Unsupervised Learning Metrics

Real-World Applications of Unsupervised Learning

Learning Objectives

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

- Implement a basic recommendation system using K-Means.
- Explain how unsupervised learning fuels personalization in recommendation engines.
- Discuss real-world use cases and challenges of recommendation systems.

Recommendation Systems

Recommendation systems power many digital platforms—like streaming services (Netflix, Spotify) and e-commerce sites (Amazon). They help users find relevant content by leveraging patterns in **unlabeled** user interactions.

Why Unsupervised Learning?

- Preferences often aren't labeled (e.g., "User A loves item X").
- Clustering can reveal patterns of similarity and group users with overlapping tastes.

Hands-On Activity: Building a Simple Recommendation Engine

Scenario

Imagine you're creating a product recommendation system for an online store. You have a matrix of user ratings (or interactions) for various items. Clustering will help find similar users and recommend new items.

Provided User-Item Data

The dataset below represents user-item ratings, where rows represent users and columns represent items. Each value indicates a rating (or preference score) for a given item.

Copy

```
import numpy as np

# User-item rating matrix (12 users x 10 items)
ratings = np.array([
    [2, 4, 1, 5, 0, 3, 2, 4, 3, 1],
    [5, 5, 4, 0, 2, 1, 3, 3, 3, 4],
    [2, 0, 5, 3, 4, 2, 1, 2, 2, 1],
    [4, 1, 2, 2, 3, 4, 5, 0, 2, 2],
    [1, 0, 3, 3, 5, 0, 4, 4, 4, 4],
    [3, 4, 3, 2, 3, 5, 2, 2, 1, 0],
    [0, 2, 4, 1, 2, 3, 5, 2, 4, 4],
    [2, 3, 0, 4, 1, 2, 2, 3, 5, 3],
    [4, 0, 2, 5, 0, 1, 3, 3, 3, 2],
    [3, 2, 2, 0, 4, 4, 2, 1, 2, 0],
    [5, 0, 5, 4, 3, 0, 4, 2, 1, 2],
    [1, 1, 3, 2, 2, 3, 2, 5, 4, 3]
])

print("User-Item Ratings:\n", ratings)
```

Clustering & Generating Recommendations

Copy

```
from sklearn.cluster import KMeans

# Perform clustering on user data
kmeans = KMeans(n_clusters=3, random_state=42)
user_clusters = kmeans.fit_predict(ratings)

# Example: Generate recommendations for user 0
user_id = 0
user_cluster = user_clusters[user_id]

# Filter out ratings belonging to the same cluster
similar_users = ratings[user_clusters == user_cluster]

# Calculate average ratings for each item among similar users
avg_ratings = similar_users.mean(axis=0)

# Sort items by highest average rating (descending)
recommended_items = np.argsort(-avg_ratings)

print(f"\nUser {user_id} is in cluster {user_cluster}.")
print("Cluster-mates:\n", similar_users)
print("Recommended items (highest avg rating first):", recommended_items)
```

Code Breakdown

1. **ratings**: Static dataset of user-item preference scores.
2. **KMeans**: Clusters users based on rating similarities.
3. **similar_users**: Extracts users in the same cluster as the target user.
4. **avg_ratings**: Computes mean ratings across those users.
5. **recommended_items**: Orders items by the highest average rating first.

Discussion & Reflection

- **How might this type of recommendation system boost user engagement or sales in a real-world platform?**
- **What challenges could arise when scaling a recommendation system like this? How can we counteract those challenges?**

Challenges to Consider

- **Data Sparsity**: Real-world user-item matrices can be huge and mostly empty (e.g., a user might only rate a few items).
- **Cold-Start Problem**: New users or items lack historical data, making initial recommendations difficult. Solutions include:
    - Content-based filtering for new items using their features
    - Demographic data for new users
    - Interactive questionnaires for preference gathering
    - Popular/trending items as default recommendations
- **Scalability**: Larger datasets (millions of users/items) require more advanced methods (e.g., matrix factorization).
- **Privacy & Ethics**: Storing user data responsibly is crucial. Consider the balance between personalization and data security.

Key Takeaways

1. **Clustering for Recommendations**: We used unsupervised learning (K-Means) to group users and make item suggestions.
2. **Real-World Considerations**: Issues like sparsity, scale, and user privacy heavily influence real implementations.
3. **Further Exploration**: Advanced methods like **Matrix Factorization** and **Autoencoders** can enhance recommendation systems.

< Previous