LAB 13 K-Means

The K-Means algorithm is an unsupervised machine learning algorithm used for clustering. It aims to partition a dataset into K distinct clusters, where each data point belongs to the cluster with the nearest mean (centroid). K-Means is an iterative algorithm that converges to a local optimum.

- **Explanation of how the K-Means algorithm works with an example:**

Step 1: Set the Number of Clusters

Start by selecting the desired number of clusters, denoted as K. This value is determined based on prior knowledge or by using domain expertise or techniques such as the elbow method.

Step 2: Initialize Cluster Centers

Randomly initialize K cluster centers in the feature space. These centers act as the initial centroids for the clusters.

Step 3: Assign Data Points to Clusters

Assign each data point to the cluster whose centroid is closest to it. This assignment is based on a distance metric, typically the Euclidean distance.

Step 4: Update Cluster Centers

Calculate the new centroid for each cluster by computing the mean of all data points assigned to that cluster. This step updates the cluster centers.

Step 5: Repeat Steps 3 and 4

Iteratively repeat steps 3 and 4 until convergence or a maximum number of iterations is reached. Convergence is achieved when the cluster assignments and cluster centers no longer change significantly.

Step 6: Finalize Clustering

The final result of the K-Means algorithm is a set of K clusters, each represented by its centroid. The data points are clustered based on their proximity to the centroids.

- **Example to illustrate the K-Means algorithm:**

Suppose we have a dataset of customer data with two numerical features: annual income and spending score. We want to segment the customers into distinct groups based on their income and spending habits.

- We start by setting the number of clusters, K, to, let's say, 3.
- We randomly initialize three cluster centers in the feature space.
- For each data point in the dataset, we calculate its distance to each of the three cluster centers and assign it to the nearest cluster.
- Once all data points are assigned to clusters, we update the cluster centers by calculating the mean of the data points in each cluster.
- We repeat steps 3 and 4 iteratively, updating the cluster assignments and cluster centers, until convergence.
- When convergence is reached, we have our final clustering result. Each customer is assigned to one of the three clusters based on proximity to the cluster centers.

The K-Means algorithm works well for datasets where the clusters are well-separated and have a spherical shape. However, it may struggle with datasets that contain irregularly shaped or overlapping clusters. It is also sensitive to the initial random assignment of cluster centers, which can result in different clustering results.

It's important to note that the K-Means algorithm is an unsupervised learning algorithm, meaning it doesn't require labeled data. It is solely based on the patterns and structure within the input data.

- **Step-by-step guide on how to use the K-Means algorithm for clustering:**

Step 1: Import Libraries

Start by importing the necessary libraries in Python, including scikit-learn, which provides the K-Means algorithm implementation.

```
from sklearn.cluster import KMeans
```

Step 2: Prepare the Data

Load your dataset and preprocess it as needed. Ensure that you have numerical features, as K-Means operates on numerical data.

Step 3: Initialize and Fit the K-Means Model

Create an instance of the KMeans class and specify the desired number of clusters (K). Then, fit the model to your dataset.

```
k = 3  # Number of clusters
kmeans = KMeans(n_clusters=k)
kmeans.fit(X)
```

Step 4: Obtain Cluster Assignments

After fitting the model, you can obtain the cluster assignments for each data point in the dataset.

```
cluster_labels = kmeans.labels_
```

Step 5: Interpret Results

You can analyze the obtained cluster assignments to gain insights from the data. For example, you can examine the distribution of data points across clusters or visualize the clusters in a scatter plot.

```
# Example: Counting the number of data points in each cluster
import numpy as np
unique_labels, counts = np.unique(cluster_labels, return_counts=True)
for label, count in zip(unique_labels, counts):
    print("Cluster {}: {} data points".format(label, count))
```

Step 6: Visualize Clusters (Optional)

If your data has two or three features, you can visualize the clusters in a scatter plot to gain a better understanding of the clustering results.

```
# Example: Visualizing clusters using matplotlib
import matplotlib.pyplot as plt
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("K-Means Clustering")
plt.show()
```

Note that the choice of K, the number of clusters, is a critical decision that can impact the clustering results. It often requires experimentation and domain knowledge.