DBSCAN Clustering

Density-Based Spatial Clustering of Applications with Noise

Unsupervised Learning – Clustering Techniques

Machine Learning Notes

Contents

1 Introduction to DBSCAN

DBSCAN stands for **Density-Based Spatial Clustering of Applications with Noise**. It is a powerful technique used for both clustering and outlier detection in unsupervised learning.

Key Concept: DBSCAN focuses on using the *density of points* as its main factor for assigning cluster labels. This creates the ability to find cluster segmentations that other algorithms have great difficulty with.

1.1 Why DBSCAN?

Consider the "moons" dataset – two crescent-shaped clusters where one moon is concave and another is convex. From a human perspective, it is very clear that there are two distinct moon-shaped clusters. However, distance-based clustering methods like K-Means have issues with such datasets.

The tips of the moons are quite close to each other in terms of Euclidean distance. This means if focusing on distance alone, the clustering algorithm may not correctly identify the two separate moon shapes. K-Means would "slice" the dataset in half rather than identify the natural moon-shaped clusters.

DBSCAN Advantage: Unlike K-Means which uses distance as the primary metric, DBSCAN focuses on density. This allows it to identify clusters of arbitrary shapes, including moons, circles, and other non-convex patterns.

2 Core Concepts and Terminology

2.1 Key Hyperparameters

DBSCAN has two main hyperparameters that must be set before training:

1. **Epsilon (ε):** The distance extended from a point to search for neighboring points. This defines the radius of the neighborhood around each point.
2. **Minimum Points (min_samples):** The minimum number of points required within the epsilon distance for a point to be considered a core point.

2.2 Point Types in DBSCAN

DBSCAN classifies points into three categories:

2.2.1 Core Points

A **Core Point** is a point that has at least `min_samples` points within its epsilon (ε) neighborhood, including itself.

For example, with $\varepsilon = 1$ and min_samples $= 3$:

- If a point has 4 points (including itself) within radius 1, it is a core point
- The point satisfies the density requirement for cluster formation

2.2.2 Border Points

A **Border Point** is within the epsilon range of a core point but does not itself contain the minimum number of points to be a core point.

Border points are part of a cluster (through their connection to a core point) but represent the "edge" or boundary of that cluster.

2.2.3 Outlier (Noise) Points

An **Outlier Point** (or Noise Point) cannot be reached by any core or border points. It is too far away from any cluster to be assigned membership.

Automatic Outlier Detection: One of DBSCAN's key features is that it automatically identifies outliers as points labeled with cluster value -1. This makes DBSCAN valuable for both clustering and anomaly detection tasks.

3 The DBSCAN Algorithm

3.1 Algorithm Process

The DBSCAN algorithm follows these steps:

1. **Pick a random unassigned point:** Select any point that has not been assigned a cluster label or point type.
2. **Determine point type:** Based on epsilon and min_samples, classify the point as core, border, or outlier.
3. **Expand clusters:** Once a core point is found, add all directly reachable points (continuous core points or border points) to the same cluster.
4. **Repeat:** Continue this process until all points have been assigned to a cluster label or marked as outliers.

Density Reachability:

- A point q is *directly density-reachable* from point p if q is within the ε-neighborhood of p and p is a core point.
- A point q is *density-reachable* from p if there is a chain of directly density-reachable points connecting them.

4 Understanding Epsilon

4.1 Effect of Epsilon

- **Increasing epsilon:** More points are considered within the neighborhood, leading to:
  - More points classified as core points
  - More border points (fewer outliers)
  - Eventually, everything merges into a single cluster
- **Decreasing epsilon:** Fewer points fall within the neighborhood, leading to:
  - Fewer core points
  - More unique clusters
  - Eventually, every point becomes its own outlier

4.2 Extreme Cases

Epsilon Extremes:

- *Very large ε:* All points fall within the same neighborhood, resulting in one giant cluster with no outliers.
- *Very small ε:* No point can reach another, resulting in every point being labeled as an outlier (cluster $= -1$).

4.3 Finding Optimal Epsilon

The elbow method can be used to find a good epsilon value:

1. Run DBSCAN with multiple epsilon values
2. For each epsilon, record the number (or percentage) of outliers
3. Plot epsilon vs. outlier count/percentage
4. Look for the "inflection point" or "elbow" where the curve changes dramatically

```python
import numpy as np
from sklearn.cluster import DBSCAN

outlier_percent = []
number_of_outlier = []

for eps in np.linspace(0.001, 10, 100):
    dbscan = DBSCAN(eps=eps)
    dbscan.fit(data)

    # Count outliers (label == -1)
    n_outliers = np.sum(dbscan.labels_ == -1)
    number_of_outlier.append(n_outliers)

    # Calculate percentage
    pct = 100 * n_outliers / len(dbscan.labels_)
    outlier_percent.append(pct)
```

Listing 1: Finding Optimal Epsilon

5 Understanding Minimum Samples

5.1 Effect of min_samples

The `min_samples` parameter determines the density threshold for core points:

- **Increasing min_samples:**
  - Requires higher density for core point classification
  - More points become outliers
  - Useful for finding only very dense clusters
- **Decreasing min_samples:**
  - Lower density threshold
  - More points qualify as core points
  - Can discover smaller, less dense clusters

5.2 Rule of Thumb for min_samples

Empirical Guideline: A commonly used starting point for `min_samples` is:

$$\text{min\_samples} = 2 \times \text{number of dimensions}$$

For a 2D dataset (two features), start with min_samples = 4.

```python
outlier_percent = []

for n in np.arange(1, 100):
    dbscan = DBSCAN(min_samples=n)
    dbscan.fit(data)

    pct = 100 * np.sum(dbscan.labels_ == -1) / len(dbscan.labels_)
    outlier_percent.append(pct)

# Plot to visualize the effect
import matplotlib.pyplot as plt
plt.plot(np.arange(1, 100), outlier_percent)
plt.xlabel('min_samples')
plt.ylabel('Percentage of Outliers')
plt.show()
```

Listing 2: Testing Different min_samples Values

6 DBSCAN vs K-Means Comparison

6.1 Key Differences

Aspect	**K-Means**	**DBSCAN**
Primary Metric	Distance to centroid	Point density
Cluster Shape	Spherical/convex	Arbitrary
Number of Clusters	Must be specified	Automatically determined
Outlier Handling	None (assigns all points)	Automatic detection
Hyperparameters	k (number of clusters)	ε, min_samples

6.2 When to Use DBSCAN

DBSCAN is particularly effective for:

- Non-spherical cluster shapes (moons, rings, arbitrary shapes)
- Datasets with noise/outliers
- When the number of clusters is unknown
- Detecting anomalies in data

7 Implementation in Python

7.1 Basic DBSCAN Usage

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
from sklearn.cluster import DBSCAN

# Load data
data = pd.read_csv('data.csv')

# Create and fit DBSCAN model
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(data)

# Visualize results
sns.scatterplot(data=data, x='X1', y='X2',
                hue=labels, palette='Set1')
plt.show()
```

Listing 3: Basic DBSCAN Implementation

7.2 Helper Function for Visualization

```python
def display_categories(model, data):
    """
    Fit model and display clustering results
    """
    labels = model.fit_predict(data)
    sns.scatterplot(data=data, x='X1', y='X2',
                    hue=labels, palette='Set1')
    plt.title(f'Clusters Found: {len(set(labels)) - (1 if -1 in labels else
        0)}')
    plt.show()

# Usage
dbscan = DBSCAN(eps=0.2)
display_categories(dbscan, moons_data)
```

Listing 4: Display Function for Clustering Results

7.3 Analyzing Outliers

```python
# Fit DBSCAN
dbscan = DBSCAN(eps=1)
dbscan.fit(data)

# Access cluster labels
labels = dbscan.labels_

# Count outliers (labeled as -1)
n_outliers = np.sum(labels == -1)
print(f"Number of outliers: {n_outliers}")

# Calculate outlier percentage
outlier_pct = 100 * n_outliers / len(labels)
print(f"Outlier percentage: {outlier_pct:.2f}%")

# Get outlier indices
outlier_mask = labels == -1
outlier_points = data[outlier_mask]
```

Listing 5: Counting and Analyzing Outliers

8 Practical Considerations

8.1 Dataset Examples

1. **Blobs Dataset:** Standard spherical clusters – both K-Means and DBSCAN perform well
2. **Moons Dataset:** Two crescent-shaped clusters – DBSCAN succeeds where K-Means fails
3. **Circles Dataset:** Concentric circles (donut shape) – DBSCAN can identify inner and outer rings; K-Means cannot
4. **Blobs with Outliers:** DBSCAN automatically identifies the outlier points between clusters

8.2 Tuning Strategy

1. Start with min_samples $= 2 \times$ dimensions
2. Use the elbow method to find optimal epsilon
3. Plot outlier percentage vs. epsilon
4. Look for the inflection point where outliers stabilize
5. Fine-tune based on domain knowledge about expected outlier proportion

8.3 Common Issues

- **All points as outliers:** Epsilon is too small
- **All points in one cluster:** Epsilon is too large
- **Too many small clusters:** min_samples might be too high relative to data density

9 Summary

Key Takeaways:

1. DBSCAN is a density-based clustering algorithm that can find clusters of arbitrary shapes
2. It automatically identifies outliers (noise points) labeled as -1
3. Two key hyperparameters: epsilon (ε) and min_samples
4. Core points have sufficient neighbors; border points are reachable from core points; outliers are unreachable
5. Use the elbow method to find optimal hyperparameters
6. Rule of thumb: min_samples $= 2 \times$ dimensions
7. DBSCAN excels with non-spherical clusters where K-Means struggles