannot overfit the data, and thus it is always better to have as large a number of clusters as is computationally feasible.

Solution: 1. False. 2. True. 3. True. 4. False

2 Hierarchical Clustering

Very often, clustering algorithms suffer a glaring problem: The number of clusters needs to be fixed before training. One would then have to rerun the entire training process if they were to use a different number of clusters. This imposes high computational costs for application scenarios where the number of clusters is unknown or where one would like to compute the clustering results for different numbers of clusters. In this exercise, we explore a different type of clustering algorithm that is free of such trouble. It is known as hierarchical clustering. More specifically, we are going to study agglomerative clustering, a subtype of hierarchical clustering[1].

Question 2: Let us walk through the process of a hierarchical clustering algorithm with a toy example. Suppose you are given four points A, B, C, and D. Their pairwise distances are denoted in the table on the right.

	A	B	C	D
A	0	1	4	5
B		0	2	6
C			0	3
D				0

We start by treating each point as a cluster. If we use a set to denote each cluster, the clusters we have at initialization of this algorithm are: {A}, {B}, {C}, {D}.
Question 2.1: If we would like to cluster these four points into three clusters, we can merge together the two clusters that have the closest distance. What would be the three clusters after this merging step?
Solution: {A,B}, {C}, {D}.
Question 2.2: Now, we are going to merge together two of the three clusters that you obtained in the last question so that we can cluster the four points into two clusters. Again, we are going to merge the two clusters that are closest to each other. We define the distance between two clusters to be *the minimum of the distances between two elements that belong to the two clusters*. What would be the three clusters after this merging step?

[1] The other subtype of hierarchical clustering is divisive clustering.

Solution: {A,B,C}, {D}.

Question 2.3: Notice that agglomerative clustering is often sensitive to how we compute the distance between clusters. If we define the distance between clusters to be *the maximum of the distances between two elements that belong to the two clusters*, what would be the two clusters we will then get?

Solution: {A,B}, {C,D}.

With this example, we see how agglomerative clustering effectively clusters data points with a flexible number of clusters. Admittedly, one always needs to compute more fine-grained clusterings before getting coarse-grained ones. Nevertheless, fine-grained clusterings can be of use later. Every step in this algorithm yields a proper clustering.

Question 3: Do the following statements accurately contrast K-means clustering with agglomerative clustering?

1. Agglomerative clustering generally requires more computational resources than K-means clustering.
2. In agglomerative clustering, the "rich get richer" phenomenon can occur, where larger clusters are more likely to merge with other clusters as the process iterates. In K-means, it doesn't.
3. K-means clustering is deterministic, while agglomerative clustering produces different results with each run due to its random initialization process.

Solution: 1. True. Agglomerative clustering involves calculating distances between all pairs of instances progressively until all are merged into a single cluster. By contrast, K-means does not require the computation of a distance matrix.

2. True. For agglomerative clustering, once a cluster starts growing, its increased size can make it statistically more likely to merge with other clusters. K-means clustering avoids this issue by recalculating the centroids of each cluster in every iteration based on the mean of the points currently assigned to that cluster.

3. False. Agglomerative clustering is deterministic in its standard form, where the results are consistent as long as its tie-breaking strategy (applied when clusters have the same minimum distance) is deterministic. By contrast, K-means clustering can produce different results across different runs due to random initialization of the centroids.

3 K-means Clustering Properties

In the K-means clustering algorithm, you are given a set of N points $\mathbf{x}_i \in \mathbb{R}^D$, $i \in \{1, \ldots, N\}$ and you want to find the centers of K clusters $\mu = (\mu_1, \ldots, \mu_K)$ by minimizing the average distance from the points to the closest cluster center. Formally, you want to minimize the following loss function

$$L(\mu) = \sum_{i=1}^{N} \min_{j \in \{1,\ldots,K\}} \|\mathbf{x}_i - \mu_j\|^2. \tag{1}$$

The K-means algorithm iterates between an assignment step and an update step to solve Eq. 1. In the assignment step, it assigns each point $\mathbf{x}_i$ to the z_i^*-th cluster with

$$z_i^* = \arg\min_{j \in \{1,\ldots,K\}} \|\mathbf{x}_i - \mu_j\|^2. \tag{2}$$

In the update step, it updates the cluster centers as $\mu_j^* = \frac{1}{|\{i:z_i=j\}|} \sum_{i:z_i=j} x_i$, which is the solution of

$$(\mu_1^*, \ldots, \mu_K^*) = \arg\min_{(\mu_1,\ldots,\mu_K)} \sum_{j=1}^{K} \left(\sum_{i:z_i=j} \|\mathbf{x}_i - \mu_j\|^2 \right). \tag{3}$$

Question 4: Show that K-means is guaranteed to converge to a local optimum. **Hint:** You need to prove that the loss function is guaranteed to decrease monotonically in each iteration until convergence.

Solution: To prove convergence of the K-means algorithm, we show that the loss function is guaranteed to decrease monotonically in each iteration until convergence for the *assignment step* and for the *update step*.

For the assignment step, we can rewrite Eq. 1 as

$$L(\mu, z) = \sum_{i=1}^{N} \|\mathbf{x}_i - \mu_{z_i}\|^2. \tag{4}$$

From Eq. 2, we know that $L(\mu, z^*) = \arg\min_z L(\mu, z) \leq L(\mu, z)$.

For the update step, we can rewrite Eq. 1 as

$$L(\mu, z) = \sum_{j=1}^{K} \left(\sum_{i:z_i=j} \|\mathbf{x}_i - \mu_j\|^2 \right). \tag{5}$$

From Eq. 3, we know that $L(\mu^*, z) = \arg\min_\mu L(\mu, z) \leq L(\mu, z)$.

Therefore, the loss function decreases monotonically in each iteration of the assignment step and the update step.

Question 5: What is the time complexity of the K-means clustering algorithm?

Solution: $O(INKD)$, where we have:

- D (dimensionality of the data): Computing the distance between two points in a D-dimensional space requires $O(D)$ operations.
- K (number of clusters): For each data point, the algorithm computes the distance to each of the K cluster centroids. This requires $O(D)$ operations per centroid, leading to a total of $O(KD)$ operations for each data point.
- N (number of data points): The algorithm iterates over each data point, computing the distances to all centroids, resulting in a total of $O(NKD)$ operations.
- I (number of iterations): The entire process is repeated for a certain number of iterations until convergence, which brings an additional factor of $O(I)$.

Question 6: While the Euclidean distance $\ell_2(\mathbf{x}_i, \mathbf{x}_j)$ is commonly employed in the K-means clustering algorithm and machine learning in general, alternative distance metrics are also available. The $\ell_0(\mathbf{x}_i, \mathbf{x}_j)$ distance computes the count of differing elements between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. The $\ell_1(\mathbf{x}_i, \mathbf{x}_j)$ distance, known as the Manhattan distance, sums the absolute differences between corresponding elements of $\mathbf{x}_i$ and $\mathbf{x}_j$. The $\ell_\infty(\mathbf{x}_i, \mathbf{x}_j)$ distance represents the maximum absolute difference between corresponding elements of $\mathbf{x}_i$ and $\mathbf{x}_j$. More generally, the $\ell_p(\mathbf{x}_i, \mathbf{x}_j)$ distance can be defined for all $1 \leq p < \infty$ as

$$\ell_p(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_p = \left(\sum_{k=1}^{D} |\mathbf{x}_i^{(k)} - \mathbf{x}_j^{(k)}|^p \right)^{1/p}.$$

How does $\ell_p(\mathbf{x}_i, \mathbf{x}_j)$ evolve as p increases? Provide a scenario where a low value of p would be advantageous and another where a high value of p would be preferable.

Solution: As p increases, the impact of larger differences between individual elements of the vectors becomes amplified. Consequently, when p is small, the distance metric is more sensitive to small differences between elements, whereas as p grows larger, the metric focuses more on the largest differences.

Scenario for low p value: In scenarios where small differences are crucial and larger differences are relatively insignificant, such as in image or audio processing where subtle variations need to be emphasized, a low value of p would be beneficial.

Scenario for high p value: In contrast, in situations where only major differences matter and minor discrepancies can be ignored, such as in outlier detection or anomaly detection, a higher value of p would be more suitable.