CS331 Programming Languages Lab

# Clustering Visualizations

Pradeep Kumar 200101080
Prakhar Pandey 200101081
Viraj Goyanka 200101107
Siddharth Hemant Khincha 200101094
Tanveen 200101098

## Instructions to Run

Open the terminal and move into the directory containing the project.

Enter the following :

**pip install -r requirements.txt**

Now all the requirements are satisfied, so run the following command to execute the program :

**python ui.py**

## Introduction

We have created a project depicting three different clustering algorithms. The entire project was coded in python. Here is a short description of the three algorithms we have executed.

### Affinity Propagation

**Parameters**:

**Damping factor**: a value between 0.5 and 1 that controls the influence of incoming messages on the current exemplar preferences.

**Maximum iterations**: a positive integer that limits the number of iterations allowed in the algorithm.

**Meaning**:

Affinity Propagation uses a similarity measure to calculate the affinity between data points, which can be any distance metric such as Euclidean or cosine similarity. The damping factor controls the convergence speed and stability of the algorithm. The maximum iterations parameter limits the number of iterations allowed in the algorithm to prevent it from running indefinitely.

**Applications:**

Affinity Propagation is useful in a variety of applications, such as:

1. Image and video segmentation

2. Natural language processing

3. Social network analysis

4. Gene expression analysis

5. Robotics

**Limitations:**

Some limitations of Affinity Propagation are:

1. Sensitivity to the choice of parameters: Choosing the correct damping factor can be challenging and may require trial and error.

2. Computational complexity: The algorithm has a quadratic time complexity, which can make it computationally expensive for large datasets.

3. Limited scalability: The algorithm may not be suitable for datasets with a high-dimensional feature space or a large number of data points.

**Algorithm:**

The algorithm uses a similarity measure to calculate the affinity between data points and iteratively updates two matrices: the responsibility matrix and the availability matrix.

The responsibility matrix, denoted by R(i,k), represents the amount of responsibility that data point k takes for point i. The availability matrix, denoted by A(i,k), represents the availability of a data point i to become an exemplar, given that k is already an exemplar.

The algorithm starts by initializing both matrices to zero. It then iteratively updates the matrices until convergence is reached or a maximum number of iterations is exceeded. At each iteration, the responsibility and availability matrices are updated as follows:

**R(i,k) = s(i,k) - max {a(i,k') + s(i,k')} where k' != k**

Update the responsibility matrix: For each pair of data points (i,k), update the responsibility matrix using the following formula:

Here, s(i,k) represents the similarity between data points i and k. The max term represents the sum of the current availability and responsibility of all other data points except k.

Update the availability matrix: For each pair of data points (i,k), update the availability matrix using the following formula:

**A(i,k) = min {0, R(k,k) + sum_{i' != i, i' != k} max(0, R(i',k))}**

Here, the max term represents the maximum responsibility that data point i' takes for point k among all other points, excluding i and k. The min term ensures that A(k,k) remains non-positive.

Dampen the matrices: To prevent oscillations, the responsibility and availability matrices are dampened using a damping factor α (usually set between 0.5 and 1

**R(i,k) = (1 - α) * R(i,k) + α * R_new(i,k)**

**A(i,k) = (1 - α) * A(i,k) + α * A_new(i,k)**

Check for convergence: The algorithm checks for convergence by computing the difference between the old and new matrices. If the difference is below a certain threshold, the algorithm terminates.

Once the algorithm has converged, it identifies the exemplars by selecting the data points with the highest net responsibility:

**exemplars = {i | max_k(R(i,k) + A(i,k))}**

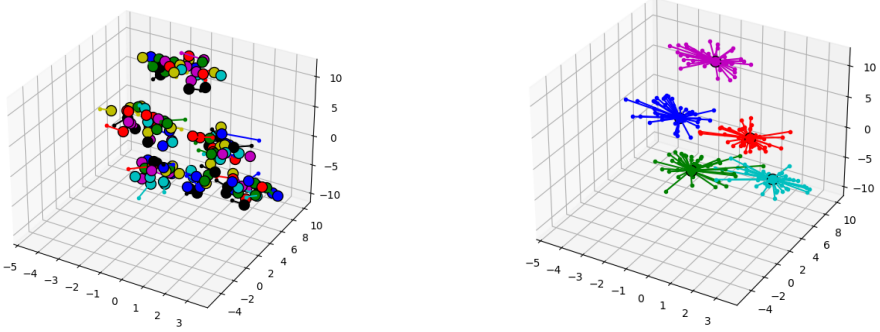
Finally, it assigns each non-exemplar data point to the nearest exemplar based on its net similarity:

**cluster(i) = argmax_k(R(i,k) + A(i,k)), i not in exemplars**

The output of Affinity Propagation is a set of exemplars and their corresponding clusters. The exemplars are the data points that best represent each cluster, while non-exemplar points are assigned to the nearest exemplar based on their net similarity.

**Example**:

Here is a screenshot of the result from our code and a dataset



## DBScan Algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a popular clustering algorithm that groups together data points that are closely packed together,

while also identifying noise points that do not belong to any cluster. The algorithm works by defining a minimum number of points required to form a dense region (density) and then expanding clusters from those regions.

**Parameters**:

**Epsilon (ε):** The radius of the neighborhood around each data point. Points within this radius are considered neighbors.

**MinPts**: The minimum number of points required to form a dense region. Points with at least MinPts neighbors are considered core points, and regions with at least MinPts core points are considered dense regions.

**Algorithm:**

1. Randomly select an unvisited data point.

2. Find all points within ε distance of the selected point, forming a new cluster if the point is a core point (has at least MinPts neighbors).

3. Expand the cluster by recursively finding all points within ε distance of the core points.

4. Mark all visited points as part of a cluster.

5. Repeat steps 1-4 until all points have been visited.

The output of the algorithm is a set of clusters, where each cluster contains a set of closely packed data points, and noise points that do not belong to any cluster.

**Applications:**

DBSCAN has many applications in various fields, including image analysis, natural language processing, and anomaly detection. It is commonly used when the underlying structure of the data is not known, or when there is a mix of dense and sparse regions.

**Limitations:**

Sensitivity to parameter selection: The performance of DBSCAN is highly dependent on the choice of ε and MinPts parameters, which can be difficult to choose in practice.

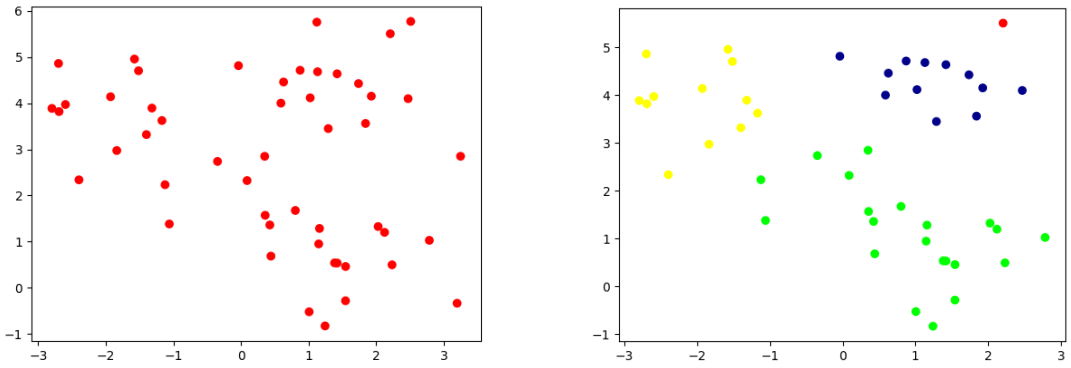
Difficulty handling datasets with varying density: The algorithm assumes that clusters have a uniform density, which can be a limitation for datasets with varying density.

Difficulty handling high-dimensional data: The curse of dimensionality can affect the performance of DBSCAN in high-dimensional data.

In summary, DBSCAN is a powerful clustering algorithm that can effectively handle noise and identify arbitrary-shaped clusters in the data. However, it has some limitations that should be considered when applying it to real-world problems.

**Example**:

Here is a screenshot of the result from our code and a dataset



## KMeans Algorithm

K-Means is a popular clustering algorithm that seeks to partition a dataset into K distinct clusters. The algorithm works by iteratively reassigning data points to the closest centroid and updating the centroid to the mean of the assigned data points.

**Parameters**:

**K**: The number of clusters to form.

**Maximum iterations**: The maximum number of iterations to run the algorithm before stopping.

**Algorithm**:

1. Initialize K centroids randomly.

2. Assign each data point to the closest centroid.

3. Update the centroids to the mean of the assigned data points.

4. Repeat steps 2-3 until convergence is reached or the maximum number of iterations is exceeded.

The algorithm converges when the assignments of data points to centroids no longer change or the change is below a certain threshold. The output of the algorithm is the final set of K centroids and their corresponding clusters.

**Applications:**

K-Means has **many applications** in various fields, including customer segmentation, image segmentation, anomaly detection, and natural language processing. It is commonly used in data preprocessing and exploratory data analysis.

**Limitations:**

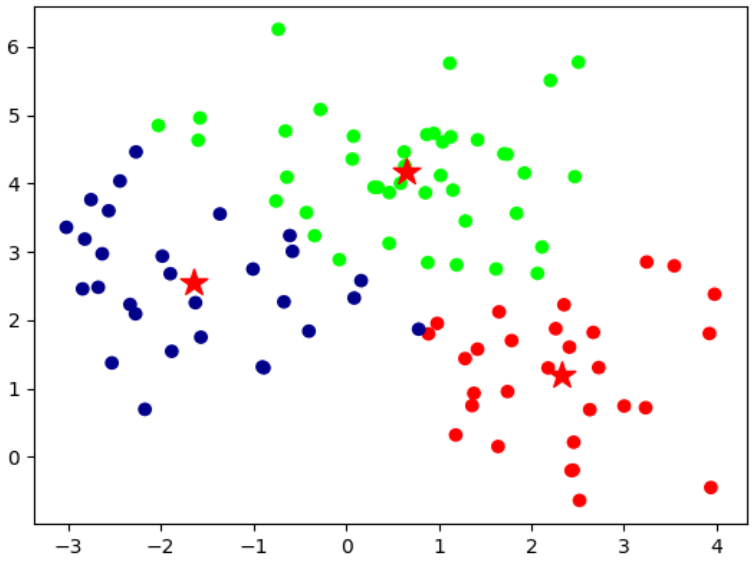
1. Sensitivity to initial centroids: The algorithm can converge to different local optima depending on the initial positions of the centroids.

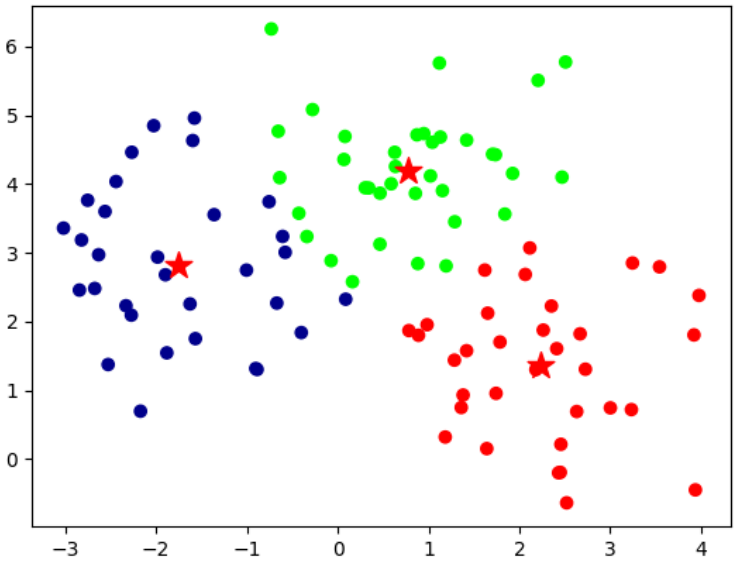
2. Sensitivity to outliers: Outliers can significantly affect the position of the centroids and the resulting clusters.

3. Difficulty handling non-linearly separable clusters: The algorithm assumes that the clusters are linearly separable, which can be a limitation for complex datasets.

**Example**:

Here is a screenshot of the result from our code and a dataset


6
5
4
3
2
1
0
−3
−2
−1
0
1
2
3
4


6
5
4
3
2
1
0
−3
−2
−1
0
1
2
3
4