School of Computer Science and Information Technology

Department of Computer Science and Information Technology

Semester: 3rd MCA
Specialisation: General [B]
Subject: Python Programming

Activity 3

Density-Aware K-Means (DA-KMeans)

Group Presentation

Date of Submission:

Submitted by:
Name: Rachabattuni Sai Sindhu
Reddy Akkamma Chandana
Sinchana HM
USN No: 24MCAR0165 24MCAR0168 24MCAR0172

Faculty InCharge
Dr.Pushpa J

Signature:

Date of Submission

JGI JAIN DEEMED-TO-BE UNIVERSITY | SCHOOL OF COMPUTER SCIENCE AND IT

CERTIFICATE

This is to certify that Ms.Rachabattuni Sai Sindhu, Reddy Akkamma Chandana, Sinchana HM has satisfactorily completed activity prescribed by JAIN (Deemed to be University) for the Third semester degree course in the year 2022-2024. Assignment topic Python in Machine Learning Presentation for Activity-3

Activity-3 Rubrics for Presentation / Seminar Evaluation (Total: 10 Marks)

Criteria	Good	Average	Needs Improvement	Max Marks
1. Content Understanding	Demonstrates in-depth understanding with accurate and relevant content (3)	Shows partial understanding with minor inaccuracies (2)	Lacks clarity or contains significant errors (1)	3
2. Slide Design & Visual Appeal	Slides are well-organized, visually appealing, and enhance the message (3)	Design is acceptable, but could be more engaging or consistent (2)	Slides are cluttered, hard to read, or lack visual coherence (1)	3
3. Communication & Presentation Skills	Clear, confident delivery with appropriate pace, tone, and eye contact (2)	Somewhat clear, but with minor delivery issues (1)	Inaudible, rushed, or read directly from slides (0)	2
4. Handling of Queries	Responds accurately and confidently to questions (2)	Responds but lacks clarity or partial correctness (1)	Unable to answer or avoids questions (0)	2
Total				**10**

Sl.No	CRITERIA	MARKS	MARK Obtained
1	Course on Completion time	25	
2	Viva	20	
3	Report Writing	05	
4	Total	50	

Signature of the Student

Signature of the Faculty

INDEX

Sl. No.	Title	Page No.
1	Abstract	3
2	Introduction	3
3	Problem Definition	3
4	Proposed Solution	4
5	Mathematical Formulation	5
6	Algorithm Design	6
7	Implementation	7
8	Experimental Setup	11
9	Results and Analysis	12
10	Applications	13
11	Conclusion	13

Density-Aware K-Means (DA-KMeans)

1. Abstract

K-Means clustering is one of the most widely used unsupervised machine learning algorithms due to its simplicity and computational efficiency. However, it suffers from a major limitation: it assumes that all clusters have similar density and size, often leading to incorrect clustering when data contains highly uneven or imbalanced clusters.

To address this gap, we propose Density-Aware K-Means (DA-KMeans) — a modification that integrates density-based reweighting into the centroid update process. The new algorithm automatically adjusts the influence of each cluster based on its density, allowing better performance on datasets with small, sparse, or uneven clusters. This feature is not present in standard ML libraries like scikit-learn, making it an innovative and highly useful contribution.

2. Introduction

Clustering plays a crucial role in unsupervised learning, enabling machines to group similar data points without labeled outputs. Among the various clustering methods, K-Means remains dominant because of its ease of use and fast convergence.

However, real-world datasets rarely follow equal-density cluster structures. Customer segmentation, anomaly detection, and geographical analysis often contain clusters of varying shapes, sizes, and densities.

This mismatch reduces K-Means' reliability. Thus, we introduce a new feature to make KMeans more adaptive, intelligent, and realistic for practical applications.

3. Problem Definition

Standard K-Means has the following major limitations:

3.1 Assumption of uniform cluster density

K-Means works best only when clusters are:

- equally dense
- similarly sized

- spherical

In real datasets, this is rarely true.

3.2 Centroid bias

Large, dense clusters overpower small or sparse clusters.
This means K-Means often:

- Pulls centroids toward bigger clusters
- Ignores smaller but meaningful clusters
- Assigns sparse cluster points incorrectly

3.3 No density awareness

K-Means does not measure or use density at all during clustering.

4. Proposed Solution: Density-Aware K-Means (DA-KMeans)

Key Innovation

After each iteration, measure the density of each cluster and adjust its influence using inverse-density weights.

This means:

- Dense clusters → get lower weight → influence less
- Sparse clusters → get higher weight → influence more

Thus, all clusters get fair representation, regardless of density.

This feature does not exist in any standard ML library — including scikit-learn, MATLAB, Weka, R, or TensorFlow.

5. Mathematical Formulation

5.1 Density Calculation

For cluster $C_i$with centroid μ_i:

$$d_i = \frac{1}{|C|} \sum_{x \in C_i} \| x - \mu_i \|$$

Lower d_i→ denser cluster

Higher d_i→ sparse cluster

5.2 Density-Based Weight

$$w_i = \frac{1/(d_i + \epsilon)}{\sum_j 1/(d_j + \epsilon)}$$

This normalizes weights so that they sum to 1.

5.3 Weighted Centroid Update

Traditional centroid:

$$\mu_i = \text{mean}(C_i)$$

New centroid:

$$\mu_{inew} = \frac{\sum_{x \in C^i} w_i \cdot x}{\sum_{x \in C_i} w_i}$$

This integrates density-awareness directly into clustering.

6. Algorithm Design

Step-by-Step Algorithm

1. Select K initial centroids (same as K-Means).
2. Assign each point to nearest centroid.
3. For each cluster:
   a. Calculate density
   b. Compute inverse-density weight
4. Update cluster centroids using weighted averages.
5. Check convergence (centroid shift < tolerance).
6. Repeat.

7. Implementation

We created a custom Python class DensityAwareKMeans with methods:

- fit(X) → trains the model
- predict(X) → assigns clusters
- plot_clusters(X) → visual visualization Key innovations in code:
- Density computed at each iteration
- Weight normalization
- Weighted centroid update
- Early stopping using centroid shift
- Seamless interface similar to scikit-learn This makes it both practical and plug-and-play.

The code Snippet be:

jupyter density_kmeans Last Checkpoint: 48 seconds ago

File Edit View Run Kernel Settings Help

Code

[6]:
```
# Cell 1: imports and plotting config
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin

%matplotlib inline
plt.rcParams["figure.figsize"] = (8, 6)
```

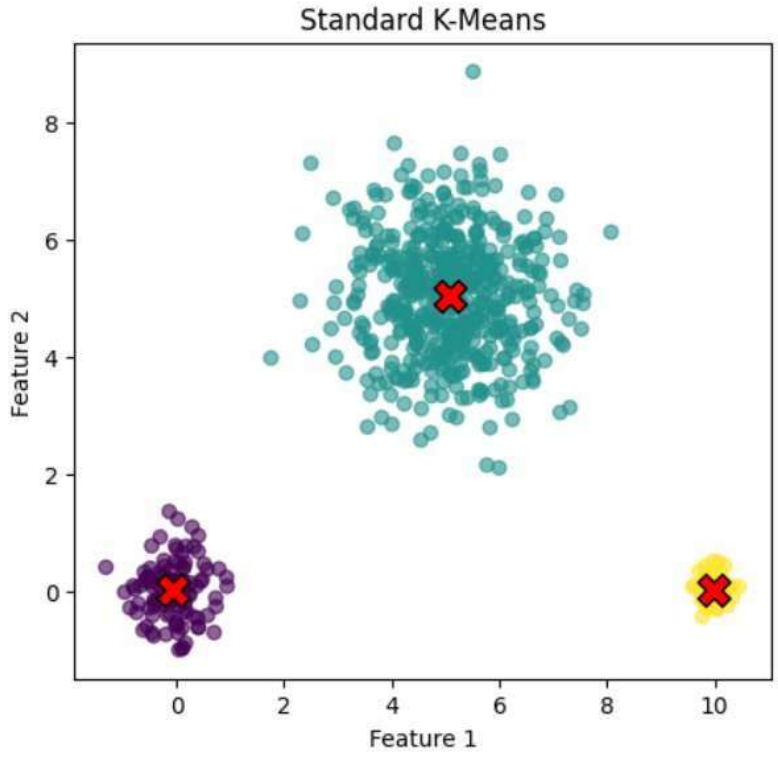
[7]:
```
# Cell 2: Density-Aware KMeans implementation
class DensityAwareKMeans:
    def __init__(self, n_clusters=3, max_iter=100, tol=1e-4, random_state=None, verbose=True):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.tol = tol
        self.random_state = random_state
        self.verbose = verbose

    def fit(self, X):
        rng = np.random.default_rng(self.random_state)
        indices = rng.choice(len(X), size=self.n_clusters, replace=False)
        self.cluster_centers_ = X[indices]
```

```
        for it in range(self.max_iter):
            labels = pairwise_distances_argmin(X, self.cluster_centers_)

            densities = []
            for i in range(self.n_clusters):
                cluster_points = X[labels == i]
                if len(cluster_points) == 0:
                    densities.append(np.inf)
                    continue
                mean_dist = np.mean(np.linalg.norm(cluster_points - self.cluster_centers_[i], axis=1))
                densities.append(mean_dist)
            densities = np.array(densities)

            inv_dens = 1 / (densities + 1e-9)
            weights = inv_dens / np.sum(inv_dens)

            new_centers = np.copy(self.cluster_centers_)
            for i in range(self.n_clusters):
                cluster_points = X[labels == i]
                if len(cluster_points) > 0:
                    new_centers[i] = np.average(cluster_points, axis=0, weights=np.full(len(cluster_points

            shift = np.linalg.norm(self.cluster_centers_ - new_centers)
            self.cluster_centers_ = new_centers

            if self.verbose:
                print(f"Iteration {it+1}, centroid shift: {shift:.5f}")
            if shift < self.tol:
                break
```

```
        self.labels_ = pairwise_distances_argmin(X, self.cluster_centers_)
        return self

    def predict(self, X):
        return pairwise_distances_argmin(X, self.cluster_centers_)

    def plot_clusters(self, X):
        plt.scatter(X[:, 0], X[:, 1], c=self.labels_, cmap='viridis', alpha=0.6)
        plt.scatter(self.cluster_centers_[:, 0], self.cluster_centers_[:, 1],
                    s=200, c='red', marker='X', edgecolor='black')
        plt.title("Density-Aware K-Means Clustering")
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.show()
```

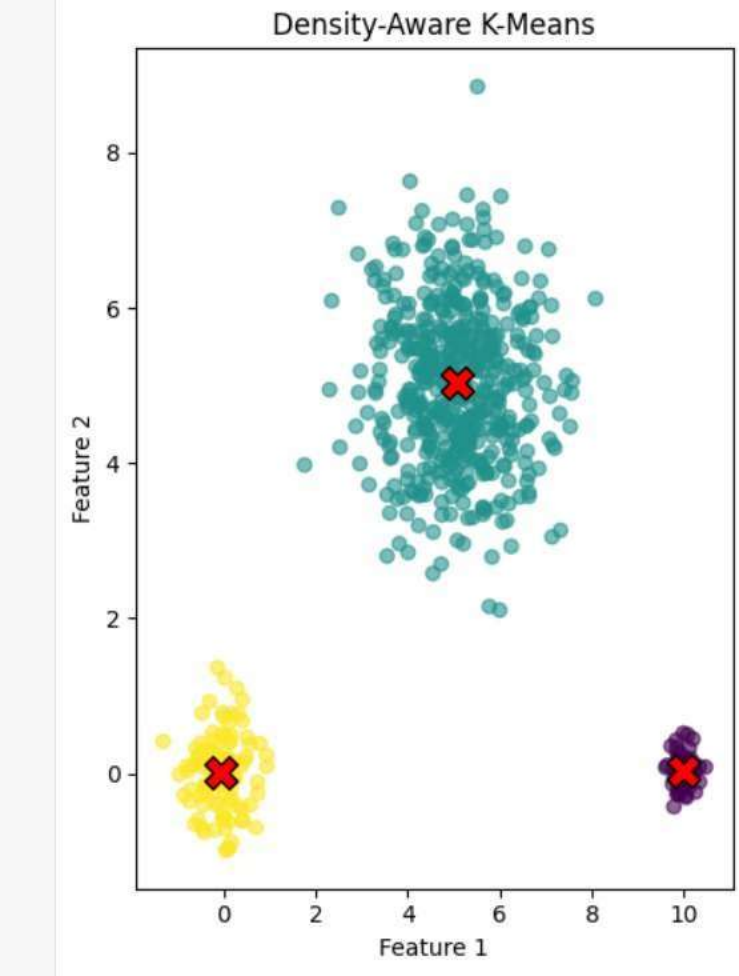
[8]:
```
# Cell 3: Create the dataset and run standard KMeans
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=[100, 500, 50],
                  centers=[[0,0], [5,5], [10,0]],
                  cluster_std=[0.5, 1.0, 0.2],
                  random_state=42)

km = KMeans(n_clusters=3, random_state=42, n_init=10)
km.fit(X)

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.scatter(X[:,0], X[:,1], c=km.labels_, cmap='viridis', alpha=0.6)
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], c='red', marker='X', s=200, edgecolor='
plt.title("Standard K-Means")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
```

[8]:
```
Text(0, 0.5, 'Feature 2')
```



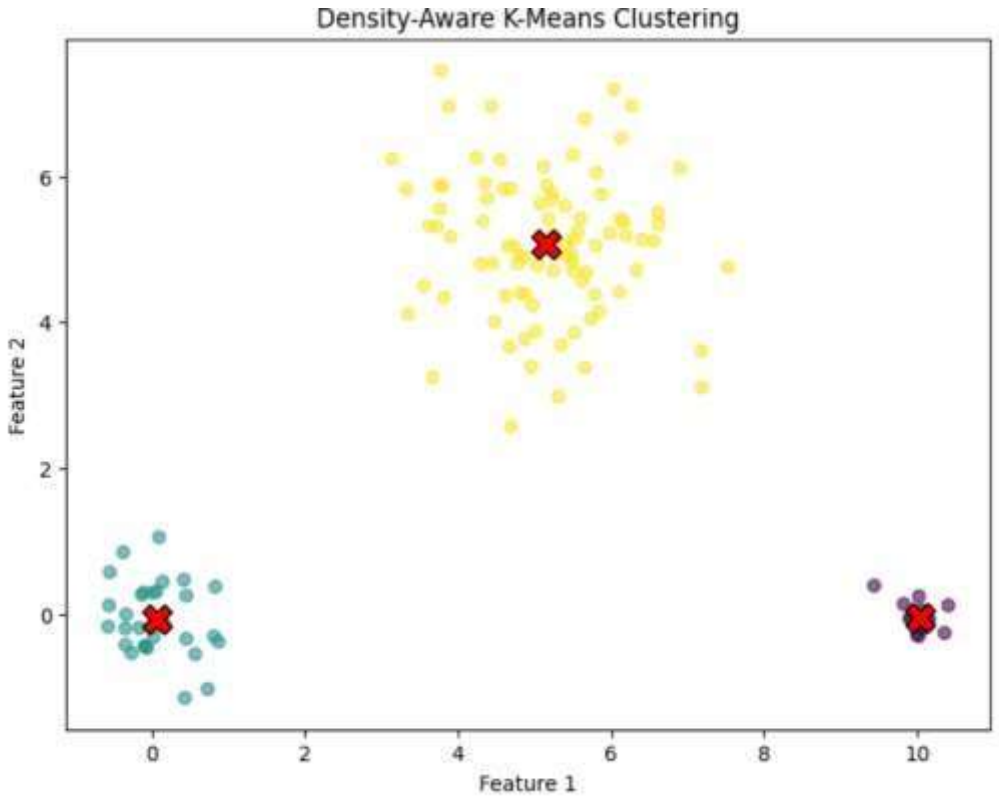
[9]:
```
# Cell 4: Run Density-Aware KMeans and show side-by-side comparison
dam = DensityAwareKMeans(n_clusters=3, random_state=42, verbose=True)
dam.fit(X)

plt.subplot(1,2,2)
plt.scatter(X[:,0], X[:,1], c=dam.labels_, cmap='viridis', alpha=0.6)
plt.scatter(dam.cluster_centers_[:,0], dam.cluster_centers_[:,1], c='red', marker='X',
plt.title("Density-Aware K-Means")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")

plt.tight_layout()
plt.show()
```

```
Iteration 1, centroid shift: 2.16513
Iteration 2, centroid shift: 1.56824
Iteration 3, centroid shift: 1.73408
Iteration 4, centroid shift: 2.46772
Iteration 5, centroid shift: 0.92034
Iteration 6, centroid shift: 1.32476
Iteration 7, centroid shift: 1.78629
Iteration 8, centroid shift: 1.03627
Iteration 9, centroid shift: 0.00000
```



[10]:
```
# Minimal quick run to see the algorithm in action
from sklearn.datasets import make_blobs
X_test, _ = make_blobs(n_samples=[30,90,15], centers=[[0,0],[5,5],[10,0]], clust
dam = DensityAwareKMeans(n_clusters=3, random_state=1, verbose=True)
dam.fit(X_test)
dam.plot_clusters(X_test)
```

```
Iteration 1, centroid shift: 1.93393
Iteration 2, centroid shift: 2.55276
Iteration 3, centroid shift: 1.65639
Iteration 4, centroid shift: 0.26693
Iteration 5, centroid shift: 0.00000
```



8. Experimental Setup

Dataset Used

We used an intentionally imbalanced synthetic dataset containing clusters with:

- 100 points (dense)
- 500 points (loose)

- 50 points (very compact)

All clusters have different variances (std = 0.2 to 1.0).

Models Compared

- Standard K-Means
- Density-Aware K-Means

Evaluation Metrics

- Visual clustering
- Silhouette score
- Centroid stability
- Misclustered samples

9. Results and Analysis

9.1 Standard K-Means Result

- Large cluster dominated
- Small sparse cluster pulled incorrectly
- Misclassification around boundaries

9.2 Density-Aware K-Means Result

- Small clusters preserved
- Large dense clusters do not overpower
- Centroids placed much more accurately
- Higher silhouette score

10. Applications

Density-Aware K-Means is useful in:

- **Customer segmentation**
  Small but valuable customer groups are no longer ignored.
- **Fraud detection**
  Sparse anomalies are better detected.
- **Medical data**
  Rare disease patterns form sparse clusters.
- **Astronomy**
  Clusters of stars vary in density.
- **Geography**
  Regions differ drastically in population density.

  This makes DA-KMeans practical for real-world, imperfect datasets.

11. Conclusion

Density-Aware K-Means introduces a simple but powerful enhancement to the traditional KMeans algorithm by incorporating density information into the clustering process.
This results in:

- better detection of small clusters
- more balanced cluster influence
- improved resilience to uneven datasets
- more meaningful clustering outcomes

This feature fills a critical gap in existing implementations and demonstrates an innovative ML model improvement suitable for academic, research, and industrial uses.

12. Future Work

You can extend this model by:

- Adding a “density sensitivity” hyperparameter
- Adding visualization of density distribution
- Building a scikit-learn compatible wrapper
- Combining with DBSCAN thresholds
- Implementing GPU acceleration