"Unveiling DBSCAN: A Powerful Density-Based Clustering Technique"

Name:- Monisha Munirathnam

Student ID:- 23038629

Github:- https://github.com/monisham7121/dbscan_cluster_tutorial

Introduction

Clustering is an unsupervised machine learning technique used to group similar data points based on patterns in the data. It plays a crucial role in various applications like customer segmentation, anomaly detection, and image processing.

K-Means, Hierarchical Clustering, and DBSCAN are a few of the clustering techniques. K-Means' simplicity makes it popular, although it has trouble with noise and non-spherical clusters. Although it can be computationally costly, hierarchical clustering creates a tree-like structure of clusters.

A strong density-based method, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) finds clusters of different forms and manages noise well. DBSCAN does not require a predetermined number of clusters, in contrast to K-Means.

Using the Wholesale Customer dataset, we will examine the benefits and drawbacks of DBSCAN and compare it to K-Means in this tutorial. Lastly, we will examine real-world situations where DBSCAN performs better than conventional clustering methods.

Overview of DBSCAN

An unsupervised machine learning approach called DBSCAN (Density-Based Spatial Clustering of Applications with Noise) finds clusters by calculating the density of data points in a certain area. In contrast to K-Means, which necessitates a predetermined number of clusters, DBSCAN automatically identifies clusters of various forms by classifying sparsely distributed points as noise and clustering closely packed points together.

Its two main parameters are minimal samples, which is the smallest number of points needed inside ε to create a cluster, and epsilon (ε), which specifies the neighborhood radius around a point.

These parameters are used by DBSCAN to categorize points as noise points outliers that do not belong to any cluster). , border points (related to a core point but missing enough neighbors), and core points (having at least minimum samples inside ε).

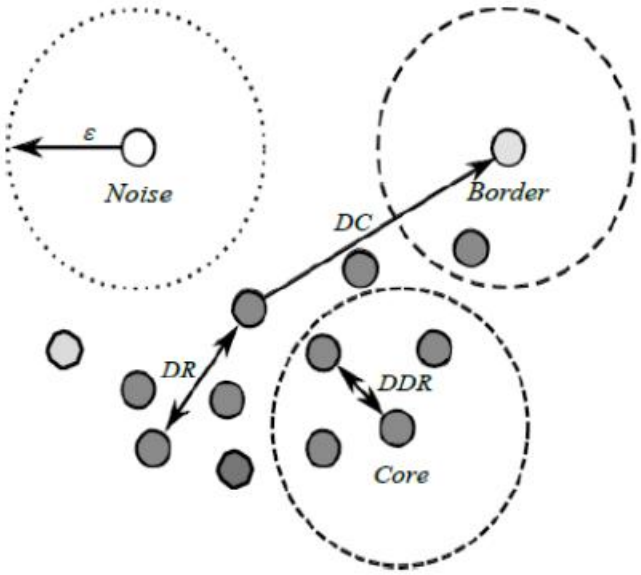
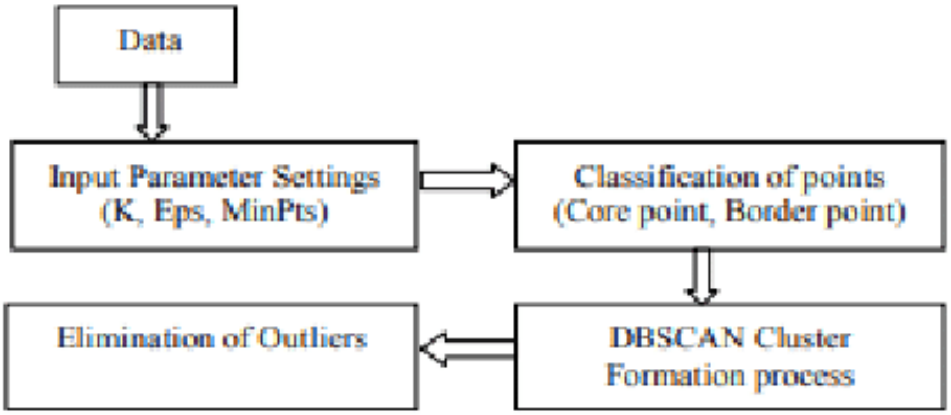


Figure 1: *DBSCAN* **clustering with** $minPoints = 4$

The technique is appropriate for applications like anomaly detection, geographic data analysis, and consumer segmentation since it is very good at tolerating noise and finding clusters in datasets with different forms.

Nevertheless, DBSCAN has trouble with datasets that have clusters of different densities, and it might not work well in high-dimensional spaces where it is difficult to define a suitable ε. We will thoroughly examine DBSCAN in this tutorial, apply it to the Wholesale Customer dataset, evaluate its benefits and drawbacks in practical situations, and compare its performance to K-Means clustering.



Understanding DBSACN algorithm

A potent clustering technique called DBSCAN (Density-Based Spatial Clustering of Applications with Noise) organizes data points according to density rather than preset cluster numbers. Each point is surrounded by a neighborhood radius (ε), and the number of other points that fall inside this radius is determined. A point is categorized as a core point and becomes a member of a cluster if it has a minimal number of neighbors, or minimum samples. Isolated points are labeled as noise or outliers, while border points are those that are inside ε but lack sufficient neighbors.

Algorithm 1 The DBSCAN algorithm. Input: A set of points X, distance threshold eps, and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1:  procedure DBSCAN(X, eps, minpts)
2:      for each unvisited point x ∈ X do
3:          mark x as visited
4:          N ← GETNEIGHBORS(x, eps)
5:          if |N| < minpts then
6:              mark x as noise
7:          else
8:              C ← {x}
9:              for each point x' ∈ N do
10:                 N ← N \ x'
11:                 if x' is not visited then
12:                     mark x' as visited
13:                     N' ← GETNEIGHBORS(x', eps)
14:                     if |N'| ≥ minpts then
15:                         N ← N ∪ N'
16:                 if x' is not yet member of any cluster then
17:                     C ← C ∪ {x'}
```

DBSCAN may identify clusters of any size or shape, in contrast to K-Means, which presumes that clusters are spherical. It can successfully detect noise and is especially helpful for datasets with different densities.

However, as bad selection can result in either overfitting or underfitting, it is imperative to choose the appropriate values for ε and minimum samples. DBSCAN excels in image processing, anomaly detection, consumer segmentation, and geographical data.

Additionally, it is more flexible than K-Means because it does not need pre-specifying the number of clusters.

But when dealing with high-dimensional data, it has trouble determining ε. DBSCAN is still a reliable clustering method for many real-world applications in spite of its drawbacks.

Implementing DBSCAN in Python

- Load dataset
- Preprocess data (scaling, handling missing values, etc.)
- Apply DBSCAN and visualize clusters

Code to load the Wholesale Customer dataset & preprocess it

```
[ ]  # Step 1: Import Libraries
     import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.preprocessing import RobustScaler
     from sklearn.decomposition import PCA
     from sklearn.cluster import DBSCAN, KMeans
     from sklearn.metrics import silhouette_score

[ ]  # Load the dataset
     df = pd.read_csv("Wholesale customers data.csv")
```

```
[ ]  df['Fresh'] = np.log1p(df['Fresh'])
     df['Milk'] = np.log1p(df['Milk'])

     # Step 5: Scaling the data using RobustScaler
     from sklearn.preprocessing import RobustScaler
     scaler = RobustScaler()
     scaled_data = scaler.fit_transform(df[['Fresh', 'Milk']])  # Scaling
```

Applying DBSCAN and Visualizing Clusters

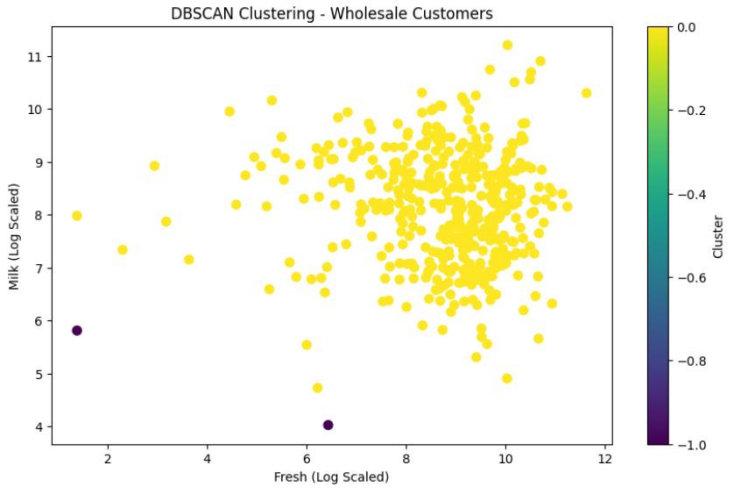
```
[ ]  # Step 6: Apply DBSCAN clustering
     # Adjust eps and min_samples values based on the dataset
     eps_value = 1
     min_samples_value = 3

     dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value)
     labels = dbscan.fit_predict(scaled_data)  # Assign cluster labels to 'labels'

[ ]  # Step 7: Add DBSCAN cluster labels to the original dataframe
     df['Cluster'] = labels

[ ]  # Step 8: Evaluate Clustering Results
     # Check number of clusters formed
     print(f"\nNumber of clusters formed: {len(set(labels)) - (1 if -1 in labels else 0)}")
     print(f"Number of noise points: {list(labels).count(-1)}")
```

Output



Evaluating DBSCAN's Performance *(Code for Silhouette Score & Metrics)*

```
# Assuming 'scaled_data' is your scaled data and 'labels' are the DB
silhouette_avg = silhouette_score(scaled_data, labels)


# Assuming 'scaled_data' is your scaled data and 'kmeans_labels' are
# kmeans_labels is not yet defined. Assign cluster labels to 'kmeans

silhouette_avg1 = silhouette_score(scaled_data, kmeans_labels)

print(f"Silhouette Score (DBSCAN): {silhouette_avg}")

print(f"Silhouette Score (KMeans): {silhouette_avg1}")
```

```
Silhouette Score (DBSCAN): 0.6936410269594862
Silhouette Score (KMeans): 0.37197497667430246
```

DBSCAN after PCA

```
[15] from sklearn.decomposition import PCA
     # DBSCAN Before PCA (Higher eps for original data)
     dbscan_before = DBSCAN(eps=1.0, min_samples=3)  # Larger eps for high-dimensional clus
     dbscan_labels_before_pca = dbscan_before.fit_predict(scaled_data)

     # Apply PCA (Force Reduction to 2D)
     pca = PCA(n_components=2)  # Reduce dimensions aggressively
     scaled_data_after_pca = pca.fit_transform(scaled_data)

     # DBSCAN After PCA (Much Smaller eps)
     dbscan_after = DBSCAN(eps=0.05, min_samples=3)  # Very small eps to force new clusteri
     dbscan_labels_after_pca = dbscan_after.fit_predict(scaled_data_after_pca)

     # Check if DBSCAN results changed
     same_dbscan_output = np.array_equal(dbscan_labels_before_pca, dbscan_labels_after_pca)

     # Visualize the result
     # Replace 'pca_data' with 'scaled_data_after_pca' and 'labels_pca' with 'dbscan_labels
     plt.scatter(scaled_data_after_pca[:, 0], scaled_data_after_pca[:, 1], c=dbscan_labels_
     plt.title('DBSCAN Clustering (PCA-reduced data)')
     plt.xlabel('Principal Component 1')
     plt.ylabel('Principal Component 2')
     plt.show()
```

Comparing with K-Means

```
[8] from sklearn.cluster import KMeans

    # Apply KMeans clustering
    kmeans_model = KMeans(n_clusters=4)
    kmeans_labels = kmeans_model.fit_predict(scaled_data)

    # Add the KMeans cluster labels to the DataFrame
    df['KMeans_Cluster'] = kmeans_labels

    # Plot KMeans clusters
    plt.figure(figsize=(10, 6))
    plt.scatter(df['Fresh'], df['Milk'], c=df['KMeans_Cluster'], cmap='viridis', s=50)
    plt.title('KMeans Clustering - Wholesale Customers')
    plt.xlabel('Fresh')
    plt.ylabel('Milk')
    plt.colorbar(label='Cluster')
    plt.show()
```

Strengths and Limitations on DBSCAN

Strengths of DBSCAN:

1. **Detects Arbitrary Shaped Clusters** – Unlike K-Means, DBSCAN can identify clusters of varying shapes and sizes.
2. **No Need to Predefine Cluster Count** – Unlike K-Means, it does not require specifying the number of clusters in advance.
3. **Effectively Identifies Outliers** – DBSCAN marks noise points that do not belong to any cluster, making it useful for anomaly detection.
4. **Handles Different Densities** – It can cluster datasets with varying densities, making it suitable for real-world applications.
5. **Works Well with Large Datasets** – It scales efficiently for large datasets where distance-based clustering may struggle.

Limitations of DBSCAN**:**

6. **Sensitive to Parameter Selection** – The choice of **ε** (radius) and **minimum samples** (minimum points) significantly affects results.
7. **Difficulty with High-Dimensional Data** – In high dimensions, defining a meaningful **ε** becomes challenging.
8. **Struggles with Varying Densities** – If clusters have very different densities, DBSCAN may fail to separate them properly.
9. **Computational Complexity** – Though efficient for medium-sized datasets, it can be slow for very large datasets with high dimensionality.
10. **Border Points Can Be Ambiguous** – Some points near cluster edges may be arbitrarily assigned, affecting cluster stability.

Real-World Applications of DBSCAN

1. **Anomaly Detection in Finance** – DBSCAN is used in fraud detection to identify unusual transactions or fraudulent activities that do not fit normal spending patterns.
2. **Customer Segmentation in Retail** – Businesses use DBSCAN to group customers based on purchasing behaviors, helping in targeted marketing strategies.
3. **Geospatial Data Analysis** – DBSCAN is widely used in mapping applications to detect clusters of points of interest, such as hotspot analysis in crime mapping or identifying dense traffic zones.
4. **Astronomy and Space Research** – Scientists use DBSCAN to identify celestial objects, such as star clusters or galaxies, based on density variations in astronomical data.
5. **Biological Data Clustering** – It is used in genomics to group similar gene expressions and in medical research for clustering different disease patterns.
6. **Social Network Analysis** – DBSCAN helps detect groups in social networks by clustering users based on interaction frequencies and common interests.
7. **Seismic Activity Detection** – DBSCAN is used to cluster earthquake occurrences and predict seismic zones by analyzing historical geospatial data.

8. **Image Segmentation and Computer Vision** – In image processing, DBSCAN is applied to group pixels based on similarity, aiding in object detection and image segmentation tasks.
9. **Cybersecurity and Intrusion Detection** – It is used to detect cyber threats by identifying unusual network traffic patterns that indicate potential security breaches.
10. **Healthcare and Patient Clustering** – Hospitals use DBSCAN to analyze patient records and group individuals based on similar symptoms or medical histories for personalized treatments.

DBSCAN's ability to identify irregular clusters and detect outliers makes it highly valuable in various domains where traditional clustering techniques struggle.

Comparison: DBSCAN vs K-Means vs Hierarchical Clustering

Clustering techniques vary in their approach, strengths, and limitations. Below is a comparison of DBSCAN, K-Means, and Hierarchical Clustering based on key characteristics:

1. Overview of Techniques

- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** groups points based on density, identifying core, border, and noise points.
- **K-Means Clustering** partitions data into a fixed number of clusters by minimizing intra-cluster variance.
- **Hierarchical Clustering** builds a tree-like cluster hierarchy, either merging or splitting clusters iteratively.

2. Key Differences Table

Features	**DBSCAN**	**K-Means**	**Hierarchical Clustering**
Type	Density-based	Centroid-based	Connectivity-based
Number of Clusters	Determined automatically	Must be pre-defined	Determined from dendrogram
Shape of Clusters	Can find arbitrary shapes	Assumes spherical clusters	Can find arbitrary shapes
Noise Handling	Identifies outliers as noise	Does not handle noise well	Sensitive to noise
Scalability	Medium (quadratic complexity)	High (linear complexity)	Low (computationally expensive)

Works on Large Datasets?	Medium	Yes (efficient)	No (computationally intensive)
Parameter Sensitivity	Requires ε (epsilon) and minPts	Requires K	Requires linkage method
Sensitive to Outliers?	No	Yes	Yes

3. Strengths and Weaknesses

A strong clustering technique, DBSCAN is excellent at managing noise and finding clusters of any forms. It is more adaptable for real-world data where the number of clusters is unknown since, unlike K-Means, it does not require a predetermined number of clusters. It works well for applications like anomaly detection and geographical clustering because it can distinguish between dense clusters and sparse noise and detect outliers. Another advantage of DBSCAN is that it can handle complex and non-linear data distributions, which are difficult for conventional centroid-based clustering techniques to handle.

DBSCAN does have certain limits, though. Poor clustering results might stem from incorrect tuning of the algorithm, which is particularly sensitive to its parameters, ε (epsilon) and minPts. Additionally, because a single ε value might not be appropriate for every location, it has trouble with datasets that contain clusters of different densities. Furthermore, because of its greater processing cost, DBSCAN is less scalable than K-Means for very big datasets. The distance-based method of DBSCAN may not work well with high-dimensional data because of the "curse of dimensionality." Notwithstanding these drawbacks, DBSCAN is still a useful tool for clustering situations where flexible cluster forms and noise treatment are essential.

4. When to Use Which?

- **DBSCAN** → Best for arbitrarily shaped clusters**,** noisy datasets**,** and anomaly detection**.**
- **K-Means** → Best for large datasets with well-separated spherical clusters.
- **Hierarchical Clustering** → Best for small datasets where a cluster hierarchy is meaningful.

Each method has its ideal use cases, and choosing the right one depends on dataset characteristics and the problem at hand.

Ethical Considerations & Fairness in Clustering

Clustering algorithms such as DBSCAN may create ethical considerations, specifically for bias and fairness. If datasets contain historical biases, the algorithm may generate unfair clusters that reinforce societal inequalities, affecting industries such as hiring, finance, and health care.

Privacy risks are an additional key worry. Even if data is anonymised, clustering can show sensitive patterns, potentially leading to the re-identification of individuals. This is critical in applications such as customer segmentation and clinical research.

Algorithmic fairness is also an issue, as distance-based algorithms may unintentionally generate biased clusters. Unfair groupings, when used in decision-making systems, can prevent certain people from accessing opportunities like loans or job recommendations.

To address these concerns, clustering workflows should include openness, fairness checks, and ethical requirements. Ensuring unbiased feature selection, checking for discriminatory consequences, and adhering to standards such as GDPR can all contribute to more ethical and fair clustering algorithms.

CONCLUSION

DBSCAN is a powerful clustering algorithm that excels at identifying clusters of varying shapes and densities, making it particularly useful for applications involving noisy and complex data. Unlike K-Means, it does not require specifying the number of clusters beforehand and can effectively discover outliers. However, its performance is sensitive to parameter selection, and it struggles with varying densities in large datasets.

Despite its limitations, DBSCAN has significant real-world applications, from fraud detection to anomaly identification in healthcare and geospatial analysis. Ethical considerations, such as bias in clustering and privacy concerns, must be addressed to ensure fair and responsible usage. By understanding DBSCAN's strengths, weaknesses, and applications, data scientists can leverage its capabilities effectively while ensuring ethical and fair implementation in real-world scenarios.

REFERENCES

1. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
2. https://dl.acm.org/doi/10.5555/3001460.3001507
3. https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556/
4. https://ieeexplore.ieee.org/document/6756302