Artificial Intelligence
Chapter 4 - Unsupervised Learning Algorithms

Dr. Adnan Ismail Al-Sulaifanie

Department of Electrical and Computer Engineering
University of Duhok, Iraq

Outline

- ➤ Concept of clustering.
- ➤ List common clustering algorithms - k-mean, mean shift, DBSCAN, ...

References

1. Aurélien Géron, **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow**, 2019
2. Christopher M. Bishop, **Pattern Recognition and Machine Learning**
3. Andreas C. Müller, **Introduction to Machine Learning with Python: A Guide for Data Scientists**,
4. Chris Albon, **Machine Learning with Python Cookbook**, 2018
5. Ashwin Pajankar, **Hands-on Machine Learning with Python**, 2022
6. scikit-learn user guide
7. Sebastian Raschka, **Machine Learning with PyTorch and Scikit-Learn**, 2022

1 Unsupervised Learning Algorithms

➤ Unsupervised learning is a type of machine learning that deals with unlabeled data.

➤ The goal is to discover hidden patterns, structures, or relationships in the data without any prior knowledge of the output.

➤ Algorithms learn by themselves, without any guidance.

➤ Types of Unsupervised Learning Algorithms: clustering and dimensionality reduction.

➤ Common clustering algorithms: K-means, Mean Shift, DBSCAN, OPTICS, ...

➤ Applications of Unsupervised Learning

1. Customer Segmentation – Grouping customers based on their behavior or preferences.
2. Anomaly Detection – Identifying unusual patterns or outliers in data.
3. Recommender Systems – Suggesting products or content to users based on their past behavior.
4. Image and Speech Recognition – Extracting features or patterns from images or audio.
5. Document Clustering – Organizing documents into topics or categories.

➤ Advantages of Unsupervised Learning

1. Can handle unlabeled data, which is often more abundant and cheaper to obtain.
2. Discovers hidden patterns that may not be apparent to humans.
3. Provides insights for exploratory data analysis.

➤ Disadvantages of Unsupervised Learning

1. Results can be difficult to evaluate, as there are no ground truth labels.
2. May require more domain knowledge to interpret the results.
3. Can be computationally expensive for large datasets.

2 K-means Clustering

➤ K-means is used for partitioning a given dataset into K clusters, where K is a user-defined number.

➤ The main goal is to group similar data points together and assign them to the same cluster, while maximizing the dissimilarity between different clusters.

➤ Procedure:

1. **Initialization –** Choose the number of clusters (K). These points can be chosen randomly or using a specific seeding technique.

2. **Assignment step –** each data point is assigned to the cluster whose centroid is closest to it using a distance metric like Euclidean distance.
3. **Update Step –** Once all data points are assigned to clusters, each centroid is recalculated as the mean of all the data points assigned to that cluster (find the new center of each cluster).
4. **Iteration –** Steps 2 and 3 are repeated iteratively until it converge – assignments of data points to clusters stop changing or change very minimally, and the centroids become stable.
5. **Final Result –** the data points will be partitioned into K clusters, and the centroids of these clusters will represent the final results of the K-means algorithm.

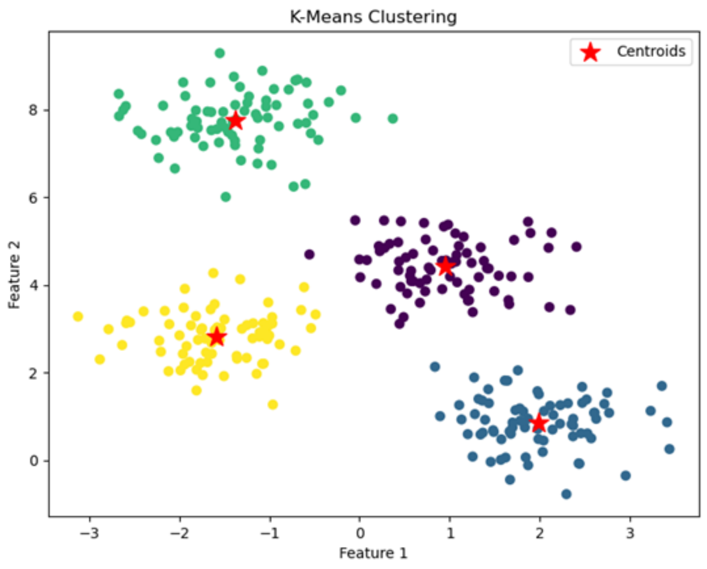
- K-means is sensitive to the initial choice of centroids, and it may converge to different solutions depending on the starting points. To mitigate this issue, K-means is often run multiple times with different initializations, and the best result is selected based on a predefined evaluation metric (e.g., within-cluster sum of squares, also known as the inertia).
- Advantages of K-means clustering:
  1. Relatively easy to understand and implement.
  2. Efficient and scalable, making it suitable for large datasets.
  3. Works well with data that has clear, well-defined clusters.
- Limitations of K-means clustering:
  1. K needs to be predefined, which may not always be known in advance.
  2. It is sensitive to the initial centroids.
  3. May not work well with clusters of different sizes, shapes, or densities.

```
from sklearn import cluster

model = sklearn.cluster.KMeans(n_clusters = 3)

model.fit(features)

labels = model.labels_

print(model.cluster_centers_)      print quantization vectors

model.predict(xnew)
```

Example 1 -

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
```

```
X, y = make_blobs(n_samples = 300, centers = 4, cluster_std = 0.60, random_state = 0)

kmeans = KMeans(n_clusters = 4, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
kmeans.fit(X)

labels = kmeans.labels_
centroids = kmeans.cluster_centers_

plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c = labels, cmap='viridis')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s = 200, color='red', label='Centroids')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



3 Mean Shift Clustering

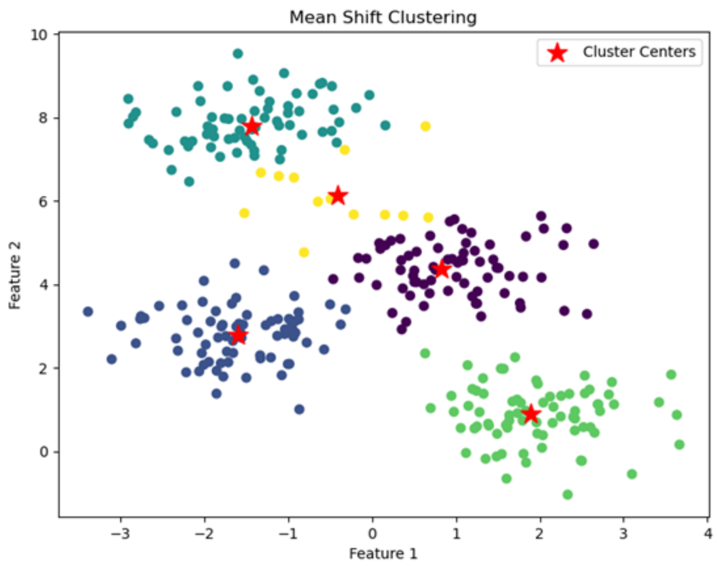
- Mean Shift is a non-parametric clustering algorithm that discovers "blobs" or areas of high density in a dataset.
- It doesn't assume clusters have a particular shape, and it can automatically determine the number of clusters.
- Algorithm
  1. **Initialization –** Start with each data point as a potential cluster center.

2. **Kernel Density Estimation –** Define a window around each data point using a kernel function (often a Gaussian). Calculate the ”mean shift” vector, which points towards the direction of the highest density of neighboring points.
3. **Shifting –** Move the center point in the direction of the mean shift vector.
4. **Iteration –** Repeat steps 2 and 3 until the center points converge (i.e., they don't move significantly).
5. **Cluster Assignment –** Data points that converge to the same center are assigned to the same cluster.

- Key Concepts
  - **Kernel –** A function that defines the neighborhood around a data point and weights the importance of other points within that neighborhood. A Gaussian kernel is commonly used.
  - **Bandwidth –** A parameter that determines the size of the window. A larger bandwidth leads to smoother density estimates and fewer clusters, while a smaller bandwidth can lead to more clusters.
  - **Density Gradient –** The mean shift vector always points in the direction of the steepest increase in density.
- Advantages
  - Doesn't require specifying the number of clusters beforehand.
  - Can handle arbitrarily shaped clusters.
  - Robust to outliers.
  - Conceptually simple and easy to understand.
- Disadvantages
  - The choice of bandwidth can significantly affect the results.
  - Computationally expensive, especially for large datasets.
  - Performance degrades in high-dimensional spaces.
- Applications
  - Image Segmentation – Grouping pixels with similar colors or textures.
  - Object Tracking – Tracking the movement of objects in a video.
  - Clustering – Finding clusters in data without prior knowledge of the number or shape of the clusters.
  - Computer Vision – Applications like background subtraction and feature tracking.

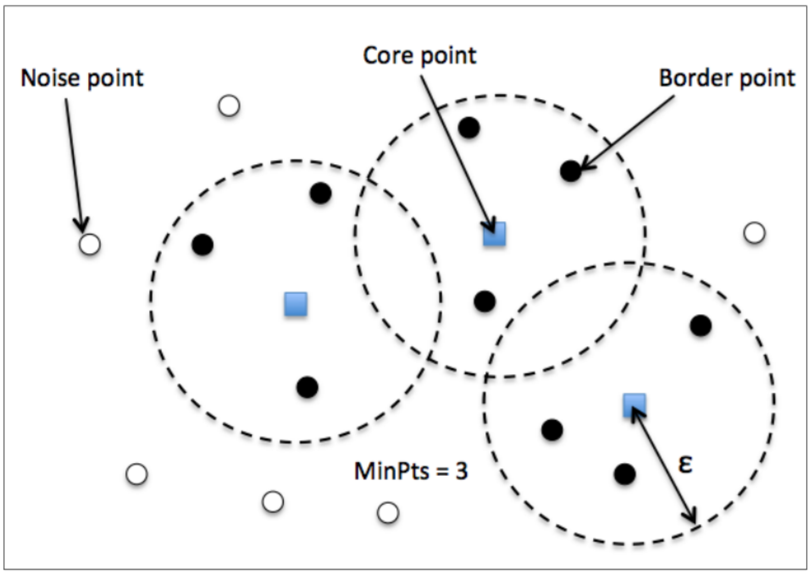
Example 2 -

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.7, random_state=0)

bandwidth = estimate_bandwidth(X, quantile=0.2)

ms = MeanShift(bandwidth=bandwidth)
ms.fit(X)
labels = ms.labels_
cluster_centers = ms.cluster_centers_

n_clusters_ = len(np.unique(labels))
print("Number of estimated clusters:", n_clusters_)

plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.scatter(cluster_centers[:, 0], cluster_centers[:, 1], marker='*', s=200, color='red', label='Cluster
Centers')
plt.title('Mean Shift Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



4 DBSCAN

- Density-Based Spatial Clustering of Applications with Noise.
- DBSCAN groups data points that are close to each other and have sufficient density into clusters, while also identifying outliers as noise points.
- DBSCAN is robust to noise and can identify clusters of arbitrary shapes and sizes. It does not require the number of clusters to be specified beforehand.
- DBSCAN requires two user-defined parameters:
  1. ε (epsilon): The radius defining the neighborhood around a point
  2. $MinPts$: Minimum number of points required to form a dense region
- The algorithm categorizes data points into three categories:
  - **Core points –** Points that have at least $MinPts$ points within a distance ε
  - **Border points –** Points that are within distance ε of a core point but have fewer than $MinPts$ points in their ε-neighborhood
  - **Noise points (outliers) –** Points that are neither core nor border points



- Algorithm Steps:
  1. Select an arbitrary data point and finding its epsilon neighborhood (points within distance ε).
  2. Define core point – If this point has at least $MinPts$ points within its epsilon neighborhood, it is classified as a core point.
  3. The process continues until all core and border points have been added to the cluster. Then, a new core point (not previously assigned to any cluster) is selected, and the process repeats until all data points are processed.

4. Any remaining unassigned data points (not part of any cluster) are marked as noise points.

➤ Advantages

1. No need to specify the number of clusters beforehand
2. Can find arbitrarily shaped clusters
3. Robust to outliers
4. Works well with spatial data

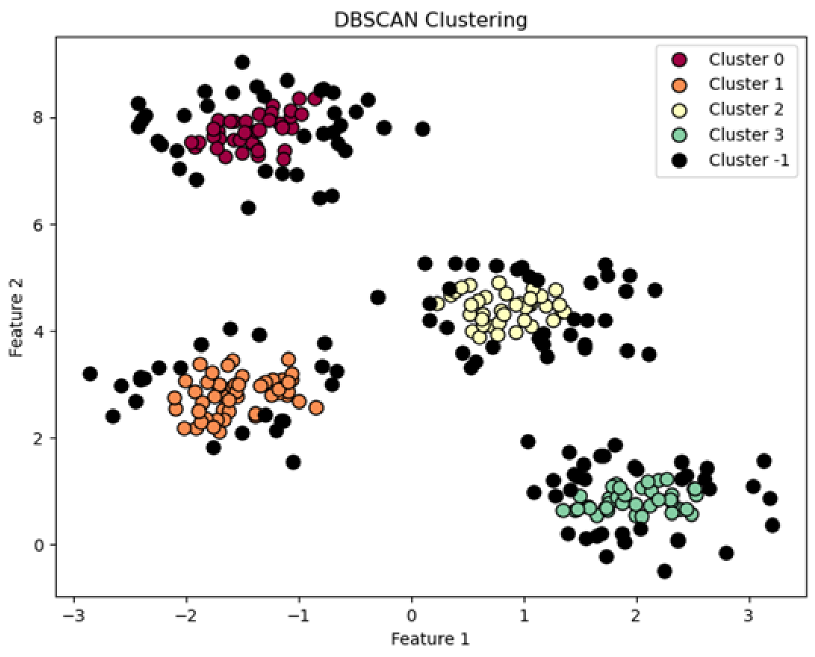
➤ Disadvantages

1. Sensitive to parameter selection (ε and $MinPts$)
2. Less efficient for very large datasets
3. Difficulty handling high-dimensional data due to the curse of dimensionality
4. Can be computationally expensive for very large datasets.

➤ Applications

- ➢ Image segmentation – Grouping pixels based on color or intensity.
- ➢ Anomaly detection – Identifying unusual patterns in data.
- ➢ Geospatial data analysis – Identifying clusters of locations with similar characteristics.
- ➢ Customer segmentation – Grouping customers with similar behaviors.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.5, random_state=0)

db = DBSCAN(eps=0.3, min_samples=10)
db.fit(X)
labels = db.labels_

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
print(”Estimated number of clusters:”, n_clusters_) 4

plt.figure(figsize=(8, 6)) Black color is used for noise points.
unique_labels = set(labels)
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:
col = [0, 0, 0, 1]
class_member_mask = (labels == k)
xy = X[class_member_mask]
plt.plot(xy[:, 0], xy[:, 1], ’o’, markerfacecolor=tuple(col),markeredgecolor=’k’, markersize=6,
label=f'Cluster k')

plt.title('DBSCAN Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```



OPTICS – can be viewed as a generalization of DBSCAN that addresses the issue of detecting meaningful clusters in data of varying density. While it's more powerful, it also typically requires more computational resources and produces more complex output that needs additional processing to extract the final clusters.