CLUSTERING

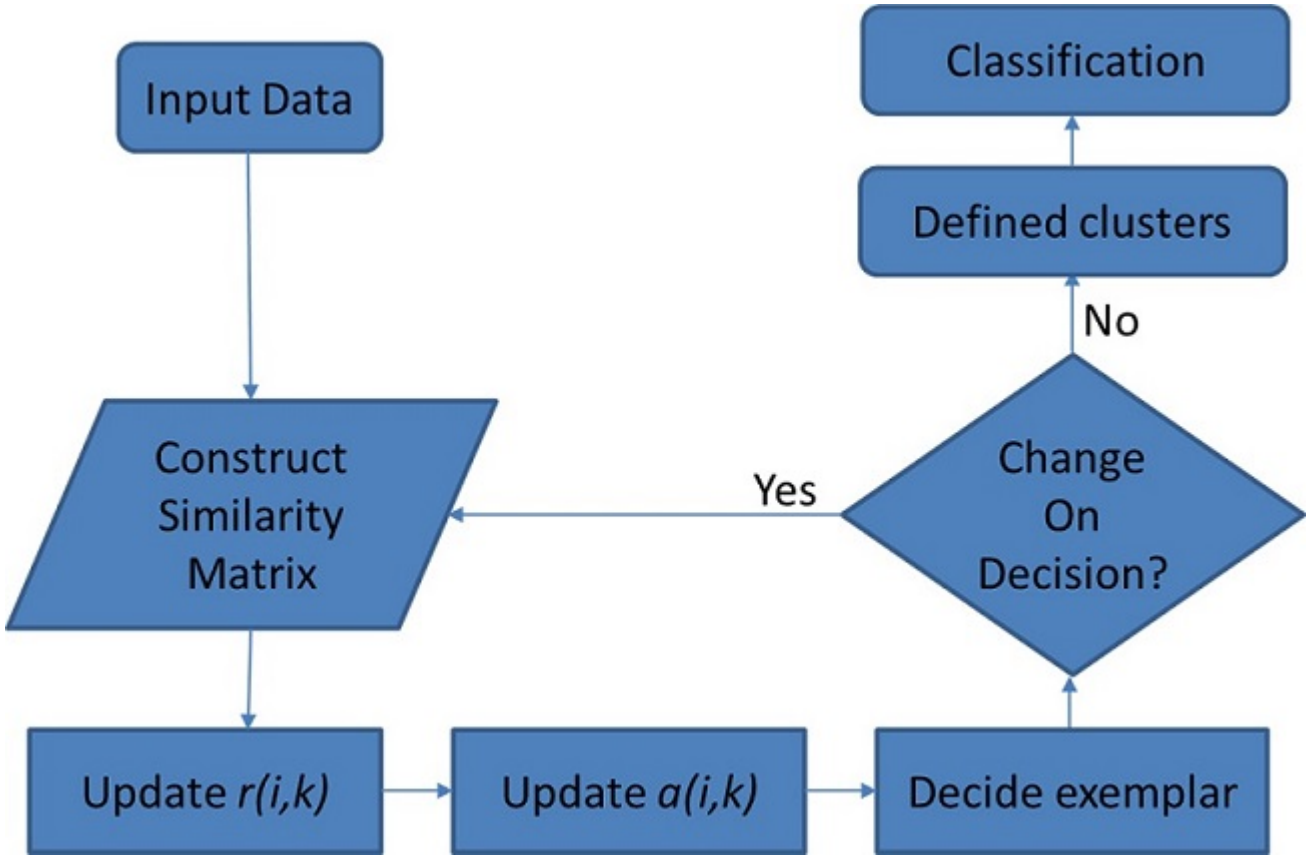
Algorithms Used in Clustering:

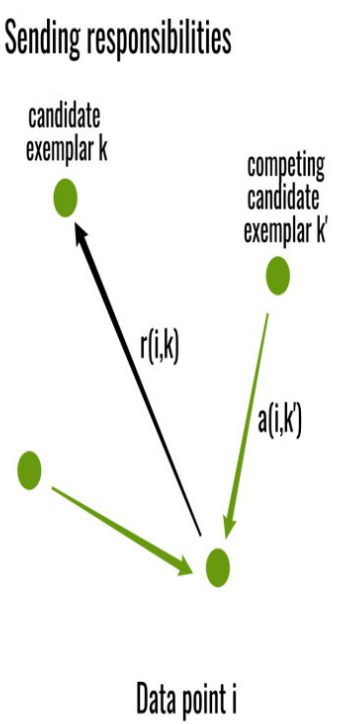
1. **Affinity Propagation**
2. **DBSCAN**
3. **HDBSCAN**
4. **Mean Shift**
5. **Gaussian Mixture model**
6. **Optics**
7. **BIRCH**
8. **Fuzzy**

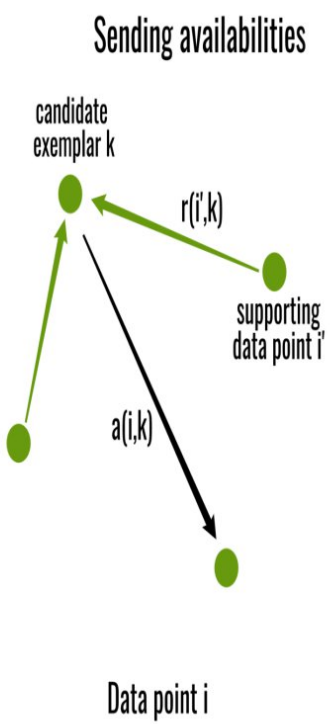
AFFINITY PROPAGATION

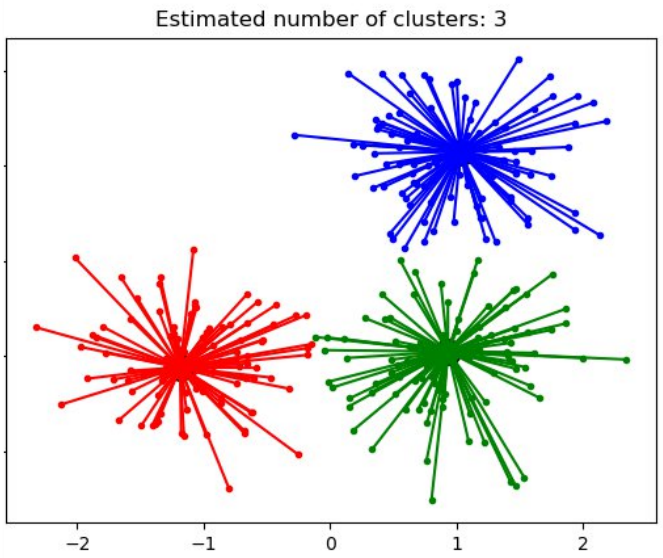
- Affinity Propagation is a clustering method that **groups similar data points together**.
- Unlike other methods like K-Means it **doesn’t need you to set the number of clusters**.
- Instead it figures out the number of clusters on its own.
- It allows data points to **communicate with each other by sending messages**.
- These messages help them decide which points should act as leaders.

Flow Diagram for Affinity Propagation




Sending responsibilities
candidate exemplar k
competing candidate exemplar k'
r(i,k)
a(i,k')
Data point i


Sending availabilities
candidate exemplar k
r(i',k)
supporting data point i'
a(i,k)
Data point i

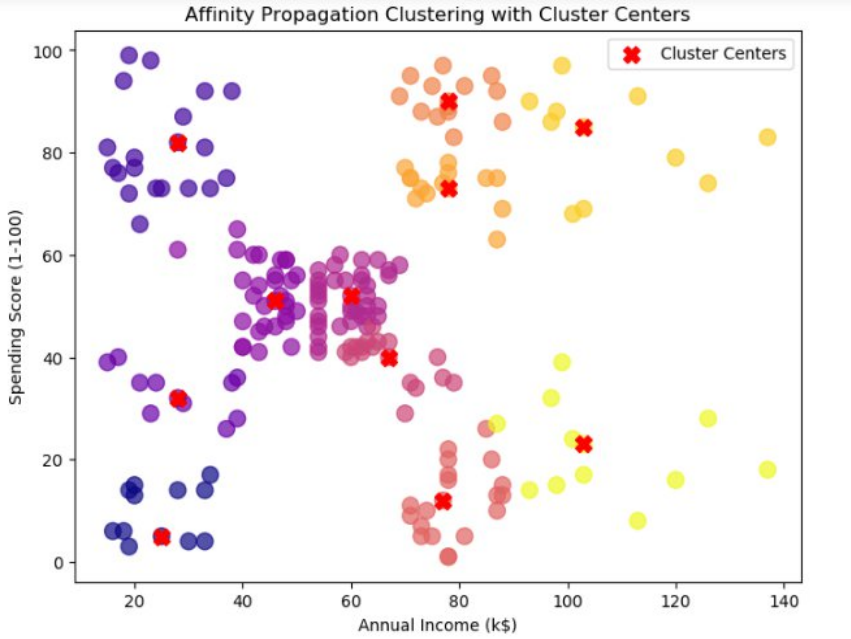

Estimated number of clusters: 3
−2
−1
0
1
2

```
]: from sklearn.cluster import AffinityPropagation
   clustering = AffinityPropagation().fit(x)
   clustering
```

```
]: AffinityPropagation(affinity='euclidean', convergence_iter=15, copy=True,
                      damping=0.5, max_iter=200, preference=None, verbose=False)
```

```
]: clustering.labels_
```

```
plt.figure(figsize=(8,6))

#Colormap = 'viridis','plasma','coolwarm'
plt.scatter(x[:,0], x[:,1], c=y_pred, cmap='plasma', s=100, alpha=0.7

# Plot cluster centers
plt.scatter(ap.cluster_centers_[:,0], ap.cluster_centers_[:,1],
            c='red', marker='X', s=100, label='Cluster Centers')

plt.title("Affinity Propagation Clustering with Cluster Centers")
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.legend()
plt.show()
```



DBSCAN(Density Based Spacial Clustering of Applications with Noise)

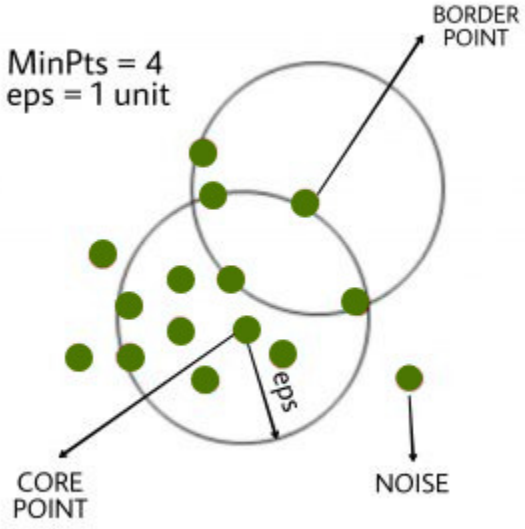
- DBSCAN is a density-based clustering algorithm that groups data points that are closely packed together and marks outliers as noise based on their density in the feature space.
- It identifies clusters as dense regions in the data space separated by areas of lower density.
- Key Parameters : Epsilon(eps), MinPts

How Does DBSCAN Work?

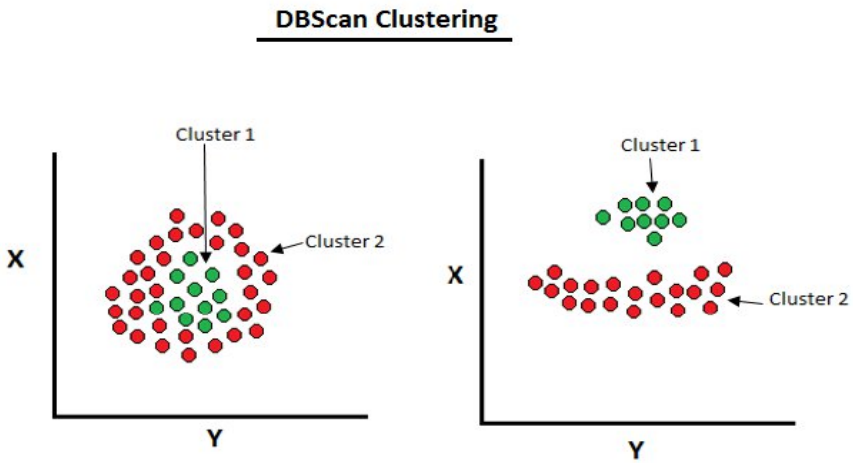
DBSCAN works by categorizing data points into three types:

- Core points which have a sufficient number of neighbors within a specified radius (eplison)
- Border points which are near core points but lack enough neighbors to be core points themselves
- Noise points which do not belong to any cluster.

Steps involved:



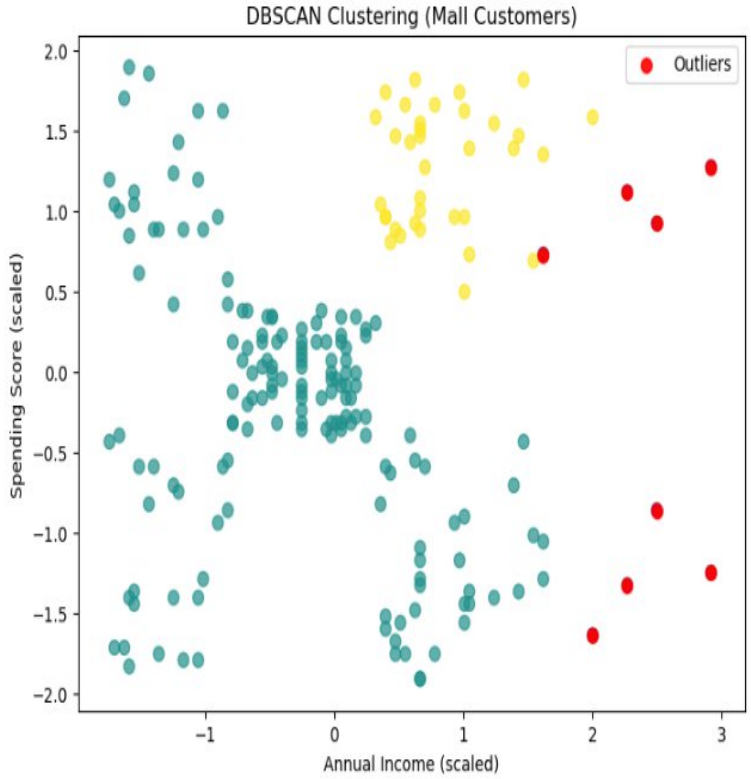
- Identify Core Points
- Form Clusters
- Density Connectivity
- Label Noise Points



```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x = sc.fit_transform(x)
```

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5,min_samples=5)
y_dbscan = dbscan.fit_predict(x)
```

```
# plot
plt.figure(figsize=(8,6))
plt.scatter(x[:, 0], x[:, 1], c=y_dbscan, cmap="viridis", s=50, alpha=0.7)

# Highlight outliers (label = -1)
outliers = (y_dbscan == -1)
plt.scatter(x[outliers, 0], x[outliers, 1],c="red", s=50, alpha=0.9, label="Outliers

plt.title("DBSCAN Clustering (Mall Customers)")
plt.xlabel("Annual Income (scaled)")
plt.ylabel("Spending Score (scaled)")
plt.legend()
plt.show()
```

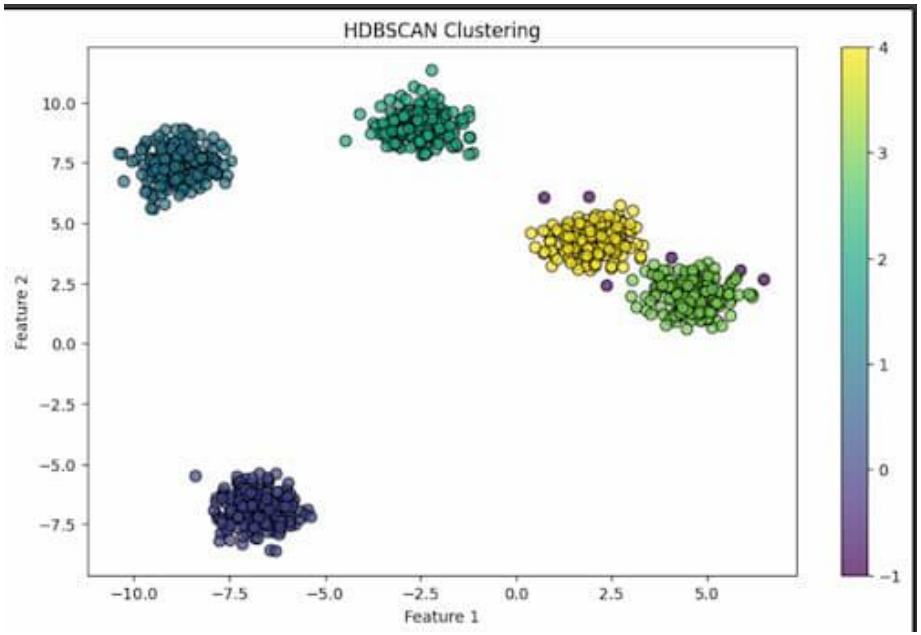


Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)

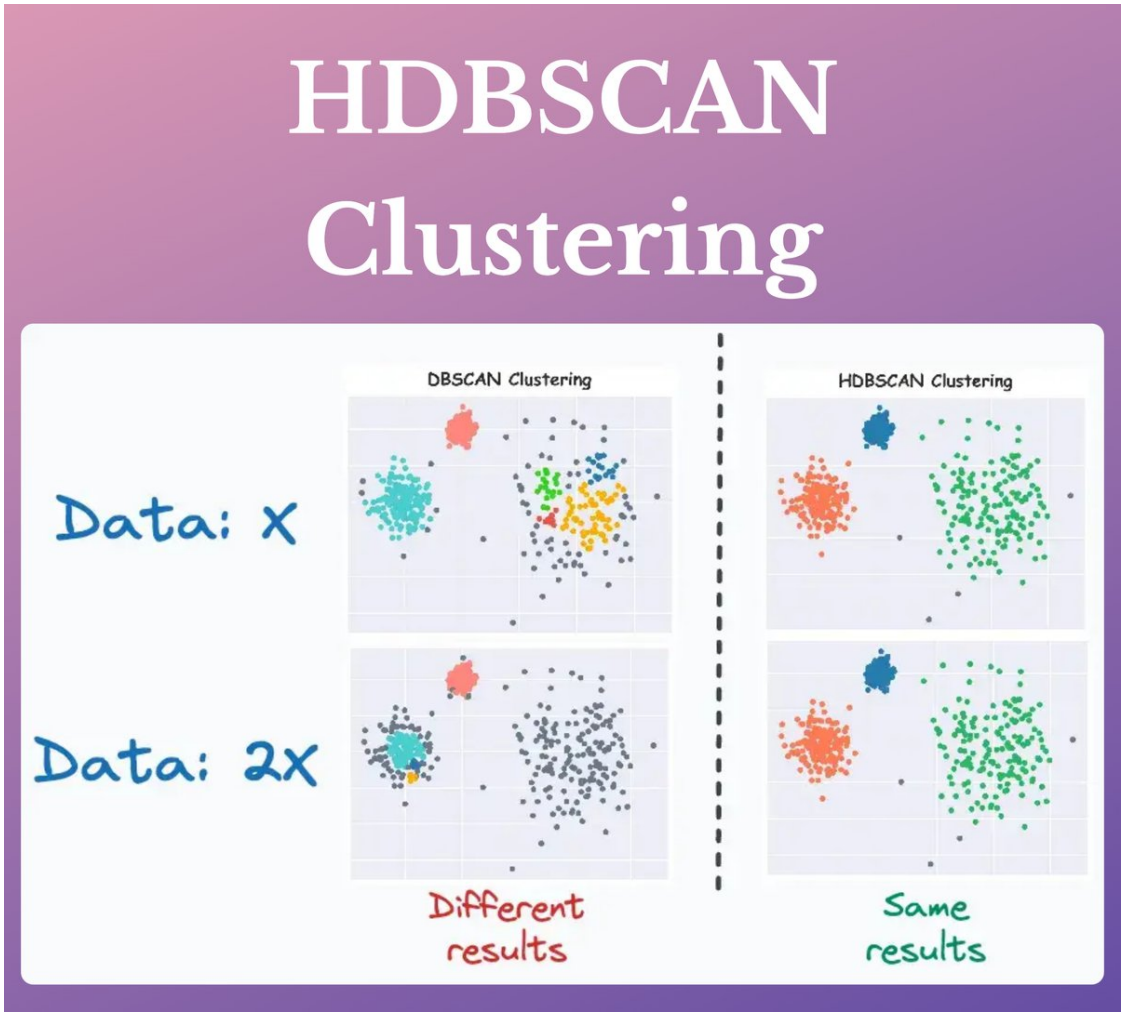
- HDBSCAN is a clustering algorithm that is designed to uncover clusters in datasets based on the density distribution of data points.
- Unlike some other clustering methods, it doesn't requires specifying the number of clusters in advance, making it more adaptable to different datasets.
- It uses high-density regions to identify clusters and views isolated or low-density points as noise.
- HDBSCAN is especially helpful for datasets with complex structures or varying.
- **Important Parameters :** 'min_cluster_size‘, 'min_samples‘, 'cluster_selection_epsilon‘, 'metric‘, 'cluster_selection_method‘, 'alpha, 'gen_min_span_tree, metric_params, algorithm, core_distance_n_jobs, allow_single_cluster

Libraries used:

- The necessary libraries that includes numpy, matplolib, hdbscan and metrics like silhouette_score, adjusted rand index for evaluating the metrics.



Differences between DB & HDB:



Advantages:

- Automatic cluster Discovery
- Handling Cluster Shapes
- Robust to noise and outliers
- Scalability

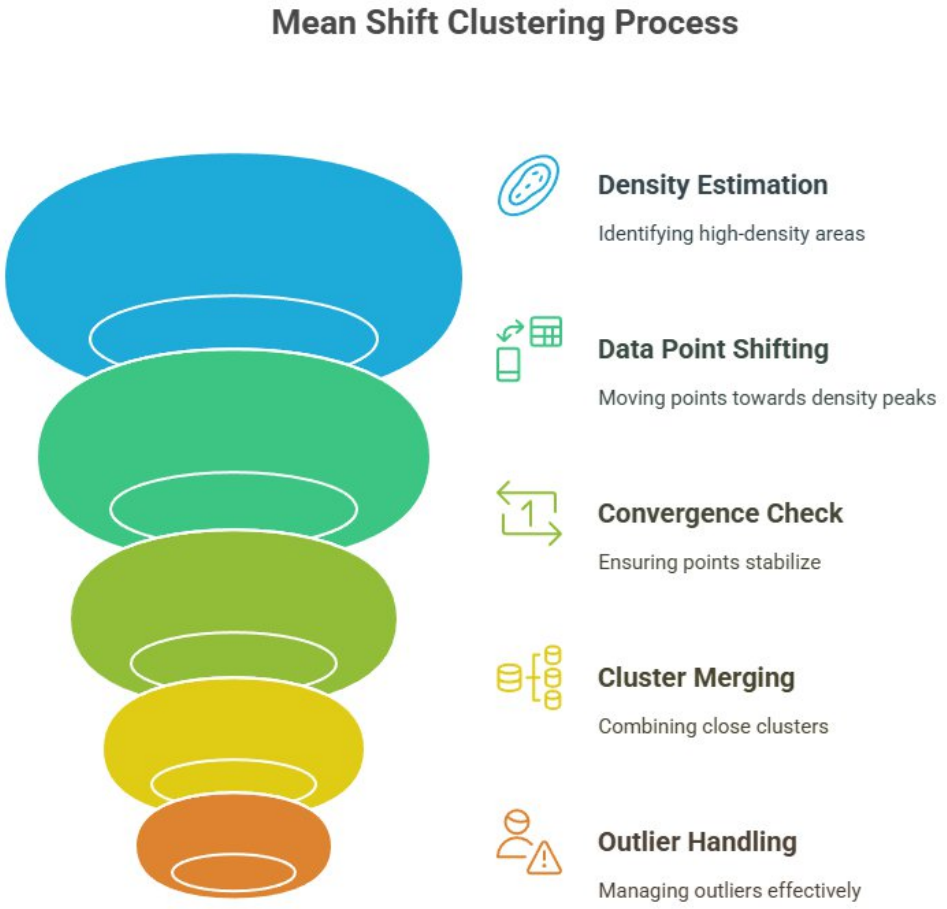
Disadvantages:

- Computationally Intensive
- Sensitive to Distance metric
- Parameter Sensitivity

Mean Shift Algorithm:

- Mean-shift clustering is a non-parametric, density-based clustering algorithm that can be used to identify clusters in a dataset.
- The basic idea behind mean-shift clustering is to shift each data point towards the mode (i.e., the highest density) of the distribution of points within a certain radius.
- The algorithm iteratively performs these shifts until the points converge to a local maximum of the density function. These local maxima represent the clusters in the data.

Process of mean-shift clustering algorithm

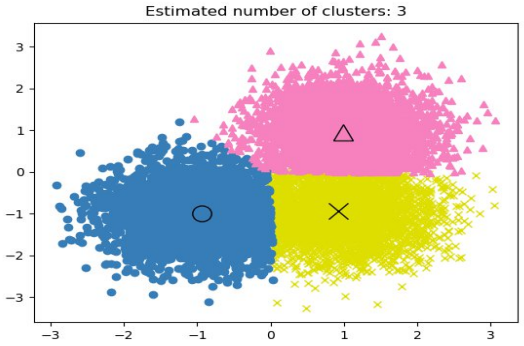


Advantages:

- It doesn’t requires number of cluster to be specified.
- It doesn’t make any assumptions about distribution of data.
- Model-free, doesn't assume any prior shape like spherical, elliptical, etc. on data clusters

Disadvantages:

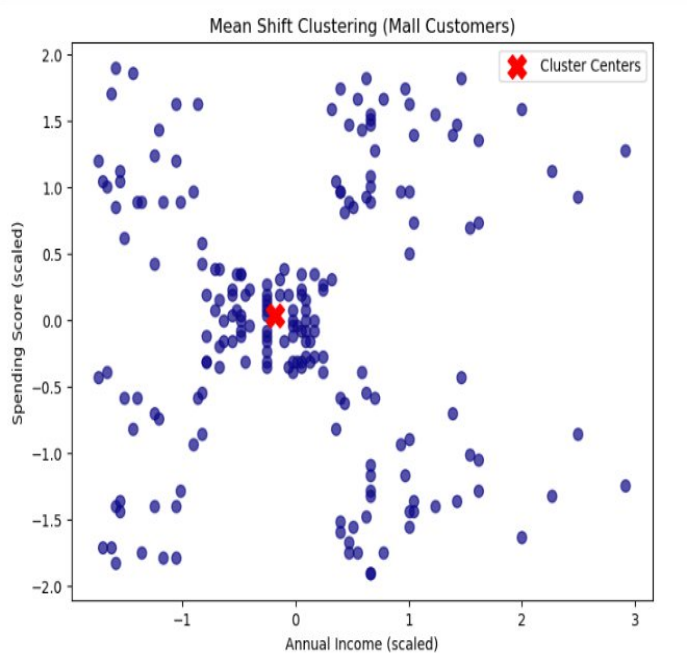
- Output depends on window size
- Computationally (relatively) expensive (approx 2s/image)
- Doesn't scale well with dimension of



```python
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
x_scaler = sc.fit_transform(x)
```

```python
from sklearn.cluster import MeanShift
ms = MeanShift()
y_ms = ms.fit_predict(x_scaler)
```

```python
# Plot clusters
plt.figure(figsize=(8,6))
plt.scatter(x_scaler[:,0], x_scaler[:,1], c=y_ms, cmap='plasma', s=50, alpha=0.7)

# Plot cluster centers
plt.scatter(ms.cluster_centers_[:,0], ms.cluster_centers_[:,1],
            c='red', marker='X', s=200, label='Cluster Centers')

plt.title("Mean Shift Clustering (Mall Customers)")
plt.xlabel("Annual Income (scaled)")
plt.ylabel("Spending Score (scaled)")
plt.legend()
plt.show()
```



OPTICS (Ordering Points To Identify the Clustering Structure)

- OPTICS (Ordering Points To Identify the Clustering Structure) is a density-based clustering algorithm similar to DBSCAN clustering.
- Unlike DBSCAN which struggles with varying densities.
- OPTICS does not directly assign clusters but instead creates a reachability plot which visually represents clusters.

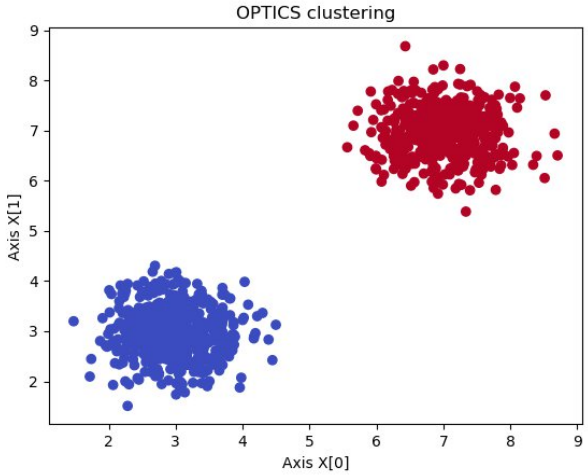
Reachability Plot

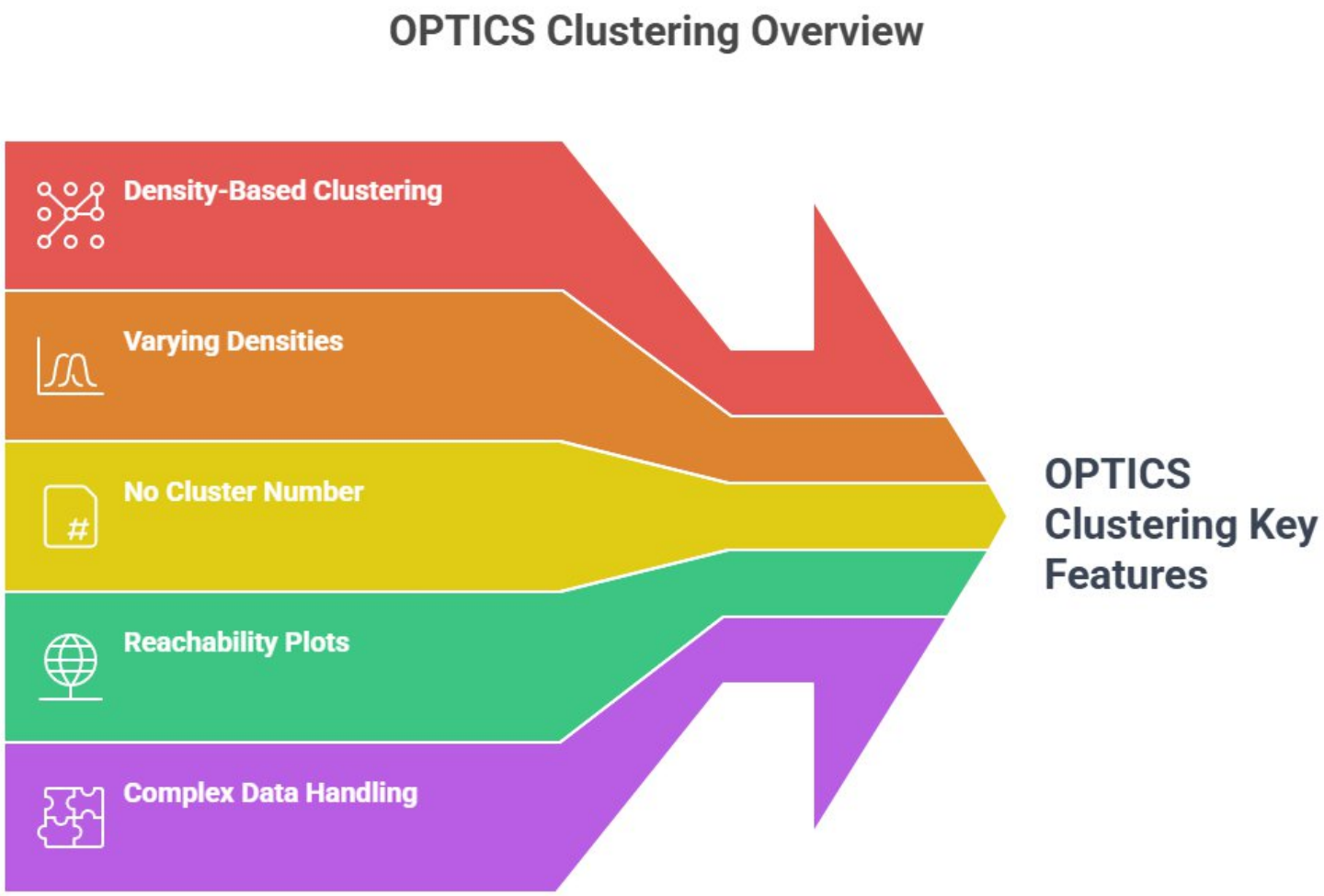
- A reachability plot is a graph that helps visualize clustering structures.
- It shows the reachability distance of each point in the dataset. It makes it ordered way based on how OPTICS processes them.

Key concepts in OPTICS:

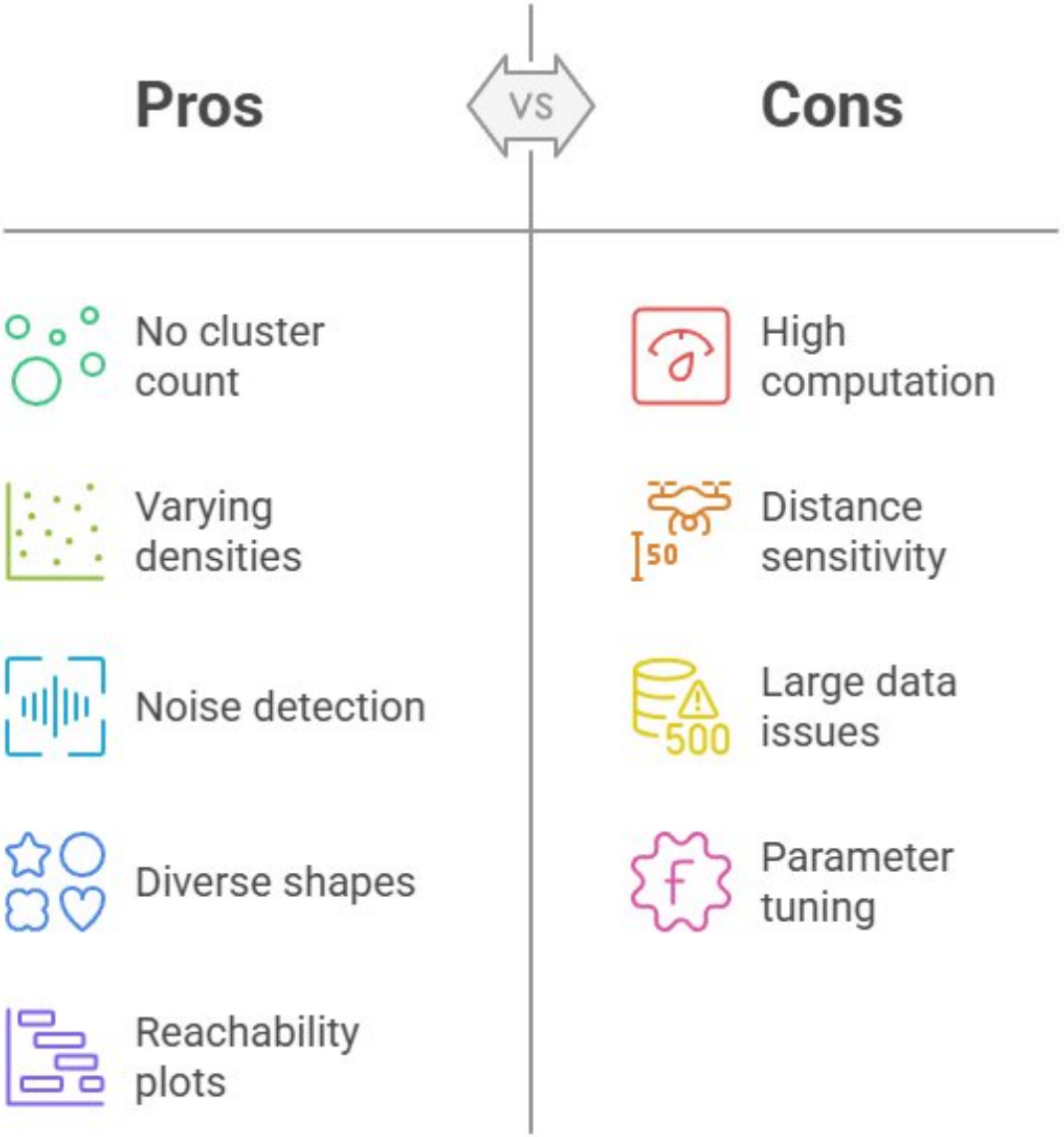
- **Core Distance** - Minimum distance needed for a point
- **Reachability Distance -** It is a measure of how difficult it is to reach from one point to another

OPTICS Algorithm Process



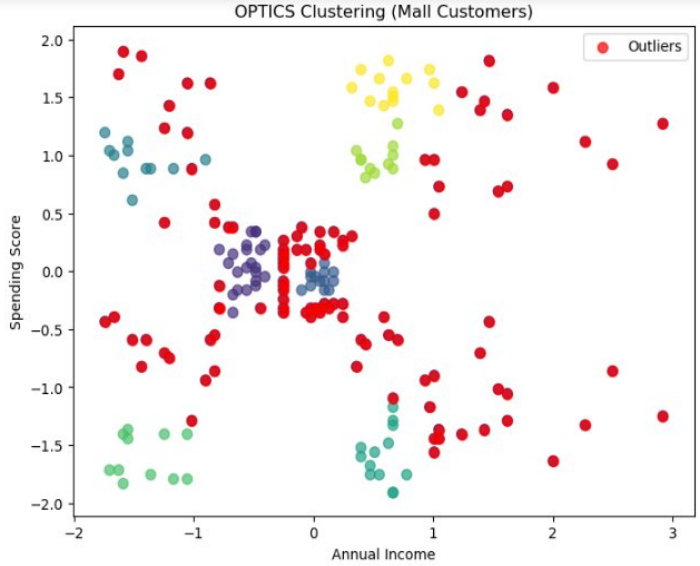

OPTICS Clustering Overview
Density-Based Clustering
Varying Densities
No Cluster Number
Reachability Plots
Complex Data Handling
OPTICS Clustering Key Features

OPTICS Clustering



```
]: from sklearn.cluster import OPTICS
optics = OPTICS(min_samples=5, xi=0.05, min_cluster_size=0.05)
y_optics = optics.fit_predict(x_scaler)
```

```
: # Plot clusters
plt.figure(figsize=(8,6))
plt.scatter(x_scaler[:,0], x_scaler[:,1], c=y_optics, cmap='viridis', s=50, alpha=0.7)

# Highlight outliers (label = -1)
outliers = (y_optics == -1)
plt.scatter(x_scaler[outliers,0], x_scaler[outliers,1], c='red', s=50, alpha=0.7, label='Outliers')

plt.title("OPTICS Clustering (Mall Customers)")
plt.xlabel("Annual Income")
plt.ylabel("Spending Score")
plt.legend()
plt.show()
```



BIRCH(Balanced Iterative Reducing and Clustering using Hierarchies) Algorithm:

- **BIRCH** is a clustering algorithm that can cluster large datasets by first generating a small and compact summary of the large dataset that retains as much information as possible.
- This smaller summary is then clustered instead of clustering the larger dataset.
- BIRCH is often used to complement other clustering algorithms by creating a summary of the dataset that the other clustering algorithm can now use.

Drawback:

- BIRCH has one major drawback - it can only process metric attributes.

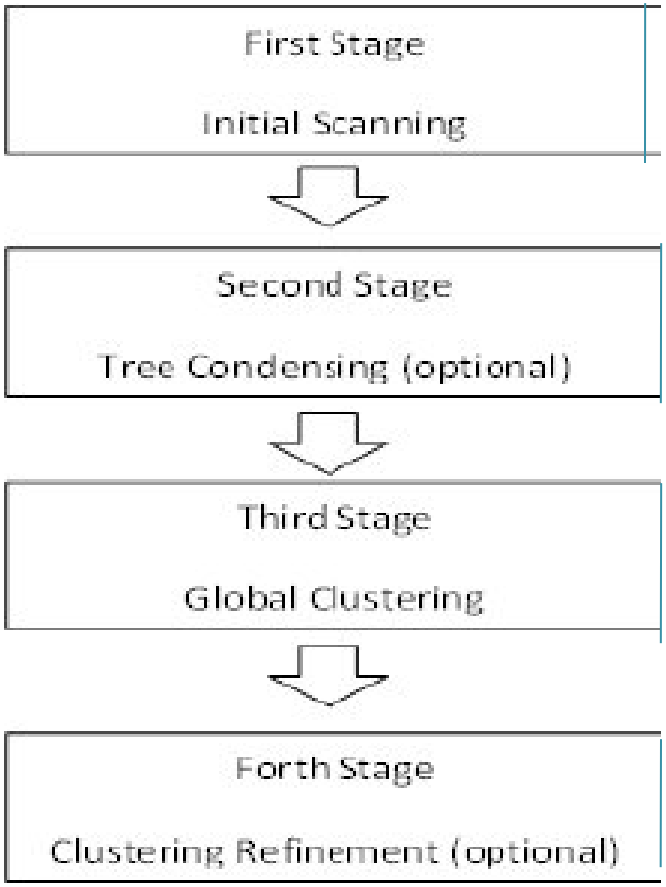
Two important terms:

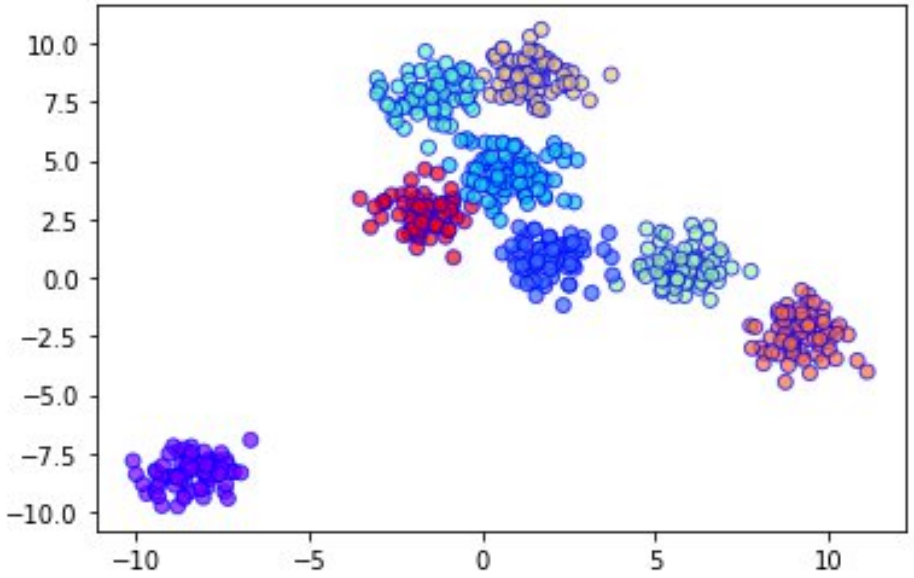
- **Clustering Feature (CF):** BIRCH summarizes large datasets into smaller, dense regions called Clustering Feature (CF) entries.
- **CF – Tree** : A CF tree is a tree where each leaf node contains a sub-cluster.

Parameters of BIRCH Algorithm :

1. **Threshold**
2. **branching_factor**
3. **n_clusters**

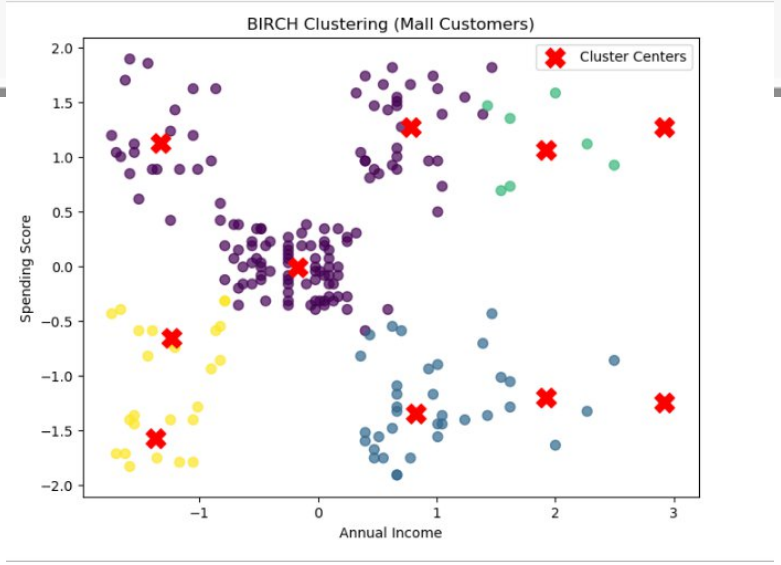
BIRCH Algorithm Process:





```python
from sklearn.cluster import Birch
birch = Birch(n_clusters =4)
y_birch = birch.fit_predict(x_scaler)
```

```python
# Plot clusters
plt.figure(figsize=(8,6))
plt.scatter(x_scaler[:,0], x_scaler[:,1], c=y_birch, cmap='viridis', s=50, alpha=0.7)

# Subclusters plot
if hasattr(birch, 'subcluster_centers_'):  #hasattr = allows us to check whether it has specific attributes or not.
    plt.scatter(birch.subcluster_centers_[:,0], birch.subcluster_centers_[:,1], c='red', marker='X', s=200, label='Cluster Cente

plt.title("BIRCH Clustering (Mall Customers)")
plt.xlabel("Annual Income")
plt.ylabel("Spending Score")
plt.legend()
plt.show()
```



Gaussian Mixture Model

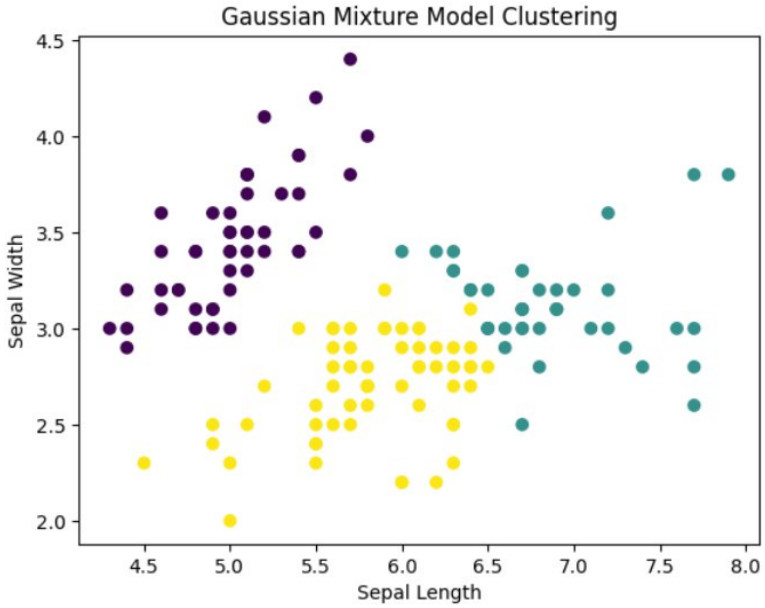
- Gaussian Mixture Model is a probabilistic model that assumes all data points are generated from a mixture of several Gaussian distributions with unknown parameters.
- GMMs perform soft clustering meaning each data point belongs to multiple clusters with certain probabilities.
- Each Gaussian in the mixture is defined by:
  - **Mean (μ)**: The center of the distribution.
  - **Covariance (Σ)**: Describes the spread and orientation.
  - **Mixing coefficient (π)**: Represents the proportion of each Gaussian in the mixture.

Steps involved:

- Gaussian works in two main steps:

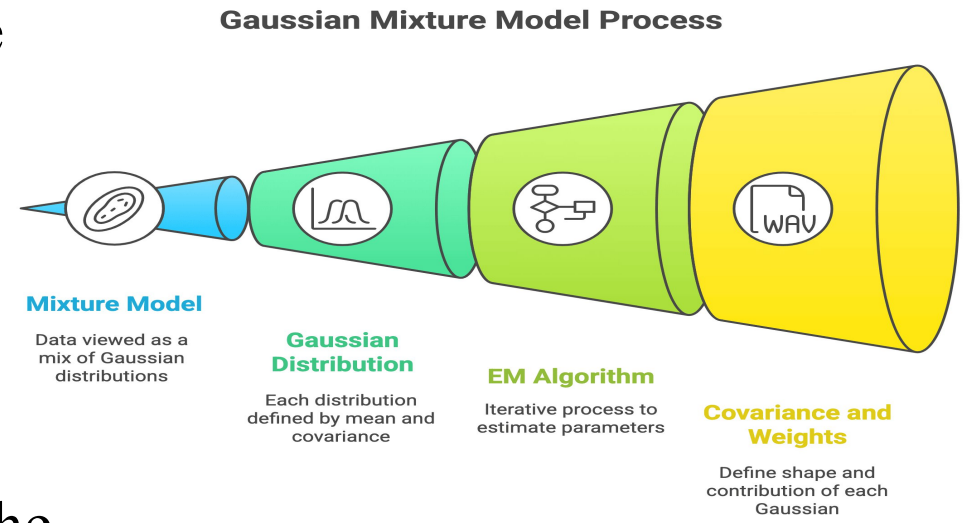
  i) Expectation Step (E-step): In this step the algorithm calculates the probability that each data point belongs to each cluster based on the current parameter estimates (mean, covariance, mixing coefficients).

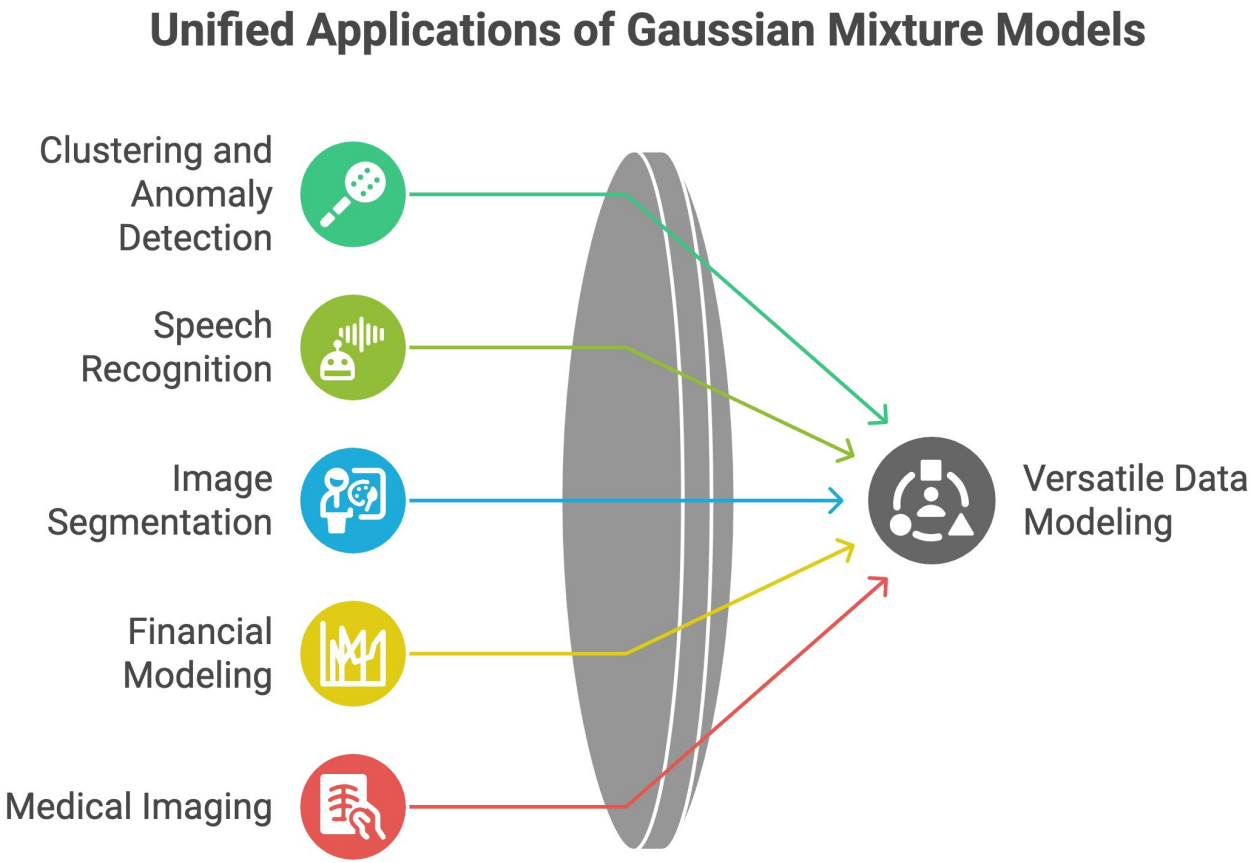
  ii) Maximization Step (M-step): After estimating the probabilities the algorithm updates the parameters (mean, covariance and mixing coefficients) to better fit the data.



Simple breakdown of the GMM process:

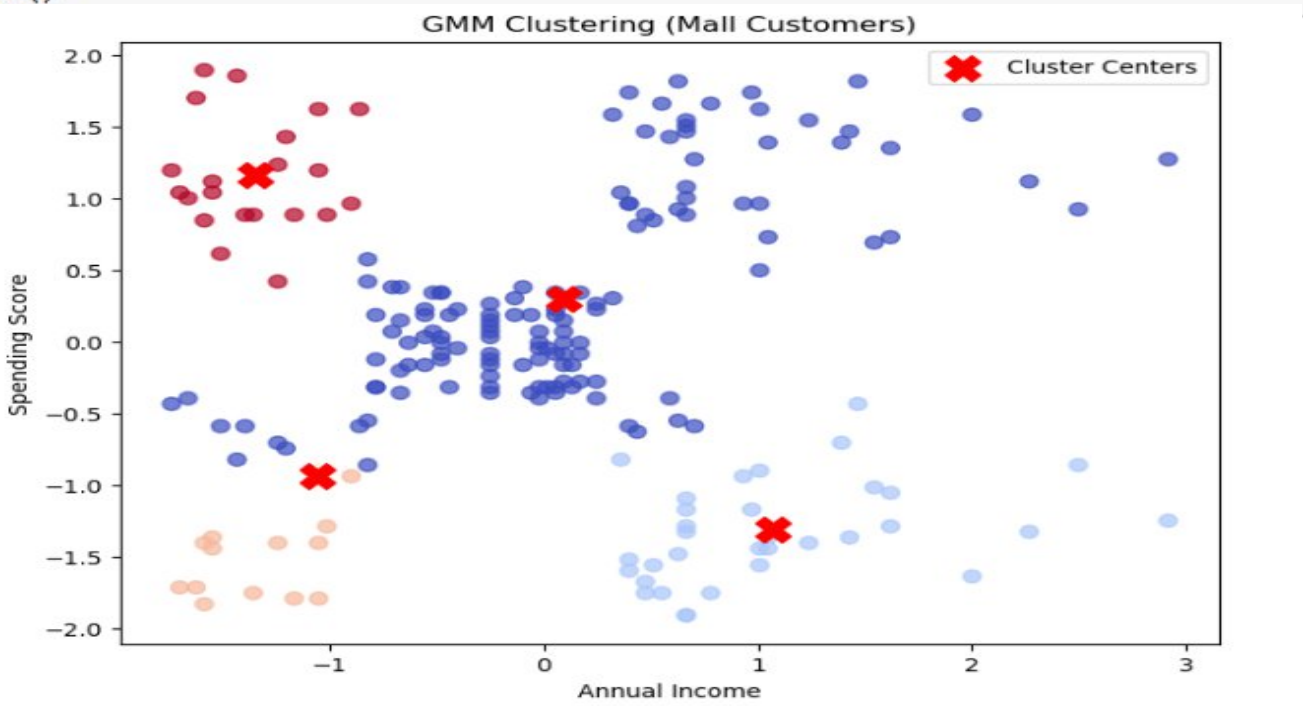
- **Initialization -** Start with initial guesses
- **E-step -** calculate the probability of each cluster.
- **M-step -** Update the parameters.
- **Repeat -** Continue alternating between the E-step and M-step




Unified Applications of Gaussian Mixture Models
Clustering and Anomaly Detection
Speech Recognition
Image Segmentation
Financial Modeling
Medical Imaging
Versatile Data Modeling

```
: from sklearn.mixture import GaussianMixture
  gmm = GaussianMixture(n_components=4, random_state=42)
  y_gmm = gmm.fit_predict(x_scaler)
```

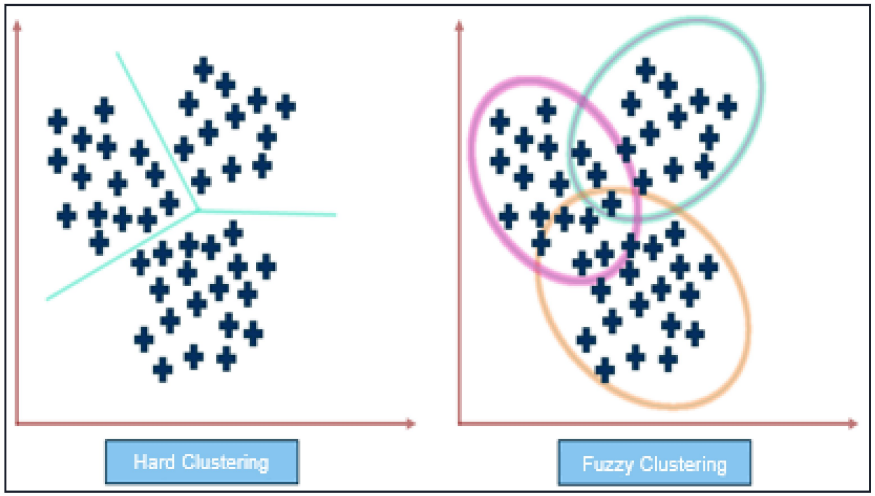
```
: # Plot clusters
  plt.figure(figsize=(8,6))
  plt.scatter(x_scaler[:,0], x_scaler[:,1], c=y_gmm, cmap='coolwarm', s=50, alpha=0.7)

  # Highlight Cluster centers
  plt.scatter(gmm.means_[:,0], gmm.means_[:,1], c='red', marker='X', s=200, label='Cluster Centers')
  plt.title("GMM Clustering (Mall Customers)")
  plt.xlabel("Annual Income")
  plt.ylabel("Spending Score")
  plt.legend()
  plt.show()
```



Fuzzy Clustering

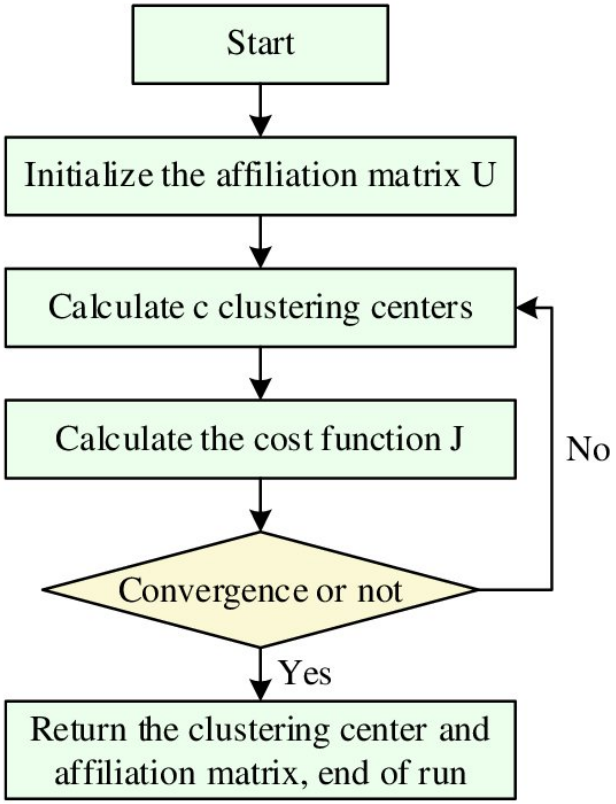
- Fuzzy Clustering also known as soft clustering, is a type of unsupervised machine learning that allows data points belonging to multiple clusters with varying degrees of membership.
- It assigns membership values between 0 and 1 for every cluster.
- It indicates the degree to which a data points belongs to a particular cluster.

- Unlike traditional clustering (like K-Means) where each data points belongs to only one cluster.



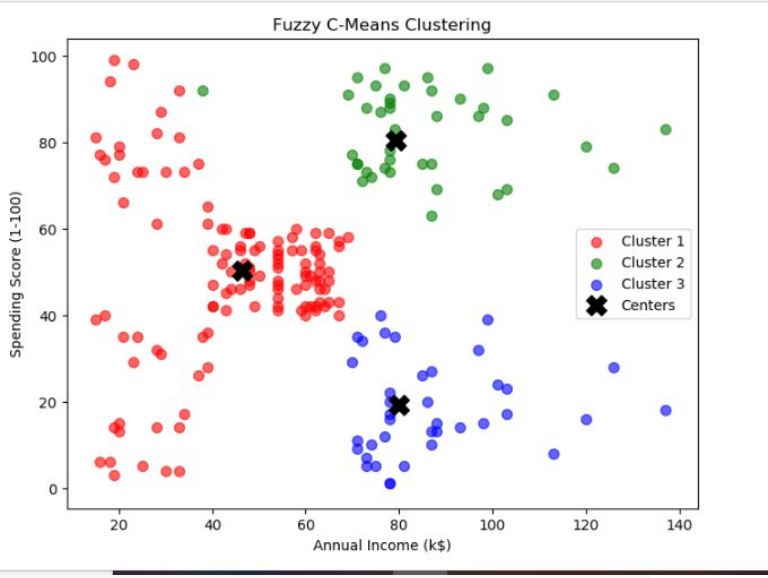
Steps involved in this algorithm:

1. Initialize membership values randomly
2. Compute cluster centroids
3. Calculate distance between Data points and Centroids
4. Update membership values
5. Repeat until convergence
6. Defuzzification(optional)



```
import skfuzzy as fuzz
n_clusters = 3
cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    X_scaled.T,          # transpose because skfuzzy expects featur
    c=n_clusters,        # number of clusters
    m=2,                 # fuzziness parameter
    error=0.005,         # stopping criterion
    maxiter=1000,        # max iterations
    init=None)
print("Labels shape:", labels.shape)   # should match number of row
print("Fuzzy Partition Coefficient (FPC):", fpc)
```

```
Labels shape: (200,)
Fuzzy Partition Coefficient (FPC): 0.6802115367982321
```




```
labels = np.argmax(u, axis=0)
plt.figure(figsize=(8,6))
colors = ['red', 'green', 'blue']  # add more if more clusters

for j, color in enumerate(colors[:n_clusters]):
    plt.scatter(X[labels == j, 0], X[labels == j, 1],c=color,label=f"Cluster {j+1}",s=50, alpha=0.6)

# Rescale cluster centers back to original space
centers = scaler.inverse_transform(cntr)
plt.scatter(
    centers[:, 0], centers[:, 1],c='black', marker='X', s=200, label='Centers'
)

plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title("Fuzzy C-Means Clustering")
plt.legend()
plt.show()
```

GitHub Link:

https://github.com/Geetharani-CodeAI/Clustering-Algorithm