Mean Shift Clustering And K-Means Clustering

Maxime Eidelsberg, Simon Shen

Contents

1 Introduction

The project consists of studying two clustering algorithms: the K-means algorithm and the mean shift clustering algorithm.

In the fields of machine learning and data science, data clustering plays a crucial role in exploratory analysis and unsupervised classification. Clustering algorithms are useful for several reasons. One application of clustering algorithms is identifying patterns in unlabeled data to discover relationships between data points. Another application is image segmentation based on pixels, where clustering is performed using different pixels to recognize shapes in an image, for example.

Among the various available clustering techniques, the K-means and Mean Shift algorithms stand out for their popularity and effectiveness in various application contexts. K-means clustering, with its simplicity and fast execution, is widely used to divide a dataset into a predefined number of clusters, thereby optimizing the sum of intra-cluster squared distances. On the other hand, Mean Shift clustering, based on density estimation, automatically detects the number of clusters by focusing on density maxima in the feature space. This report aims to explore in detail the operation, advantages, and disadvantages of these two algorithms, highlighting their fundamental differences and respective application domains.

2 Clustering Algorithms

Clustering algorithms assign each observation to a cluster without considering a probability model describing the data. These algorithms partition the observations $i \in (1, \ldots, N)$ into K groups, called "clusters." The assignment of an observation to a cluster can be done using a function $k = C(i)$ that assigns observation i to cluster k. The goal is to find the best assignment $C^*(i)$ that minimizes a certain function. One approach is to directly provide a minimization function using algorithms. A function that assigns a point to its nearest cluster is:

$$W(C) = \sum_{k=1}^{K} \sum_{i:C(i)=k} \sum_{i':C(i')=k} d(x_i, x_{i'})$$

where $d(x_i, x_{i'})$ is the distance between the two observations i and i'. Minimizing this function helps find C^* such that the sum of distances between two elements of the same cluster is as small as possible.

This formula is derived from the total number of points [1]:

$$T = \sum_{k=1}^{K} \sum_{i:C(i)=k} \sum_{i':C(i')=k} d(x_i, x_{i'}) + \sum_{k=1}^{K} \sum_{i:C(i)=k} \sum_{i':C(i')\neq k} d(x_i, x_{i'})$$

The sum is split into two parts: one part corresponds to elements belonging to the same cluster, and the other sum corresponds to elements not belonging to the same clusters. Thus, T is constant. The goal here is to minimize the first part of the sum (or maximize the second part).

3 K-means Algorithm

3.1 General Overview

The K-means algorithm partitions N points into k groups called "clusters." This partitioning is done by minimizing a function: the sum of the squares of the distance between the points and the centroid of their respective clusters. The distance is thus:

$$d(x_i, x_{i'}) = \|x_i - x_{i'}\|^2$$

The value function becomes:

$$W(C) = \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k}\|x_i - x_{i'}\|^2$$

$$= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k}\left(\|x_i - \bar{x}_k - (x_{i'} - \bar{x}_k)\|^2\right)$$

$$= \frac{1}{2}\sum_{k=1}^{K}\sum_{C(i)=k}\sum_{C(i')=k}\left(\|x_i - \bar{x}_k\|^2 + \|x_{i'} - \bar{x}_k\|^2 - 2\langle x_i - \bar{x}_k, x_{i'} - \bar{x}_k\rangle\right)$$

$$= \frac{1}{2}\sum_{k=1}^{K}\left(N_k\sum_{C(i)=k}\|x_i - \bar{x}_k\|^2 + N_k\sum_{C(i')=k}\|x_{i'} - \bar{x}_k\|^2 - 2\langle\sum_{C(i')=k}(x_{i'} - \bar{x}_k), \sum_{C(i')=k}(x_{i'} - \bar{x}_k)\rangle\right)$$

$$= \sum_{k=1}^{K} N_k \sum_{C(i)=k}\|x_i - \bar{x}_k\|^2$$

where $N_k = \sum_{i=1}^{N}\text{card}(\{C(i) = k\})$.

The algorithm is initialized with k means m_i^1 at time $t = 1$.

Then, through an iterative two-step process, we aim to minimize this value function:

The first step (cluster assignment) assigns each point to the cluster whose centroid is closest, i.e., by finding the minimum distance between the cluster centroids and the point. Thus, at each time t, we can define k clusters:

$$C_i^t = \{x : \|x - m_i^t\|^2 \leq \|x - m_j^t\|^2 \ \forall j, 1 \leq j \leq k\} \quad \forall i, 1 \leq i \leq k$$

Next, the second step (centroid recalculation) updates the new mean of each cluster at time $t + 1$. For each cluster, the centroid is recalculated as the mean of the points assigned to that cluster. This recalculation aims to further reduce the total intra-cluster variation.

$$m_i^{(t+1)} = \frac{1}{|C_i^{(t)}|}\sum_{x \in C_i^{(t)}} x$$

These two steps are repeated until the cluster assignments no longer change or the decrease in total intra-cluster variation becomes negligible, indicating the algorithm's convergence.

There are two types of initialization for K-means: the Forgy method and the Random partition. The Forgy method assigns k centers randomly from the data list, while the Random partition assigns each point to a cluster and then takes the centroid of each cluster as the mean of its points. Initialization is a challenging aspect of the K-means algorithm because the final result depends on the initial conditions. "Alternatives to the k-means algorithm that find better clusterings" suggests that generally, for the K-means algorithm, the Random partition provides better minimization of the function, but they observe the opposite in their work.

3.2 Importance of Total Variation

The total intra-cluster variation (value function W) is a crucial criterion for evaluating the quality of clusters formed by the K-means algorithm. A lower W indicates more compact and distinct clusters, which is often desired in clustering applications. However, it is important to note that minimizing WCSS can sometimes lead to unevenly sized clusters or results sensitive to outliers.

However, depending on the initialization, the total variation can differ, leading to different local minima. This is why it is relevant to iterate the K-means algorithm with different initializations.

3.3 Understanding Key Concepts

3.3.1 Algorithm Convergence

The K-means algorithm is convergent because the minimization function decreases at each iteration and is non-negative. Thus, the algorithm converges to a local minimum.

The minimization function can be written as:

$$L(\mu, z) = \sum_{i=1}^{n} \|x_i - \mu_{z_i}\|^2$$

where $(z_1, \ldots, z_n)$ denotes the cluster indices.

If x_i is a point and z_i its previous cluster index, the new cluster assignment index is:

$$z_i^* \in \arg\min_{j \in \{1,\ldots,k\}} \|x_i - \mu_j\|^2$$

Thus:

$$L(\mu, z^*) - L(\mu, z) = \sum_{i=1}^{n} \|x_i - \mu_{z_i^*}\|^2 - \|x_i - \mu_{z_i}\|^2 \leq 0$$

The same reasoning applies to the new centroid μ^*.

Thus, convergence only occurs at a local minimum because the algorithm will not give the same optimal solution depending on the initial centroid values.

To partition these two figures into four, we can take two different initializations and predict the solution pattern as shown below.

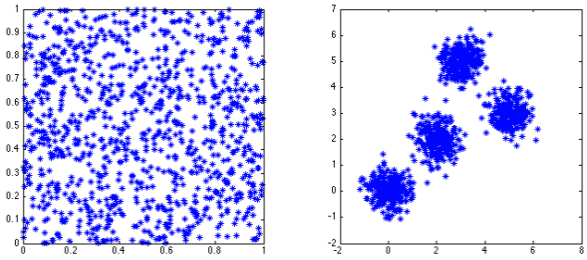

Figure 1: Two different data profiles [2]

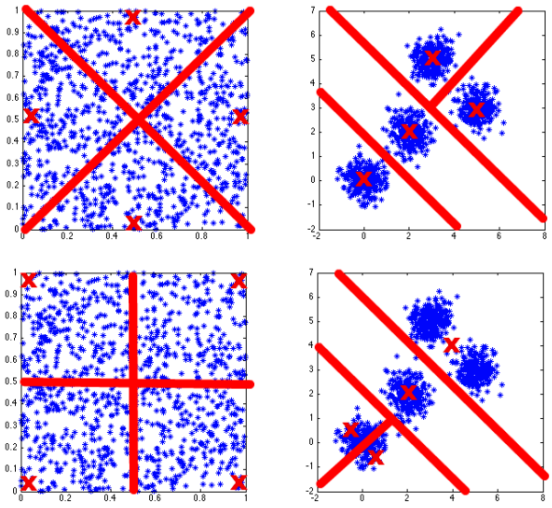

Figure 2: Partitioning of figures based on centroid initialization [2]

3.3.2 Advantages and Disadvantages

We have seen that the algorithm converges to a local minimum, and we can set a finite limit on the number of iterations to bound the complexity. Thus, we have a very fast algorithm that can be executed multiple times to obtain the best result.

The disadvantages come, first, from initialization as described above, and from the choice of the number of clusters. A classic method for choosing the number of clusters is the "elbow method," which involves plotting the data variance as a function of the number of clusters and selecting the point corresponding to the "elbow" of the function.

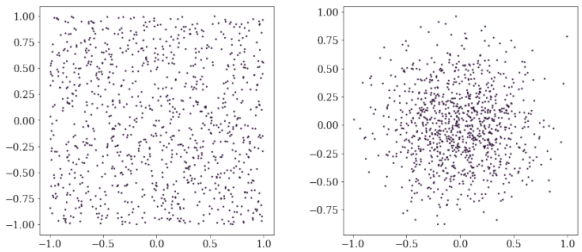

Figure 3: Uniform dataset (left) and normal dataset (right) [3]

Figure 3 shows two datasets of different natures. The authors of the article [3] plotted the variance as a function of the number of clusters and observed that the profile is similar.

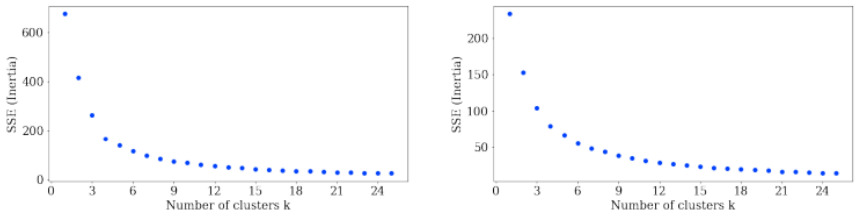

Figure 4: Uniform dataset (left) and normal dataset (right) [3]

An elbow can be seen in both profiles around $k = 3$. However, the authors believe it is very easy to make mistakes and draw incorrect conclusions with a heuristic method like the "elbow method" and suggest other methods such as the "variance-ratio criterion (VRC)" as preferable.

Thus, the K-means algorithm is an algorithm based on simple principles but remains effective for data analysis. However, this algorithm has limitations, such as the dependence of the result on the initial choice and the possibility of convergence to a suboptimal solution.

4 Mean Shift Clustering Algorithm

4.1 General Overview

The Mean Shift algorithm is an unsupervised clustering method that does not require specifying the number of clusters in advance. It is based on density estimation to identify local maxima of a density function (called modes), thereby automatically discovering the clusters present in the data. It is a mode-seeking algorithm.

Starting from a cloud of points in the plane, $P = \{y_i, i = 1, \ldots, I\}$, we want to subdivide P to form ideally disjoint clusters ($P = \cup_{k=1}^{K} P_k$). To do this, we fix a radius $r > 0$, which allows defining the clusters $(C_k)_{1<k<K}$. Each cluster C_k is associated with a closed disk $B(X_k, r)$ centered at X_k with radius r, as well as a subset P_k.

The basic idea of Mean Shift clustering is to move each data point toward the mode (i.e., the highest density) of the point distribution within a certain radius. The algorithm performs these shifts iteratively until the points converge to a local maximum of the density function. These local maxima represent the clusters in the data.

The fundamental idea behind Mean Shift is to consider each data point as a sample from an underlying probability density function of the data space. This probability density function can be defined in different ways depending on the specific problem, but it essentially represents the distribution of data in space.

Then, we aim to define the local probability density. At each data point, a kernel window is defined. This kernel window defines a local region around the point, and the points within this window contribute to calculating the local probability density around the central point. More precisely, each point contributes to the probability density based on its distance to the central point, with a decreasing weight based on this distance. For a data point x_i and $K(x, x_i)$ a symmetric kernel centered at x_i, the local probability density $f(x)$ around x is calculated as the weighted average of the contributions of all data points:

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} K(x, x_i)$$

where N is the total number of data points.

Next, the Mean Shift is calculated as the weighted average displacement vector of the points within the kernel window around the central point. This displacement vector is calculated by taking the weighted average of the vectors connecting the central point to each of the points in the kernel window, with the weights determined by the kernel function used. Thus, the Mean Shift vector is written as:

$$m(x) = \frac{\sum_{i=1}^{N} K(x, x_i) x_i}{\sum_{i=1}^{N} K(x, x_i)} - x$$

Then, the points are updated by moving each point toward a region of higher density, following the previously calculated Mean Shift vector. This process is repeated until the points converge to the local modes of maximum density. The new locations of the points are determined by:

$$x_i^{(t+1)} = x_i^{(t)} + m(x_i^{(t)})$$

where $x_i^{(t)}$ is the position of point x_i at iteration t.

As the points move according to the Mean Shift vector, they converge toward the local modes of maximum density of the underlying probability density function. This means that each point eventually clusters around a specific mode, representing a cluster.

Once convergence is reached, the points are assigned to clusters based on their proximity to the local modes. Points that converge to the same mode are grouped into the same cluster.

In summary, Mean Shift exploits the density structure of the data to identify local modes of maximum density, moving points toward these modes at each iteration until convergence. This allows grouping points into clusters based on the local density of the data, without requiring a priori specifications of the number of clusters.

4.2 Kernels and Bandwidth

To fully understand Mean Shift clustering, it is essential to grasp the concepts of kernels and bandwidth.

A kernel is a function that assigns weights to data points based on their distance from a reference point. The choice of kernel influences how local data are smoothed and how the center of gravity (or mean) is calculated.

Kernels are generally used to weight the contributions of neighboring points when calculating the center of gravity. A commonly used kernel in Mean Shift is the **Gaussian kernel**, defined as follows:

$$K(x) = e^{-\frac{\|x\|^2}{2}}$$

In this formula, x represents the distance between a data point and the reference point, and $K(x)$ is the weight assigned to that distance. Points closer to the reference point receive higher weights, while points farther away receive lower weights.

Other types of kernels can also be used, such as the uniform kernel, the Epanechnikov kernel, or the tricube kernel. The choice of kernel can affect the shape and size of the clusters detected by the algorithm.

The other crucial concept is the bandwidth, often denoted h, which is a critical parameter that determines the range of the kernel, i.e., the maximum distance at which data points influence the calculation of the center of gravity.

In practice, the bandwidth defines the size of the neighborhood window around each data point. More precisely, it controls the area in which data points will be considered to estimate the local density and determine the direction of the point shifts.

A small bandwidth leads to a restricted neighborhood window, which can result in the formation of many small clusters. This setting is sensitive to local variations and can capture fine details in the data. Conversely, a large bandwidth leads to a larger neighborhood window, which can merge several small clusters into a single large cluster. This setting smooths local variations and may ignore fine details but is more robust to noise.

The kernel and bandwidth are essential and linked for estimating the local density around each data point and determining the direction of the shift. During each Mean Shift iteration, the center of gravity is calculated by weighting neighboring data points using the kernel and then moving them toward this weighted average.

The general formula for the shift of a point x using a kernel K and a bandwidth h is:

$$m(x) = \frac{\sum_{x_i \in N(x,h)} K\left(\frac{x_i - x}{h}\right) x_i}{\sum_{x_i \in N(x,h)} K\left(\frac{x_i - x}{h}\right)}$$

where $N(x, h)$ represents the set of points within the bandwidth window around x.

The choice of bandwidth h is crucial for the performance and results of Mean Shift clustering. A poorly chosen bandwidth can lead to unsatisfactory clustering results.

4.3 Density Gradient

We will show that the Mean Shift vector is strongly related to the density gradient of the data.

The author of this article defines a "shadow kernel" that corresponds to another kernel function related to our kernel.

Thus, a kernel H is said to be a "shadow kernel" of a kernel K if the Mean Shift using K, $m(x) = \frac{\sum_{i=1}^{N} K(x,x_i)x_i}{\sum_{i=1}^{N} K(x,x_i)} - x$, is in the direction of the gradient of the density estimated using H, $q(x) = \sum_{i=1}^{N} H(x, x_i)x_i$.

From this definition, a theorem allows relating the shadow kernel H to the initial kernel K: H is the shadow kernel of K if and only if the profiles of the two kernels h and k satisfy:

$$h(r) = f(r) + c \int_r^{\infty} k(t)\,\mathrm{d}t \tag{1}$$

where f is a constant function and c is a positive constant.

For example, the Kernel used in our project, which is the ball function of radius $\Lambda > 0$, has a shadow Kernel: the Epanechnikov Kernel.

The theorem implies that the Mean Shift algorithm is a gradient method with the data density. Thus, the convergence of the algorithm is a consequence of the convergence of the gradient algorithm.

4.4 Advantages and Disadvantages of Mean Shift

Mean Shift has several advantages:

Unlike K-means, Mean Shift does not require specifying the number of clusters in advance, making it a more flexible method suitable for situations where the data structure is unknown.

Mean Shift can identify clusters of arbitrary shapes and varying sizes, unlike K-means, which assumes spherical clusters.

The algorithm is less sensitive to outliers because they do not significantly influence the density maxima.

However, Mean Shift also has disadvantages:

The repeated calculation of centers of gravity for each data point can be computationally expensive, especially for large datasets.

The appropriate choice of bandwidth h is crucial for Mean Shift's performance. A bandwidth that is too small can lead to many small clusters, while a bandwidth that is too large can merge distinct clusters.

Data points can converge to different local maxima, leading to clustering results that strongly depend on the local data density.

4.5 Summary of Steps

Choose an initial point: The algorithm starts by selecting a data point as the initial point. This choice can be random or based on a predefined strategy.

Then, calculate the center of gravity: For each initial point, the center of gravity (or mean) of neighboring data points within a window of size h (called bandwidth) is calculated. The bandwidth determines the local region over which the density is estimated.

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

where $N(x)$ is the set of points within the bandwidth window around x, and K is a kernel function often chosen as the Gaussian kernel.

Shift the Point: The initial point is shifted toward the calculated center of gravity.

$$x \leftarrow m(x)$$

Convergence: Steps 2 and 3 are repeated until the point's displacement is below a predefined threshold, indicating that the point has reached a region of maximum density.

Group Converged Points: Data points that have converged to nearby local maxima are grouped to form the final clusters.

5 Codes for the Algorithms

We start by generating random points following a Gaussian distribution.

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

# Define parameters
J = 5  # Number of initial clusters
N = 1000  # Number of points per cluster
mu_min, mu_max = -40, 40  # Range for means
sigma_min, sigma_max = 1, 2  # Range for standard deviations

# Generate random means
mu_x = np.random.uniform(mu_min, mu_max, J)
mu_y = np.random.uniform(mu_min, mu_max, J)

# Initialize the set P
P = np.empty((0, 2))

# For each cluster
for j in range(J):
    # Generate Gaussian parameters
    sigma_x = np.random.uniform(sigma_min, sigma_max)
    sigma_y = np.random.uniform(sigma_min, sigma_max)
    covariance = np.random.uniform(-0.8, 0.8) * sigma_x * sigma_y

    # Generate points following a 2D Gaussian distribution
    clusterd = np.random.multivariate_normal([mu_x[j], mu_y[j]],
    [[sigma_x, covariance], [covariance, sigma_y]], N)

    # Add points to the point cloud
    P = np.vstack((P, clusterd))

# Plot the point cloud
plt.figure(figsize=(8, 6))
plt.scatter(P[:, 0], P[:, 1], s=5, alpha=0.5)
plt.title('Point cloud in $\\mathbb{R}^2$ with varied Gaussian clusters')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.grid(True)
plt.show()

```

5.1 Code for the K-means Algorithm

5.1.1 Code

```
# Initialize cluster centers
def init_(P, J):
    indices = np.random.choice(range(len(P)), J, replace=False)
    return P[indices]

# Implement K-means algorithm with total variation calculation
def k_means(P, J, max_iter=300, tol=1e-4):
    means_init = init_(P, J)
    for _ in range(max_iter):
        # Step 1: Assign each point to the nearest cluster
        clusters = [[] for _ in range(J)]
        for point in P:
            distances = np.linalg.norm(means_init - point, axis=1)
            nearest_cluster = np.argmin(distances)
            clusters[nearest_cluster].append(point)

        # Step 2: Recalculate cluster centers
        new_means = []
        for i, cluster in enumerate(clusters):
            if cluster:
                new_means.append(np.mean(cluster, axis=0))
            else:
                new_means.append(means_init[i]) # Keep old center if cluster is empty
        new_means = np.array(new_means)

        # Check convergence
        if np.linalg.norm(means_init - new_means, axis=None) < tol:
            break

        means_init = new_means

    # Calculate total variation W
    W = 0
    for i, cluster in enumerate(clusters):
        if cluster:
            W += np.sum(np.linalg.norm(np.array(cluster) - means_init[i], axis=1)**2)

    return clusters, means_init, W

# Run K-means multiple times to minimize total variation
num_runs = 20
```

```
best_W = np.inf
best_clusters = None
best_means = None

for _ in range(num_runs):
    clusters, means_init, W = k_means(P, J)
    if W < best_W:
        best_W = W
        best_clusters = clusters
        best_means = means_init

print(f"Best VT obtained: {best_W}")

# Plot the best clusters
plt.figure(figsize=(8, 6))
colors = ['b', 'g', 'r', 'c', 'm']
for i in range(J):
    cluster_points = np.array(best_clusters[i])
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], s=5, alpha=0.5,
    color=colors[i], label=f'Cluster {i+1}')
    plt.scatter(best_means[i, 0], best_means[i, 1], s=100, color=colors[i], edgecolor='k')

plt.title('Clusters generated by the K-means algorithm')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.legend()
plt.grid(True)
plt.show()

```

5.1.2 Complexity Study

Let's analyze the complexity of the K-means algorithm by breaking down each step and evaluating their complexity:

For the initialization of cluster centers (init function), the random selection of initial centers from the data points has a complexity of $O(J)$, where J is the number of clusters. This is because we use 'np.random.choice' to select J indices from N points without replacement.

Next, for assigning points to the nearest clusters, for each point, we must calculate the distance to each of the J cluster centers. This takes $O(J)$ per point. Thus, for N points, the total complexity of this step is $O(NJ)$.

Then, for recalculating cluster centers, for each cluster, we recalculate the centroid by taking the mean of all points assigned to that cluster. If n_i is the number of points in cluster i, recalculating each centroid takes $O(n_i)$. Thus, the total complexity for recalculating all centers is $O(N)$ because each point is used once to recalculate a centroid.

For checking convergence, calculating the norm between the old and new centers has a complexity of $O(J)$.

For calculating the total variation, for each cluster, we calculate the sum of squared distances between each point and the cluster center. For a cluster i containing n_i points, this operation takes $O(n_i)$. Thus, the total complexity for this step is also $O(N)$.

For the main loop (iterations), steps 2, 3, and 4 are repeated until convergence. If we denote the number of iterations as T, then the total complexity of the repeated steps is $O(T \cdot (NJ + N + J))$, which simplifies to $O(TNJ)$ since $N \gg J$.

Finally, we execute K-means multiple times to minimize the total variation. By running the K-means algorithm R times to find the best clustering, the total complexity is $O(RTNJ)$.

Thus, the complexity of the K-means algorithm, with R executions to minimize the total variation, is dominated by $O(RTNJ)$.

5.2 Code for the Mean Shift Clustering Algorithm

5.2.1 Code

```
# Function to find points within a given radius around a center
def points_in_radius(points, center, radius):
    distances = np.linalg.norm(points - center, axis=1)
    return points[distances <= radius], distances[distances <= radius]

# Function to compute the Gaussian kernel
def gaussian_kernel(distance, bandwidth):
    return np.exp(-0.5 * (distance / bandwidth) ** 2)

# Function to compute the uniform kernel
def uniform_kernel(distance, radius):
    return np.ones_like(distance)

# Function to compute the weighted barycenter of points
def calculate_weighted_barycenter(points, distances, kernel, bandwidth):
    weights = kernel(distances, bandwidth)
    return np.sum(points * weights[:, np.newaxis], axis=0) / np.sum(weights)

# Function to update the cluster center
def update_cluster_center(points, center, radius, kernel, bandwidth):
    points_in_cluster, distances = points_in_radius(points, center, radius)
    if len(points_in_cluster) > 0:
        new_center = calculate_weighted_barycenter(points_in_cluster, distances, kernel, bandwidth)
    else:
        new_center = center
    return new_center, points_in_cluster

# Function to perform dynamic clustering
def cluster_dynamic(P, centers, r, kernel, bandwidth):
    while True:
        new_centers = []
        cluster_points = []
        for center in centers:
            new_center, new_points = update_cluster_center(P, center, r, kernel, bandwidth)
            new_centers.append(new_center)
            cluster_points.append(new_points)

        new_centers = np.array(new_centers)
        if np.allclose(centers, new_centers):
            break
        centers = new_centers
```

```
    return centers, cluster_points

# Function to remove empty clusters or those with fewer points than the minimum threshold
def delete_clusters_below_threshold(centers, cluster_points, min_points):
    valid_clusters = [points for points in cluster_points if len(points) >= min_points]
    valid_centers=[centers[i] for i,points in enumerate(cluster_points) if len(points)>=min_points]
    return np.array(valid_centers), valid_clusters

# Function to merge nearby clusters
def merge_clusters(centers, cluster_points, r):
    merged_centers = []
    merged_cluster_points = []
    used = np.zeros(len(centers), dtype=bool)

    for i, center in enumerate(centers):
        if used[i]:
            continue
        merged_center = center
        merged_points = cluster_points[i]
        for j in range(i + 1, len(centers)):
            if not used[j] and np.linalg.norm(centers[i] - centers[j]) < r:
                merged_points = np.vstack((merged_points, cluster_points[j]))
                used[j] = True
        merged_centers.append(merged_center)
        merged_cluster_points.append(merged_points)

    return np.array(merged_centers), merged_cluster_points

# Function to assign points to the nearest clusters
def assign_points_to_clusters(points, centers):
    assigned_clusters = []
    for point in points:
        distances = np.linalg.norm(point - centers, axis=1)
        nearest_cluster_index = np.argmin(distances)
        assigned_clusters.append(nearest_cluster_index)
    return np.array(assigned_clusters)

# Function to plot clustering results
def plot_cluster_result(centers, cluster_points, kernel_name):
    plt.figure(figsize=(16, 10))
    for i, points in enumerate(cluster_points):
        plt.scatter(points[:, 0], points[:, 1], label=f'Cluster {i+1}')
    plt.scatter(centers[:, 0], centers[:, 1], color='red', marker='*', s=200, label='Cluster Centers')
    plt.xlabel('X')
```

```
    plt.ylabel('Y')
    plt.title(f'Final Cluster Result using {kernel_name} Kernel')
    plt.legend()
    plt.grid(True)
    plt.show()

# Main clustering function
def dynamic_clustering(P, r, min_points, distribution='gaussian'):
    if distribution == 'gaussian':
        kernel = gaussian_kernel
        bandwidth = r
        kernel_name = 'Gaussian'
    elif distribution == 'uniform':
        kernel = uniform_kernel
        bandwidth = r
        kernel_name = 'Uniform'
    else:
        raise ValueError("Distribution must be 'gaussian' or 'uniform'.")

    # Define x and y coordinates of the point cloud
    x = P[:, 0]
    y = P[:, 1]

    # Determine the limits of rectangle R
    x_min, x_max = np.min(x), np.max(x)
    y_min, y_max = np.min(y), np.max(y)

    # Create the Cartesian grid
    grid_spacing = 10
    x_grid = np.arange(np.floor(x_min), np.ceil(x_max) + grid_spacing, grid_spacing)
    y_grid = np.arange(np.floor(y_min), np.ceil(y_max) + grid_spacing, grid_spacing)

    # Initialize cluster centers at grid nodes
    centers = np.array([(x_, y_) for x_ in x_grid for y_ in y_grid])

    # Perform dynamic clustering
    updated_centers, updated_cluster_points = cluster_dynamic(P, centers, r, kernel, bandwidth)

    # Remove empty clusters or those with fewer points than the minimum threshold
    updated_centers,updated_cluster_points= delete_clusters_below_threshold(updated_centers,
                                            updated_cluster_points, min_points)

    # Merge nearby clusters (adjusted merge radius)
    merged_centers, merged_cluster_points = merge_clusters(updated_centers, updated_cluster_points, r)

```

```
    # Assign each point to the nearest cluster
    assigned_clusters = assign_points_to_clusters(P, merged_centers)

    # Plot final result
    plot_cluster_result(merged_centers, merged_cluster_points, kernel_name)

    return merged_centers, merged_cluster_points, assigned_clusters

# Example usage
r = 4
min_points = 5

# For Gaussian kernel
dynamic_clustering(P, r, min_points, distribution='gaussian')

# For uniform kernel
dynamic_clustering(P, r, min_points, distribution='uniform')
```

5.2.2 Complexity Study

Let's analyze the complexity of the Mean Shift Clustering algorithm.

For finding points within a given radius around a center (points_in_radius function), for each data point, this function calculates the distance between the point and the given center and returns the points within the specified radius. Calculating distances is $O(N)$ for N points, and selecting points within the radius is $O(N)$. Thus, the total complexity of the function is $O(N)$.

The gaussian kernel function calculates the Gaussian kernel for a given distance and bandwidth. The calculation for each distance is $O(1)$. Thus, the total complexity is $O(N)$ because it is called for N distances.

The uniform kernel function calculates the uniform kernel by returning an array of 1s for each distance. The calculation for each distance is $O(1)$. The total complexity is $O(N)$.

The calculate weighted barycenter function calculates weights for each point and the weighted barycenter. Calculating weights is $O(N)$, and calculating the barycenter is $O(N)$. The total complexity is $O(N)$.

The update cluster center function updates the cluster center by finding points within a given radius and calculating the weighted barycenter. Finding points within the radius is $O(N)$, and calculating the barycenter is $O(N)$. The total complexity is $O(N)$.

The cluster dynamic function updates the cluster centers until convergence. For the initialization of the lists, the complexity is $O(J)$ for J centers. For updating the cluster centers, the complexity is $O(JN)$ per iteration. Thus, the total complexity is $O(TJN)$, where T is the number of iterations until convergence.

The delete clusters below threshold function removes clusters containing fewer than the minimum threshold of points. Filtering the clusters has a complexity of $O(J)$ since each cluster is checked. The total complexity is therefore $O(J)$.

The merge clusters function merges clusters whose centers are close to each other. Checking the distances between clusters has a complexity of $O(J^2)$ to compare all pairs of clusters. As for merging the clusters, it is $O(JN)$ to reassign the points. The total complexity is thus $O(J^2 + JN)$.

The assign points to clusters function assigns each point to the nearest cluster. Calculating distances for each point has a complexity of $O(J)$, and assigning N points has a complexity of $O(NJ)$. The total complexity is therefore $O(NJ)$.

The plot cluster result function plots the final clustering results. Plotting the points and centers has a complexity of $O(N)$. The total complexity is $O(N)$.

Finally, the dynamic clustering function executes the steps of dynamic clustering.

- Initialization of centers on the grid: $O(G^2)$, where G is the number of points on the grid (depends on the grid dimensions). - Dynamic clustering: $O(TJN)$. - Removal of empty clusters: $O(J)$. - Merging nearby clusters: $O(J^2 + JN)$. - Point assignment: $O(NJ)$. - Plotting results: $O(N)$.

The total complexity is: $O(G^2 + TJN + J^2 + JN)$.

In summary, the time complexity of the Mean Shift Clustering algorithm is dominated by $O(TJN + J^2 + G^2)$, where: - T is the number of iterations until convergence, - J is the initial number of cluster centers, - N is the number of data points, - G is the number of grid points used to initialize the cluster centers.

5.3 Comparison of the Two Algorithms

For K-means, the complexity is $O(TNJ)$. The algorithm primarily depends on the number of points N, the number of clusters J, and the number of iterations until convergence T. For Mean Shift Clustering, the complexity is $O(G^2 + TJN + J^2 + JN)$.

The algorithm includes an additional dependency on the number of grid points G. The dominant terms are $O(TNJ)$ and $O(J^2 + JN)$, but $O(G^2)$ can also be significant.

Regarding initialization complexity: K-means has a simpler initialization with $O(J)$. Mean Shift Clustering requires grid initialization $O(G^2)$, which can be more costly.

Convergence considerations: K-means tends to converge faster with simple iterations $O(NJ)$. Mean Shift Clustering may take longer due to merging steps and barycenter recalculations.

The algorithms have parameter dependencies:

K-means is more sensitive to the initial number of clusters J. Mean Shift Clustering is influenced by the choice of radius and bandwidth, as well as the grid density.

The K-means algorithm is generally simpler and faster in terms of time complexity, but it may be less flexible and sensitive to initial choices. Mean Shift Clustering is more complex and potentially more computationally expensive, but it can provide more precise and robust results, especially for non-convex data distributions. The choice between the two often depends on the specific needs of the application and the characteristics of the data.

6 Results

6.1 K-means

6.1.1 Results

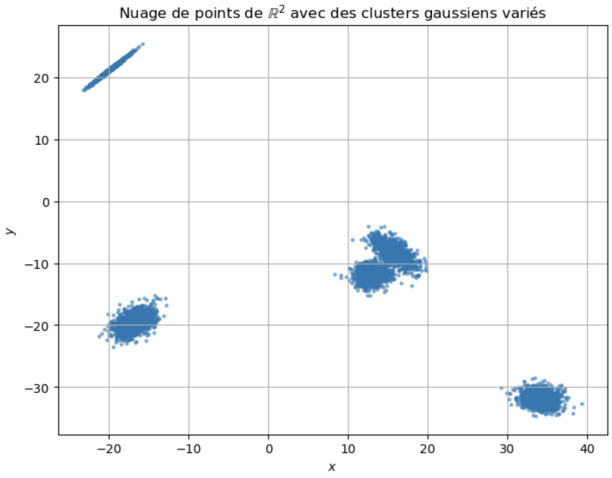

Figure 5: Point distribution in the plane

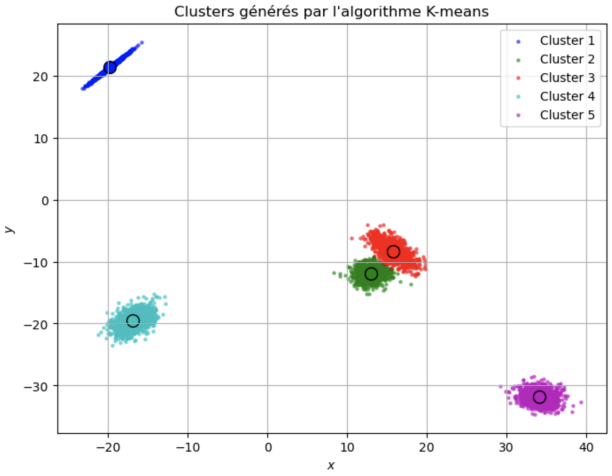

Figure 6: Clusters after applying K-means algorithm

In Figures 5 and 6, we clearly observe the clusters separated by the K-means algorithm. Next, we will examine the influence of algorithm parameters, particularly the number of clusters given as input.

6.1.2 Elbow Method

According to theory, the goal is to find the "elbow" of the graph - the point where adding new clusters no longer significantly improves the sum of squared distances from points to

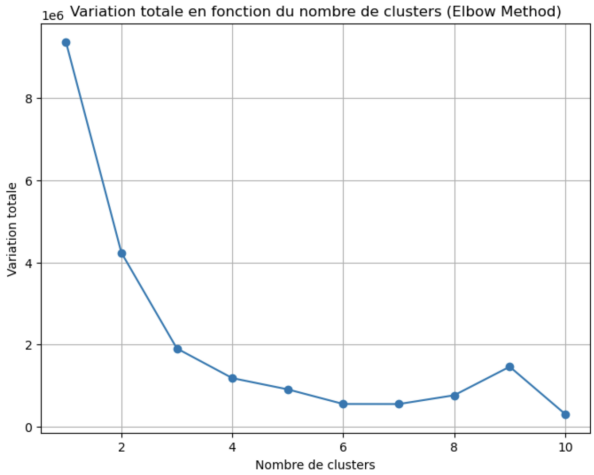

Figure 7: Total variation as function of number of clusters

their cluster centers (total variation).

We observe a rapid continuous decrease in total variation up to about 3 or 4 clusters. Beyond 4 clusters, the decrease becomes less significant and the curve stabilizes.

Thus, the "elbow" is generally considered as the point where the reduction in total variation becomes less pronounced. Visually, this appears around 3 or 4 clusters in our case.

The optimal point (the elbow) indicates the number of clusters where improvement in total variation begins to diminish. This means that beyond this point, adding more clusters provides only marginal improvements. In our case, choosing 3 or 4 clusters appears to be a good compromise between model complexity (number of clusters) and performance (total variation).

6.2 Mean Shift Clustering

6.2.1 Kernel Comparison

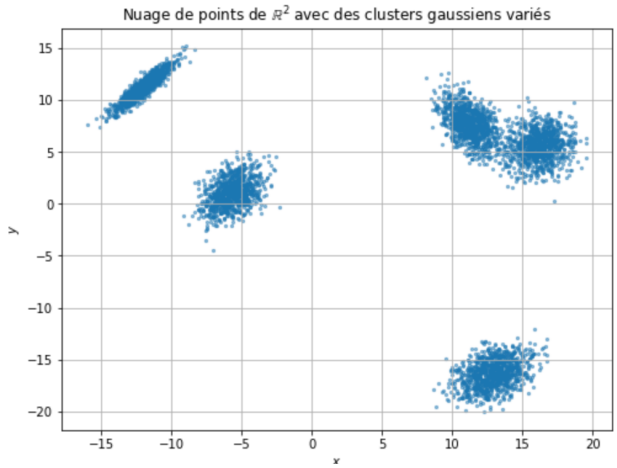

Figure 8: Point distribution in the plane

We created a random point distribution following Gaussian distributions as seen in the coding section. We can observe five Gaussian clouds, two of which are relatively close together.

We will now examine the results of Mean Shift Clustering using both uniform and Gaussian kernels.

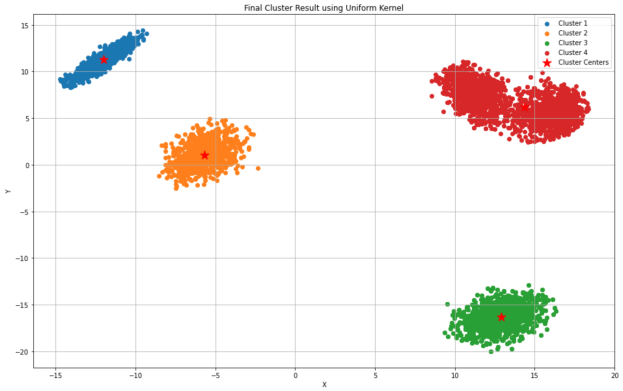

Figure 9: Same distribution clustered using Mean Shift with uniform kernel

We can see that Mean Shift clustering with uniform kernel forms four clusters for the given distribution. It considers that one of the clouds should merge with another based on our parameters.

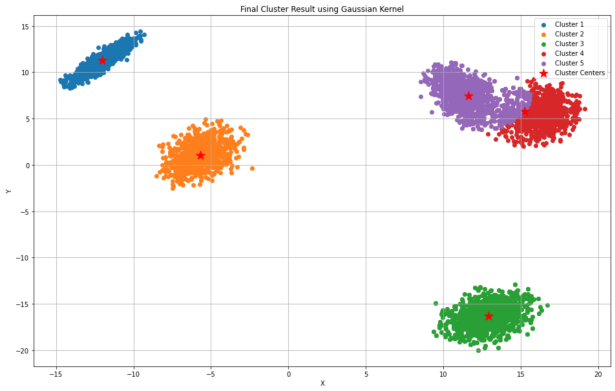

Figure 10: Same distribution clustered using Mean Shift with Gaussian kernel

Here we see the result for the same distribution but with a Gaussian kernel. We obtain five clusters, which is different from the uniform kernel result.

This can be explained in several ways: The Gaussian kernel applies higher weights to points closer to the center and exponentially decreasing weights to more distant points. Consequently, clusters are more strongly influenced by nearby points, creating typically smaller clusters. In contrast, the uniform kernel gives equal weight to all points within the radius, which can lead to broader, less dense clusters since all points contribute equally to the cluster center.

Moreover, when merging nearby clusters, the uniform kernel is more likely to merge clusters because the distance between cluster centers is less significant compared to the Gaussian kernel. The Gaussian kernel, with its more precise weighting, maintains better cluster separation.

6.2.2 Parameter Influence

Both Gaussian and uniform kernels depend on a parameter - the ball radius for uniform kernel and bandwidth for Gaussian. We'll examine the influence using just the uniform kernel as example, since they behave similarly with respect to their parameters.

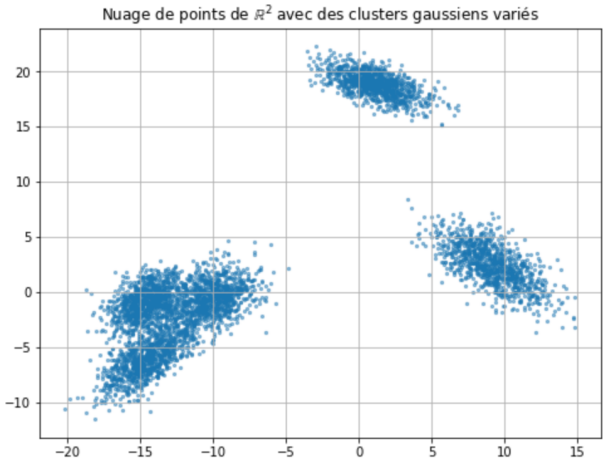

Figure 11: Point distribution in the plane

In this example, we observe three very close Gaussian clusters and two more distant ones.

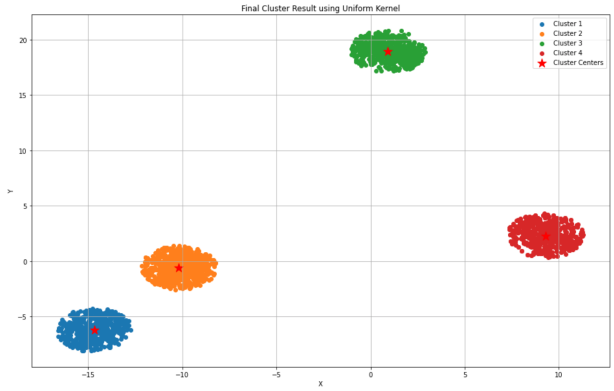

Figure 12: Same distribution clustered using Mean Shift with uniform kernel (r=2)

We observe that the same distribution is divided into four clusters for $r = 2$ and three clusters for $r = 4$. When the radius is small, the uniform kernel only covers a small region around each initial center. The formed clusters are small and sensitive to local variations. Each small group of points can be detected as a distinct cluster. Therefore, we generally obtain more clusters since the kernel doesn't effectively group more distant points.

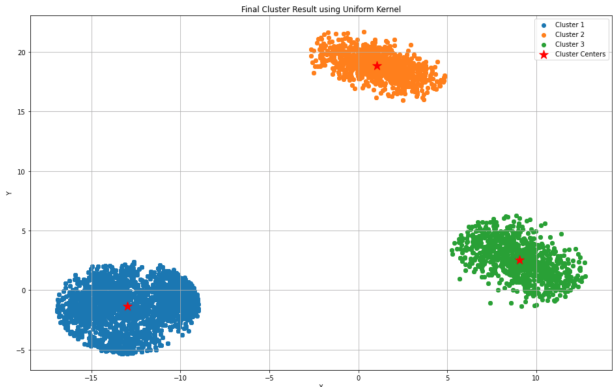

Figure 13: Same distribution clustered using Mean Shift with uniform kernel (r=4)

6.3 Algorithm Comparison

From a general perspective, K-means and Mean Shift differ in their handling of cluster count. The K-means algorithm requires knowing the initial number of clusters to find them, while Mean Shift doesn't require specifying the number of clusters as it's determined by the density modes found. Additionally, to get good results with K-means, multiple initializations are needed to find the smallest total variation, whereas Mean Shift is less sensitive to initial points since each point is a potential cluster center, converging toward density modes. However, Mean Shift can be slower, especially with large datasets, due to searching for density modes for each point.

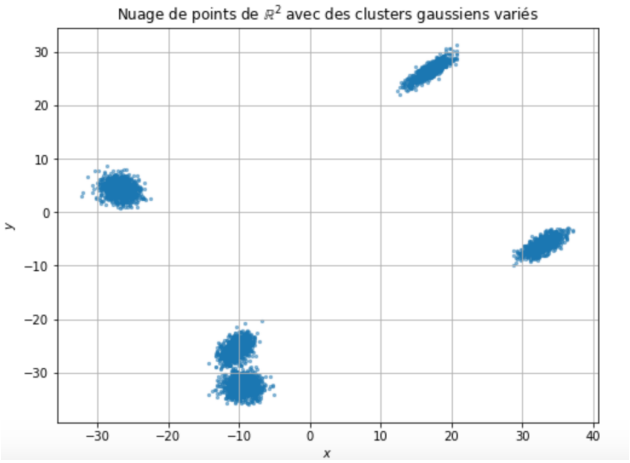

Figure 14: Point distribution in the plane

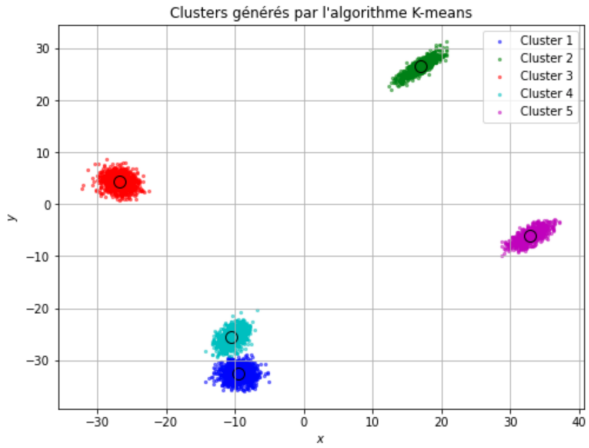

Figure 15: Same distribution clustered using K-means

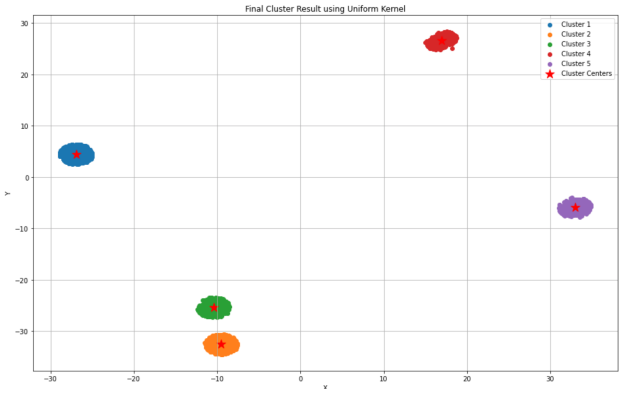

Figure 16: Same distribution clustered using Mean Shift with uniform kernel (r=2)

In our case, both algorithms detected the five initial Gaussian clusters, but Mean Shift's cluster centers moved due to the density mode seeking. This demonstrates Mean Shift's flexibility in detecting local density structures more precisely, even if it may shift centers compared to the exact means of initial Gaussians. K-Means maintained centers closer to the initial cluster means due to its approach based on averaging assigned points.

7 Conclusion

The project aimed to compare two popular clustering algorithms, K-Means and Mean Shift, by applying each to a point distribution generated by Gaussian mixtures. This project demonstrated both algorithms' ability to detect clusters in Gaussian-generated data while illustrating each method's strengths and limitations.

The K-means algorithm works by iterating between two main steps: assigning data points to the nearest cluster and updating cluster centers based on assigned points. K-means seeks to minimize total variation (sum of squared distances between data points and their respective cluster centers). This algorithm's strengths include being easy to implement and generally

fast for moderately sized datasets, with guaranteed convergence to a local minimum of total variation.

However, K-means also has limitations. Results can vary depending on cluster center initialization. Additionally, it assumes spherical clusters of similar size, which can be restrictive. Most importantly, the algorithm requires specifying the number of clusters in advance, which isn't always obvious.

Solutions exist though: methods like K-means++ provide better center initialization, and algorithms like DBSCAN or agglomerative clustering don't require specifying cluster count.

The Mean Shift algorithm is a non-parametric clustering method that doesn't require specifying the number of clusters beforehand. It works by moving data points toward the highest density of neighboring points until they converge to density modes.

Mean Shift's strengths include detecting arbitrarily shaped clusters and not requiring pre-specification of cluster count. Being non-parametric, it uses data density to determine clusters, making it adaptable to different data types.

However, Mean Shift also has limitations. Its high computational complexity makes it more expensive, especially for large datasets. Performance and results heavily depend on bandwidth choice, which can be difficult to determine.

Fortunately, solutions exist for Mean Shift's limitations: heuristic or adaptive methods can help choose appropriate bandwidth, and optimization/dimensionality reduction techniques can handle large datasets more efficiently.

In summary, K-means is fast and simple but limited by its assumptions about cluster shape and count. Mean Shift is more flexible and non-parametric but can be computationally expensive and requires good bandwidth estimation. The choice between them depends on the specific problem, data characteristics, and performance/flexibility requirements.

For choosing between the two algorithms, prefer K-means when data are well-separated, spherical, and the cluster count is known. Choose Mean Shift for complex-shaped data or when the cluster count is unknown or hard to estimate.

8 Bibliography

1 T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, vol. 2, Springer, 2009.

2 LAS Group, Institute for Machine Learning Dept. of Computer Science, ETH Zürich Prof. Dr. Andreas Krause

3 Schubert, E. 2023. *Stop using the elbow criterion for k-means and how to choose the number of clusters instead.*

4 https://dm.cs.tu-dortmund.de/

5 Cheng, Y. *Mean Shift, Mode Seeking, and Clustering.*