1)Explain the basic concept of clustering and give examples of applications where clustering is useful.

Ans- Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing. Clustering can also help marketers discover distinct groups in their customer base. And they can characterize their customer groups based on the purchasing patterns.

Clustering technique is used in various applications such as market research and customer segmentation, biological data and medical imaging, search result clustering, recommendation engine, pattern recognition, social network analysis, image processing, etc.

2) What is DBSCAN and how does it differ from other clustering algorithms such as k-means and hierarchical clustering?

Ans- DBSCAN can identify clusters in a large spatial dataset by looking at the local density of corresponding elements. The advantage of the DBSCAN algorithm over the K-Means algorithm, is that the DBSCAN can determine which data points are noise or outliers.

DBSCAN stands for density-based spatial clustering of applications with noise. It is able to find arbitrary shaped clusters and clusters with noise (i.e. outliers). The main idea behind DBSCAN is that a point belongs to a cluster if it is close to many points from that cluster.

3) How do you determine the optimal values for the epsilon and minimum points parameters in DBSCAN clustering?

Ans- Step 1: Calculate the average distance between each point in the data set and its 20 nearest neighbors (my selected MinPts value). Step 2: Sort distance values by ascending value and plot. The ideal value for ε will be equal to the distance value at the “crook of the elbow”, or the point of maximum curvature.

In layman's terms, we find a suitable value for epsilon by calculating the distance to the nearest n points for each point, sorting and plotting the results. Then we look to see where the change is most pronounced (think of the angle between your arm and forearm) and select that as epsilon.

4) How does DBSCAN clustering handle outliers in a dataset?

Ans- Here SK-Learn library DBSCAN comes to the rescue to allow us to handle outliers for the Multi-variate datasets. DBSCAN considers two main parameters (as mentioned below) to form a cluster with the nearest data point and based on the high or low-density region, it detects Inliers or outliers.

5) How does DBSCAN clustering differ from k-means clustering?

Ans-

| **K-Means** | **DBSCAN** |
| --- | --- |
| K-means generally clusters all the objects. | DBSCAN discards objects that it defines as noise. |
| K-means needs a prototype-based concept of a cluster. | DBSCAN needs a density-based concept. |
| K-means has difficulty with non-globular clusters and clusters of multiple sizes. | DBSCAN is used to handle clusters of multiple sizes and structures and is not powerfully influenced by noise or outliers. |
| K-means can be used for data that has a clear centroid, including a mean or median. | DBSCAN needed that its definition of density, which depends on the traditional Euclidean concept of density, be significant for the data. |
| K-means can be used to sparse, high dimensional data, including file data. | DBSCAN generally implements poorly for such information because the traditional Euclidean definition of density does not operate well for high dimensional data. |
| The basic K-means algorithm is similar to a statistical clustering approach (mixture models) that consider all clusters come from spherical Gaussian distributions with several means but the equal covariance matrix. | DIISCAN creates no assumption about the distribution of the record. |

6) Can DBSCAN clustering be applied to datasets with high dimensional feature spaces? If so, what are

some potential challenges?

Ans- DBSCAN is a typically used clustering algorithm due to its clustering ability for arbitrarily-shaped clusters and its robustness to outliers. Generally, the complexity of DBSCAN is O(n^2) in the worst case, and it practically becomes more severe in higher dimension.

7) How does DBSCAN clustering handle clusters with varying densities?

Ans- DBSCAN is a density-based clustering algorithm that works on the assumption that clusters are dense regions in space separated by regions of lower density. It groups 'densely grouped' data points into a single cluster.

8) What are some common evaluation metrics used to assess the quality of DBSCAN clustering results?

Ans- DBSCAN has two parameters. The first is ε, epsilon (“esp”), which defines the maximum distance allowed between two points within the same cluster. The second is minimum samples (“MinPts”), which defines the minimum number of data points required to form a distinct cluster.

9) Can DBSCAN clustering be used for semi-supervised learning tasks?

Ans- Density-based spatial clustering of applications with noise (DBSCAN) is an unsupervised clustering ML algorithm. Unsupervised in the sense that it does not use pre-labeled targets to cluster the data points. Clustering in the sense that it attempts to group similar data points into artificial groups or clusters.

10) How does DBSCAN clustering handle datasets with noise or missing values?

Ans- Clustering algorithms can identify groups in large data sets, such as star catalogs and hyperspectral images. In general, clustering methods cannot analyze items that have missing data values. Common solutions either fill in the missing values (imputation) or ignore the missing data (marginalization).

11) Implement the DBSCAN algorithm using a python programming language, and apply it to a sample dataset. Discuss the clustering results and interpret the meaning of the obtained clusters.

Ans- import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import DBSCAN

from sklearn import metrics

from sklearn.datasets.samples\_generator import make\_blobs

from sklearn.preprocessing import StandardScaler

from sklearn import datasets

# Load data in X

X, y\_true = make\_blobs(n\_samples=300, centers=4,

cluster\_std=0.50, random\_state=0)

db = DBSCAN(eps=0.3, min\_samples=10).fit(X)

core\_samples\_mask = np.zeros\_like(db.labels\_, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = True

labels = db.labels\_

# Number of clusters in labels, ignoring noise if present.

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)

print(labels)

# Plot result

# Black removed and is used for noise instead.

unique\_labels = set(labels)

colors = ['y', 'b', 'g', 'r']

print(colors)

for k, col in zip(unique\_labels, colors):

if k == -1:

# Black used for noise.

col = 'k'

class\_member\_mask = (labels == k)

xy = X[class\_member\_mask & core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,

markeredgecolor='k',

markersize=6)

xy = X[class\_member\_mask & ~core\_samples\_mask]

plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=col,

markeredgecolor='k',

markersize=6)

plt.title('number of clusters: %d' % n\_clusters\_)

plt.show()

#evaluation metrics

sc = metrics.silhouette\_score(X, labels)

print("Silhouette Coefficient:%0.2f"%sc)

ari = adjusted\_rand\_score(y\_true, labels)

print("Adjusted Rand Index: %0.2f"%ari)