Experiment 5 - Unsupervised Learning

In Experiment 5, we delve into the domain of unsupervised learning, which involves machine learning algorithms used to analyze and cluster unlabeled datasets. Unlike supervised learning, which we explored in the previous experiment, unsupervised learning seeks to identify hidden patterns within the data through grouping and categorization, without the need for human intervention. The applications of unsupervised learning are extensive and span across various real-world scenarios, including image segmentation and recognition, natural language processing, text summarizing, biological data analysis, protein clustering, and data feature interpretation.

To carry out this experiment, we will employ Python in combination with the Scikit-learn library to implement different unsupervised learning techniques. Specifically, we will explore distance-based clustering methods like k-means clustering and hierarchical clustering, as well as density-based clustering methods such as DBSCAN. Additionally, we will utilize dimensionality reduction techniques like PCA to reduce the complexity of the data. Throughout the experiment, we will assess the performance of these models using appropriate evaluation measures to gauge their effectiveness and accuracy.

1.1 Clustering

Clustering is a fundamental technique in the field of unsupervised learning, where the primary objective is to group a set of input data points into distinct clusters. Unlike classification tasks, in clustering, the groups or clusters are not known in advance and are determined based on the underlying structure and similarity in the data.

Cluster analysis involves the process of assigning observations to subsets, known as clusters, in such a way that items within the same cluster exhibit a certain level of similarity, as defined by specific criteria. Simultaneously, items from different clusters are expected to be dissimilar. Various clustering techniques exist, each making different assumptions about the data's structure. These assumptions are typically defined using similarity metrics and evaluated based on factors like internal compactness (how similar members within a cluster are) and separation between different clusters.

Some clustering methods rely on estimating data density and establishing connectivity through graph-based approaches. The ultimate goal is to create meaningful and coherent groupings from the input data, which can then be further analyzed for various purposes.

Overall, clustering plays a crucial role in statistical data analysis and serves as a valuable tool for understanding the inherent patterns and structures within datasets in the absence of labeled information.

1.1.1 K-means

K-means is a popular distance-based clustering method used to partition a set of observations into k clusters, with each observation being assigned to the cluster whose mean (also called cluster center or centroid) is closest to it.

The k-means algorithm aims to minimize the within-cluster variances, which are calculated as the squared Euclidean distances between the data points and their respective cluster centroids. By minimizing these variances, k-means effectively groups similar points together while keeping different clusters well-separated.

However, it's essential to note that k-means is sensitive to outliers and can be influenced by the initial placement of cluster centroids, which can result in suboptimal solutions. As you mentioned, for scenarios where the Euclidean distances are not the ideal measure of similarity, other distance metrics like Manhattan distance or Mahalanobis distance can be used, or alternative clustering algorithms like k-medians or k-medoids can be employed.

In contrast to k-means, k-medians minimizes the sum of absolute distances (L1 distance) between data points and their respective cluster medians, while k-medoids minimizes the sum of dissimilarities (using any chosen distance metric) between data points and their medoids, which are actual data points within the cluster. These alternative clustering methods can be more robust when dealing with datasets containing outliers or when the Euclidean distance is not appropriate for the given data distribution.

1.1.2 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular clustering algorithm used in unsupervised machine learning. It is particularly effective in identifying clusters of arbitrary shapes and handling noisy data. The key idea behind DBSCAN is to group together data points that are densely located, while also identifying outliers as points that are in low-density regions.

A brief overview of how DBSCAN works:

1. **Density-Based Clustering:** DBSCAN operates based on the notion of density. It defines two important parameters: "epsilon" (ϵ) and "minimum points" ($MinPts$). Epsilon represents the maximum distance that defines the neighborhood of a data point, and MinPts sets the minimum number of points required within that epsilon neighborhood to form a cluster.

2. **Core Points:** A data point is considered a core point if it has at least $MinPts$ data points (including itself) within its epsilon neighborhood. Core points are

the central elements of clusters.

3. **Directly Reachable:** Two core points are said to be directly reachable if they are within each other's epsilon neighborhood.

4. **Density-Reachable:** A data point is density-reachable from a core point if there is a chain of core points such that each consecutive core point is directly reachable from the previous one.

5. **Border Points:** Data points that are not core points but have at least one core point in their epsilon neighborhood are called border points. Border points can be part of a cluster but do not contribute to forming the core structure of the cluster.

6. **Noise Points:** Data points that are neither core points nor border points are considered noise points or outliers.

DBSCAN proceeds by iterating through the dataset and identifying core points and their corresponding density-reachable points to form clusters. The algorithm does not require the number of clusters to be specified in advance, making it very flexible for discovering clusters of various shapes and sizes.

One of the key advantages of DBSCAN is its ability to handle datasets with varying cluster densities and irregularly shaped clusters, as it uses local density information to form clusters. However, setting appropriate values for ϵ and $MinPts$ can be challenging and might impact the quality of the clusters obtained.

Overall, DBSCAN is a powerful and widely used clustering method for datasets with varying densities and outliers. It has applications in various fields, such as image processing, spatial data analysis, and anomaly detection.

1.1.3 Gaussian Mixture Model (GMM)

Gaussian Mixture Model (GMM) is a probabilistic clustering algorithm used to model data as a mixture of multiple Gaussian distributions. It is a powerful unsupervised learning technique that assumes the data is generated from a combination of several Gaussian distributions, each representing a distinct cluster.

Here's an overview of how the Gaussian Mixture Model (GMM) clustering method works:

1. **Initialization:** GMM starts with an initial guess for the parameters of the Gaussian distributions. These parameters include the mean, covariance matrix, and the mixing coefficients (weights) for each Gaussian component.

2. **Expectation-Maximization (EM) Algorithm:** GMM utilizes the EM algorithm to iteratively update the parameters and improve the model's fit to the data.

- **Expectation Step (E-step):** In this step, GMM calculates the probability (responsibility) that each data point belongs to each Gaussian component based on the current parameter estimates.
- **Maximization Step (M-step):** In this step, GMM updates the parameters by maximizing the likelihood of the data, given the computed probabilities from the E-step.

3. **Convergence:** The EM algorithm iterates between the E-step and M-step until the model converges, or a predefined stopping criterion is met. Convergence occurs when the parameters no longer change significantly between iterations.

4. **Cluster Assignment:** Once the GMM model converges, each data point is assigned to the Gaussian component that has the highest probability (responsibility) for that point.

GMM allows for flexible cluster shapes and can model data distributions that are not well-suited to traditional distance-based clustering methods like k-means. It can also handle data with overlapping clusters, as each Gaussian component captures a portion of the data points from different clusters.

One of the key advantages of GMM is its probabilistic nature, which provides a soft assignment of data points to clusters. Instead of hard assignments, where data points belong exclusively to one cluster, GMM assigns probabilities, indicating the likelihood of each data point's association with different clusters. This makes GMM more robust in the presence of noise and uncertainty in the data.

GMM has various applications, including image segmentation, density estimation, anomaly detection, and data generation. However, it's essential to note that GMM's performance may be sensitive to the number of Gaussian components and the initial parameter estimation, which can require careful consideration and tuning.

1.2 Clustering Algorithms Evaluation Criteria

When evaluating the performance of clustering algorithms, several criteria can be used to assess the quality of the clustering results. The choice of evaluation criteria depends on the nature of the data and the specific objectives of the clustering task. Here are some common clustering evaluation criteria:

1. Silhouette Score: The silhouette score measures the compactness and separation of clusters. It ranges from -1 to 1, where a higher score indicates better-defined clusters and a more appropriate clustering result.

2. Davies-Bouldin Index: This index quantifies the average similarity between each cluster and its most similar cluster, providing a measure of cluster separation. Lower values indicate better clustering.

3. Dunn Index: The Dunn index assesses the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance, aiming to maximize the distance between clusters and minimize the distance within clusters.

4. Calinski-Harabasz Index (Variance Ratio Criterion): This index evaluates the ratio of the sum of between-cluster dispersion to the sum of within-cluster dispersion. Higher values represent better clustering results.

5. Adjusted Rand Index (ARI): The ARI measures the similarity between the true labels and the predicted cluster labels, accounting for chance agreement. It ranges from -1 to 1, where 1 indicates perfect clustering agreement.

6. Normalized Mutual Information (NMI): NMI measures the mutual information between the true labels and the predicted cluster labels, normalized to a value between 0 and 1.

7. Homogeneity, Completeness, and V-measure: These three metrics evaluate the purity of clusters. Homogeneity measures whether all data points in a cluster belong to the same true class. Completeness assesses whether all data points of a given true class are assigned to the same cluster. V-measure is the harmonic mean of homogeneity and completeness.

8. Fowlkes-Mallows Index: This index calculates the geometric mean of precision and recall for clustering evaluation.

9. Rand Index: The Rand index measures the similarity between the true and predicted clustering by considering the number of true positive and true negative pairs.

10. Jaccard Index: The Jaccard index measures the similarity between the true and predicted clustering based on the number of pairs that are either in the same cluster or in different clusters.

It is important to note that there is no one-size-fits-all evaluation metric, and the appropriate criterion should be chosen based on the specific characteristics of the data and the goals of the clustering task. Moreover, clustering evaluation is often subjective, and a combination of multiple metrics can provide a more comprehensive assessment of clustering quality.

1.3 Procedures

1.3.1 Data Generation

We use *make_blobs* to create 3 synthetic clusters in 2D.

Listing 1.1: Genertaing clusters/blobs

```
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

# Generate a Gaussian 2D dataset (blobs)
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(
    n_samples=750, centers=centers, cluster_std=0.4, random_state=0)
X = StandardScaler().fit_transform(X)
```

We can visualize the resulting data:

Listing 1.2: Data/blobs visualization

```
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(12,6))
colormap=np.array(['red','lime','black'])
# Plot the blobs without lables
plt.subplot(1,2,1)
plt.scatter(X[:, 0], X[:, 1])
plt.title('Blobs')
#plt.show()

# Plot the blobs with lables (Ground Truth (GT)). We will use it for
    clustering results evaluation
plt.subplot(1,2,2)
plt.scatter(X[:, 0], X[:, 1], c=colormap[labels_true],s=40)
plt.title('Blobs with lables')
#plt.show()
```

You should get a figure similar to Figure 1.1

1.3.2 Compute KMeans

The first algorithm to be leveraged on the generated data (i.e., blobs) is KMeans. Try different values of K

Listing 1.3: Data clustering using KMeans algorithm

```
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Use K = 3
model=KMeans(n_clusters=3)
model.fit(X)
predY=np.choose(model.labels_,[0,1,2]).astype(np.int64)
```

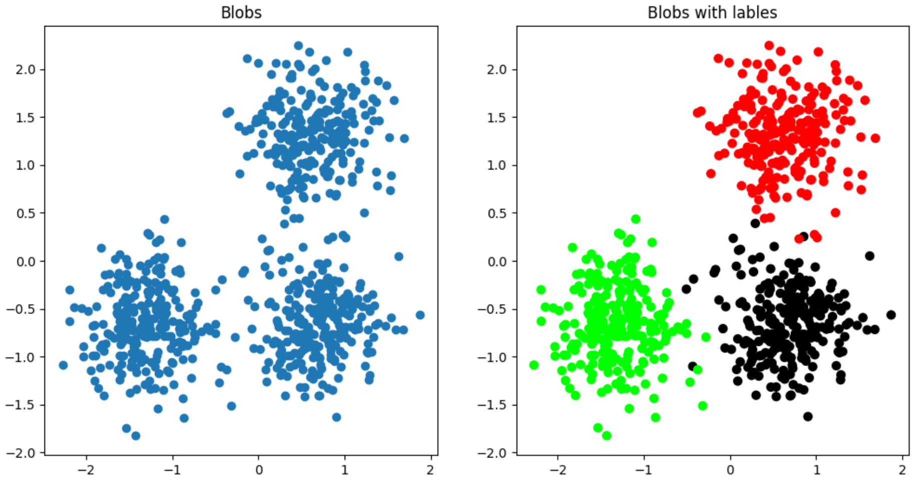

Figure 1.1: The generated blobs

Visualize the point membership after KMeans clustering algorithm. Compare the results with ground truth (GT) point memberships.

Listing 1.4: Data/blobs visualization after KMeans clustering

```
plt.figure(figsize=(12,6))
colormap=np.array(['red','lime','black'])
# Plot the blobs before the clustering
plt.subplot(1,2,1)
plt.scatter(X[:, 0], X[:, 1], c=colormap[labels_true],s=40)
plt.title('Blobs with GT lables')

# Plot the blobs after KMeans clustering
plt.subplot(1,2,2)
plt.scatter(X[:, 0], X[:, 1], c=colormap[predY],s=40)
plt.title('Blobs after KMeans clustering')


# We will also get the coordinates of the cluster centers using KMeans
    .cluster_centers_ and save it as k_means_cluster_centers.
k_means_cluster_centers = model.cluster_centers_

# define the centroid, or cluster center.
cluster_centers = k_means_cluster_centers
#print(cluster_center)
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='*',
    c='orange',s=100)
```

You should get a figure similar to Figure 1.2

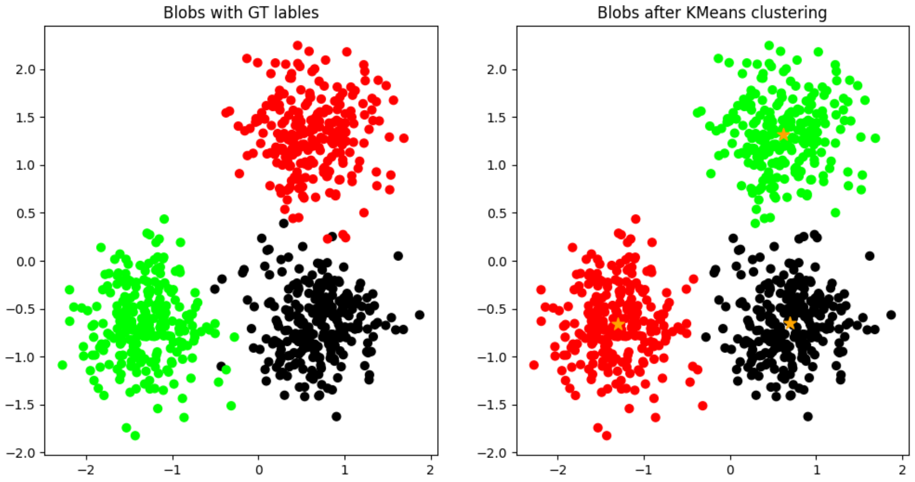

Figure 1.2: KMeans clustering results when using $K = 3$

1.3.3 Compute DBSCAN

The second algorithm to be leveraged on the generated data (i.e., blobs) is DBSCAN. Try different values of ϵ and *MinPts* (*min_samples*)

One can access the labels assigned by DBSCAN using the *labels_* attribute. Noisy samples are given the label math:-1.

Listing 1.5: Data clustering using DBSCAN algorithm

```
import numpy as np

from sklearn import metrics
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.3, min_samples=10).fit(X) # Params for blobs data
# db = DBSCAN(eps=0.15, min_samples=5).fit(X) # Params for two moons data
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
```

Visualize the point membership after DBSCAN clustering algorithm. Compare the results with GT point memberships.

Listing 1.6: Data/blobs visualization after DBSCAN clustering

```
plt.figure(figsize=(12,6))
# Plot the blobs before the clustering
```

```
plt.subplot(1,2,1)
plt.scatter(X[:, 0], X[:, 1], c=colormap[labels_true],s=40)
plt.title('Blobs with GT lables')

# Plot the blobs after DBSCAN clustering. I used a different
   visualization/plot to show the core points and the noisy points in
   different color and size
plt.subplot(1,2,2)
unique_labels = set(labels)
core_samples_mask = np.zeros_like(labels, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True

colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1,
   len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = labels == k

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=14,
    )

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.title(f"DBSCAN estimated number of clusters: {n_clusters_}")
```

You should get a figure similar to Figure 1.3

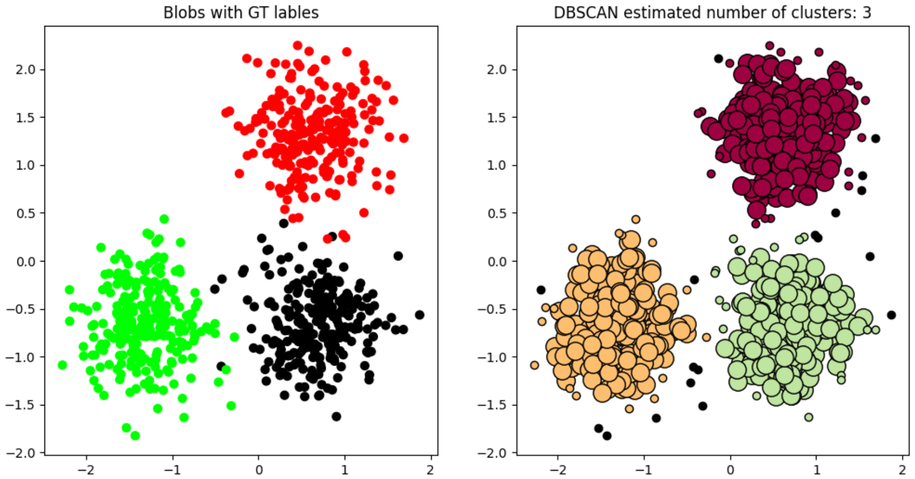

Figure 1.3: KMeans clustering results when using $K = 3$

1.3.4 Compute Gaussian Mixture Model (GMM)

The third algorithm to be leveraged on the generated data (i.e., blobs) is Gaussian mixture models (GMMs). Try different values of *n_components*.

Listing 1.7: Data clustering using GMM algorithm

```
from sklearn.mixture import GaussianMixture
gmm=GaussianMixture(n_components=3)
gmm.fit(X)
y_cluster_gmm=gmm.predict(X)
```

Visualize the point membership after GMM clustering algorithm. Compare the results with GT point memberships.

Listing 1.8: Data/blobs visualization after GMM clustering

```
plt.figure(figsize=(12,6))
colormap=np.array(['red','lime','black'])
# Plot the blobs before the clustering
plt.subplot(1,2,1)
plt.scatter(X[:, 0], X[:, 1], c=colormap[labels_true],s=40)
plt.title('Blobs with GT lables')

# Plot the blobs after KMeans clustering
plt.subplot(1,2,2)
plt.scatter(X[:, 0], X[:, 1], c=colormap[y_cluster_gmm],s=40)
plt.title('Blobs after GMMs clustering')
```

You should get a figure similar to Figure 1.4

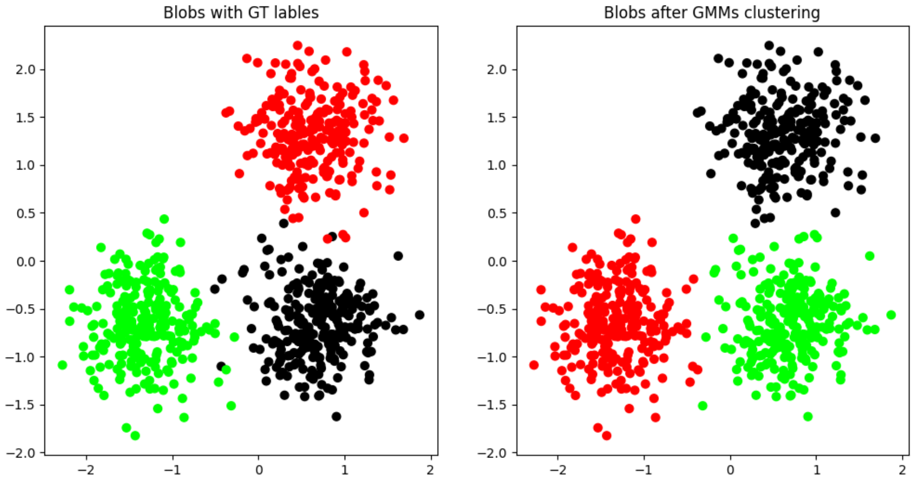

Figure 1.4: GMM clustering results when using $n_components = 3$

1.4 Quantitative results comparison of the clustering methods using evaluation measures

In the previous sections, we qualitatively assessed the performance of the clustering methods by visually comparing their results with GT data. However, we need quantification to determine the performance of the various clustering algorithms. The quantitative analysis requires GT labels. The below code shows the performance comparison of the three clustering methods adopted in this experiment. We selected the well-known evaluation measures. You can try the other evaluation criteria by yourself.

Listing 1.9: Quantitative analysis of the clustering methods

```
print(f"---------------------------")
print(f"DBSCAN Evaluation measures\n")

print(f"Homogeneity: {metrics.homogeneity_score(labels_true, labels):.3f}")
print(f"Completeness: {metrics.completeness_score(labels_true,
    labels):.3f}")
print(f"V-measure: {metrics.v_measure_score(labels_true, labels):.3f}")
print(f"Adjusted Rand Index: {metrics.adjusted_rand_score(labels_true,
    labels):.3f}")
print(
    "Adjusted Mutual Information:"
    f" {metrics.adjusted_mutual_info_score(labels_true, labels):.3f}"
)
print(f"Silhouette Coefficient: {metrics.silhouette_score(X, labels):.3f}")

print(f"\n---------------------------")
print(f"K-Means Evaluation measures\n")
print(f"Homogeneity: {metrics.homogeneity_score(labels_true, predY):.3f}")
print(f"Completeness: {metrics.completeness_score(labels_true,
   predY):.3f}")
print(f"V-measure: {metrics.v_measure_score(labels_true, predY):.3f}")
print(f"Adjusted Rand Index: {metrics.adjusted_rand_score(labels_true,
   predY):.3f}")
print(
    "Adjusted Mutual Information:"
    f" {metrics.adjusted_mutual_info_score(labels_true, predY):.3f}"
)
print(f"Silhouette Coefficient: {metrics.silhouette_score(X, predY):.3f}")

print(f"\n---------------------------")
print(f"GMMs Evaluation measures\n")
print(f"Homogeneity: {metrics.homogeneity_score(labels_true,
   y_cluster_gmm):.3f}")
print(f"Completeness: {metrics.completeness_score(labels_true,
   y_cluster_gmm):.3f}")
print(f"V-measure: {metrics.v_measure_score(labels_true,
   y_cluster_gmm):.3f}")
print(f"Adjusted Rand Index: {metrics.adjusted_rand_score(labels_true,
   y_cluster_gmm):.3f}")
print(
    "Adjusted Mutual Information:"
    f" {metrics.adjusted_mutual_info_score(labels_true,
       y_cluster_gmm):.3f}"
)
print(f"Silhouette Coefficient: {metrics.silhouette_score(X,
   y_cluster_gmm):.3f}")
```

1.5 To DO

1. Repeat the above procedure by generating two moons of data. Compare the performance of the clustering algorithms, qualitatively and quantitatively, when applied to the blobs dataset and two moons datasets. Use the below code to generate the two moons dataset.

   ```
   # Generate a non Gaussian 2D dataset (two moons)
   from sklearn.datasets import make_moons
   X, labels_true = make_moons(n_samples=500, noise=0.1)
   ```

   You need to set $K = 2$ for KMeans and $\epsilon = 0.15$, $min\_samples = 5$ for the DBSCAN algorithm.

2. K-means attempts to minimize the total squared error, while k-medoids minimize the sum of dissimilarities between points labeled to be in a cluster and

a point designated as the center of that cluster. In contrast to the k -means algorithm, k -medoids choose data points as centers (medoids or exemplars). Compare, qualitatively and quantitatively, the results of k-means and k-medoid when applied to the blobs dataset.

3. Write a Python program that takes your own color image as input and segments the image into a) $K = 2$, b) $K = 5$ and c) $K = 10$ clusters using K-mean clustering. Display the resulting images. Discuss your observation about both images. You might need to hand this ToDo in the next lab.