Unsupervised Learning Metrics

Clustering Techniques

Clustering Techniques & Evaluation

Learning Objectives

By the end of this lesson, students will be able to:

- Implement clustering algorithms (K-Means, DBSCAN).
- Apply key evaluation metrics for clustering (silhouette score).
- Differentiate clustering methods based on their strengths and weaknesses.

K-Means Clustering (Continued)

Recall the K-Means clustering example from the previous lesson, where we segmented customers based on their purchase frequency and average spend.

Let's take it one step further with evaluation Metrics

Let's evaluate our clustering using the **silhouette score**, which measures cluster cohesion and separation:

Copy

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Generate synthetic data
np.random.seed(42)
customer_data = np.random.rand(100, 2)

# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(customer_data)
labels = kmeans.labels_

# Evaluate clustering
sil_score = silhouette_score(customer_data, labels)
print(f"Silhouette Score: {silhouette_score(customer_data, labels):.2f}")

# Visualization
plt.figure(figsize=(8,6))
plt.scatter(customer_data[:,0], customer_data[:,1], c=labels, cmap='viridis')
plt.title('K-Means Customer Segmentation')
plt.xlabel('Purchase Frequency')
plt.ylabel('Average Spend')
plt.show()
```

> **Interpretation:** A silhouette score close to 1 indicates well-defined clusters.

Think About It...

Try and guess what would happen if you grouped into 5 clusters instead of 3.

> `KMeans(n_clusters=3...` -> `KMeans(n_clusters=5...`

DBSCAN: Density-Based Clustering

DBSCAN clusters data based on density, effectively handling noise and non-linear cluster shapes.

Since we have been generating random data, let's see if DBSCAN performs differently than K-Means!

Hands-On DBSCAN Example:

Copy

```
from sklearn.cluster import DBSCAN

# Apply DBSCAN
clustering = DBSCAN(eps=0.1, min_samples=5)
clustering.fit(customer_data)
labels_dbscan = clustering.labels_

# Evaluate
silhouette_dbscan = silhouette_score(customer_data, labels_dbscan)
print(f"DBSCAN Silhouette Score: {silhouette_dbscan:.2f}")

# Visualize
plt.figure(figsize=(8,6))
plt.scatter(customer_data[:,0], customer_data[:,1], c=labels_dbscan, cmap='plasma
plt.title('DBSCAN Customer Segmentation')
plt.xlabel('Purchase Frequency')
plt.ylabel('Average Spend')
plt.show()
```

Key Parameters:

- `eps` : Radius around a data point to find neighbors.
- `min_samples` : Minimum points required to form a cluster.

> **Note:** While silhouette scores can provide insights into cluster cohesion, they're not always ideal for DBSCAN, especially when clusters vary significantly in density or shape.
> Other internal cluster validation measures (like Davies–Bouldin, Calinski–Harabasz, or S_Dbw) can sometimes give a more nuanced view of DBSCAN results. —

Compare & Discuss

Compare your results:

- Which clustering method performed better?
- When might you choose DBSCAN over K-Means in a real-world situation?

Key Takeaways

- K-Means is effective for spherical, distinct clusters; DBSCAN excels with irregular clusters and noise.
- The silhouette score is one of the most common ways to evaluate clustering quality.
- Choosing the right algorithm depends on data characteristics.

Next: We'll explore **Dimensionality Reduction techniques and their evaluation.**