Name:Advik Hegde **D15C/15**

Experiment 8

Aim: To implement recommendation system on your dataset using the any one of the following machine learning techniques.
o Regression
o Classification
o Clustering
o Decision tree
o Anomaly detection
o Dimensionality Reduction
o Ensemble Methods

We chose **K-Means Clustering** to build a hybrid recommendation system on the MovieLens 100K dataset, predicting movies users might like based on genres and ratings. K-Means clusters movies by genres (e.g., Animation, Action), enabling content-based filtering (e.g., “Toy Story” with “Lion King”). It’s unsupervised, fitting the dataset’s structure (1682 movies, 19 genre features), and its elbow method (K=6) ensured optimal clustering. We added a collaborative layer by sorting clusters by average ratings, creating a hybrid system. K-Means’ silhouette score (0.354) confirmed decent clustering, outperforming alternatives like Regression, which require labeled data.

Theory:

Recommendation types and measures.
Recommendation systems suggest items to users using various approaches. Content-based filtering recommends items based on their features, such as movie genres (e.g., suggesting “Lion King” for “Toy Story” due to shared Animation/Children’s genres). Collaborative filtering uses user behavior, like ratings, to find patterns (e.g., users who liked “Star Wars” also liked “Empire Strikes Back”). Hybrid filtering combines both, improving relevance by balancing item similarity and user preferences. Measures include quantitative metrics like silhouette score (0.354 in our case, assessing clustering quality) and qualitative checks, such as genre relevance (e.g., ensuring “Toy Story” recs match its Animation genre) and rating quality (recs averaging 4.2–5.0).
Types:

- Content-based: Our K-Means clustering on genres.
- Collaborative: Sorting by average ratings within clusters.
- Hybrid: Combining both in our system.

The **silhouette score** is a metric used to evaluate the quality of clusters generated by K-means clustering (or other clustering algorithms). It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to 1: values close to 1 indicate that the data points are well-clustered, with points closer to their own cluster than to

others, while values near 0 suggest overlapping clusters, and negative values imply misclassified points. It helps in determining the optimal number of clusters and assessing how well the algorithm has performed.

Measures:

- Quantitative: Silhouette score (0.354) for clustering.
- Qualitative: Genre match and high ratings of recs.

Dataset Description:

We used the **MovieLens 100K dataset**, a standard benchmark for recommendation systems, containing 100,000 ratings from 943 users on 1682 movies. It includes two key files: u.data (ratings: user ID, movie ID, rating 1–5, timestamp) and u.item (movie details: ID, title, 19 binary genre features like Action, Comedy). This dataset fits the professor's "content type data" hint (genres) and "customer review-like" requirement (ratings), providing both content-based (genres) and collaborative (ratings) data for our hybrid system. We accessed it via wget for efficiency.

Steps:

1. Fetch and Load Data:

```
import pandas as pd

# Load ratings
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=['user_id', 'movie_id', 'rating', 'timestamp'])

# Load movies (genres)
# The file has 24 columns, we specify 24 names and read the first 24 columns
movies = pd.read_csv('ml-100k/u.item', sep='|', encoding='latin-1',
                     names=['movie_id', 'title', 'release_date', 'video_release_date',
                            'IMDb_URL'] + [f'genre_{i}' for i in range(19)],
                     usecols=range(24))
```

```
print("Movies loaded:", movies.shape)
print(movies.head())
```

```
Movies loaded: (1682, 24)
   movie_id             title release_date  video_release_date  \
0         1   Toy Story (1995)  01-Jan-1995                 NaN
1         2   GoldenEye (1995)  01-Jan-1995                 NaN
2         3  Four Rooms (1995)  01-Jan-1995                 NaN
3         4  Get Shorty (1995)  01-Jan-1995                 NaN
4         5      Copycat (1995)  01-Jan-1995                 NaN

                                      IMDb_URL  genre_0  genre_1  \
0  http://us.imdb.com/M/title-exact?Toy%20Story%2...        0        0
1  http://us.imdb.com/M/title-exact?GoldenEye%20(...        0        1
2  http://us.imdb.com/M/title-exact?Four%20Rooms%...        0        0
3  http://us.imdb.com/M/title-exact?Get%20Shorty%...        0        1
4  http://us.imdb.com/M/title-exact?Copycat%20(1995)        0        0

   genre_2  genre_3  genre_4  ...  genre_9  genre_10  genre_11  genre_12  \
0        0        1        1  ...        0         0         0         0
1        1        0        0  ...        0         0         0         0
2        0        0        0  ...        0         0         0         0
3        0        0        0  ...        0         0         0         0
4        0        0        0  ...        0         0         0         0

   genre_13  genre_14  genre_15  genre_16  genre_17  genre_18
0         0         0         0         0         0         0
1         0         0         0         1         0         0
2         0         0         0         1         0         0
3         0         0         0         0         0         0
4         0         0         0         1         0         0

[5 rows x 24 columns]
```

2. Preprocess Features and Ratings: Extracted 19 genre columns for clustering and computed average ratings per movie from 100k ratings, merging them into a unified dataset (1682 rows).

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Extract genre features again (just to be safe)
genre_cols = [f'genre_{i}' for i in range(19)]
X = data[genre_cols]

# Elbow method to pick K
inertias = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

# Plot elbow curve
plt.plot(K_range, inertias, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for K-Means')
plt.show()
```

3. Cluster Movies with K-Means: Applied K-Means (K=6, chosen via elbow method) on genre features, grouping movies into 6 clusters based on genre similarity (e.g., Animation, Sci-Fi).

```
# Cluster with K=6
kmeans = KMeans(n_clusters=6, random_state=42)
data['cluster'] = kmeans.fit_predict(X)
print("Cluster assignments:")
print(data[['title', 'cluster']].head())
print("\nCluster sizes:")
print(data['cluster'].value_counts())
```

```
Cluster assignments:
                title  cluster
0   Toy Story (1995)        0
1   GoldenEye (1995)        2
2  Four Rooms (1995)        5
3  Get Shorty (1995)        4
4     Copycat (1995)        1

Cluster sizes:
cluster
1    620
4    403
3    279
5    221
2    116
0     43
Name: count, dtype: int64
```

4. Build Hybrid Recommendation Function: Created a function to recommend top-rated movies within a movie's cluster (e.g., "Toy Story" → "Lion King"), combining content-based (genres) and collaborative (ratings) filtering.

```
def recommend_movies(movie_title, data, n=5):
    # Debug: Check available titles
    matches = data[data['title'].str.contains(movie_title, case=False, regex=False)]
    if matches.empty:
        raise ValueError(f"No movie found matching '{movie_title}'. Check title or data.")

    # Get first match
    movie = matches.iloc[0]
    cluster = movie['cluster']

    # Get top-rated in cluster
    recs = data[data['cluster'] == cluster].sort_values('rating', ascending=False)
    recs = recs[['title', 'rating']].head(n)
    return recs

# Test with debug
print("Data shape:", data.shape)
print("Sample titles:")
print(data['title'].head(10))
print("\nRecommendations for 'Toy Story (1995)':")
try:
    print(recommend_movies('Toy Story (1995)', data))
except ValueError as e:
    print(e)
print("\nRecommendations for 'Star Wars (1977)':")
try:
    print(recommend_movies('Star Wars (1977)', data))
except ValueError as e:
    print(e)
```

```
Data shape: (1682, 26)
Sample titles:
0                                    Toy Story (1995)
1                                    GoldenEye (1995)
2                                   Four Rooms (1995)
3                                   Get Shorty (1995)
4                                      Copycat (1995)
5    Shanghai Triad (Yao a yao yao dao waipo qiao) ...
6                                Twelve Monkeys (1995)
7                                          Babe (1995)
8                             Dead Man Walking (1995)
9                                  Richard III (1995)
Name: title, dtype: object
```

5. Evaluate the Results: Visualized clusters with PCA (showing separation with overlap), analyzed genre profiles (e.g., Cluster 0 = Animation/Children's), and checked ratings (recs 4.2–5.0, above cluster averages of 2.95–3.20). Silhouette score (0.354) confirmed clustering quality.

```
[ ]  from sklearn.decomposition import PCA
     import matplotlib.pyplot as plt

     # Extract genre features again
     genre_cols = [f'genre_{i}' for i in range(19)]
     X = data[genre_cols]

     # PCA to 2D for visualization
     pca = PCA(n_components=2)
     X_pca = pca.fit_transform(X)
     data['PCA1'] = X_pca[:, 0]
     data['PCA2'] = X_pca[:, 1]

     # Plot clusters
     plt.figure(figsize=(8, 6))
     plt.scatter(data['PCA1'], data['PCA2'], c=data['cluster'], cmap='viridis', s=10)
     plt.title('K-Means Clusters (K=6) - PCA Visualization')
     plt.xlabel('PCA1')
     plt.ylabel('PCA2')
     plt.show()

     # Cluster profiles
     print("Cluster Genre Profiles (Mean Genre Presence):")
     print(data.groupby('cluster')[genre_cols].mean())
     print("\nAverage Rating per Cluster:")
     print(data.groupby('cluster')['rating'].mean())
```

```
[ ]  from sklearn.metrics import silhouette_score

     # Calculate silhouette score
     genre_cols = [f'genre_{i}' for i in range(19)]
     X = data[genre_cols]
     score = silhouette_score(X, data['cluster'])
     print("Silhouette Score (K=6):", score)

→  Silhouette Score (K=6): 0.35434207694507774
```

Silhouette score:

. **a(i)**: Average distance between (i) and all other points in the same cluster C_i (cohesion).

- $a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d(i,j)$
- $|C_i|$: Number of points in cluster C_i.
- (d(i, j)): Distance (typically Euclidean) between points (i) and (j).
- Lower (a(i)) means (i) is close to its cluster mates.

b(i): Average distance from (i) to all points in the nearest neighboring cluster C_k (separation).

- $b(i) = \min_{k \neq i} \left(\frac{1}{|C_k|} \sum_{j \in C_k} d(i,j) \right)$
- C_k: Cluster closest to (i) (smallest average distance).
- Higher (b(i)) means (i) is far from other clusters.

Silhouette Coefficient for (i):

- $s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$
- **Numerator**: $b(i) - a(i)$ = separation - cohesion. Positive if (i) is more similar to its cluster than others.
- **Denominator**: $\max(a(i), b(i))$ normalizes the score (1 if fully separated, 0 if equal, -1 if misclassified).
- Range: $-1 \leq s(i) \leq 1$.

Overall Silhouette Score:

- $S = \frac{1}{n} \sum_{i=1}^{n} s(i)$
- (n): Total number of points

DB Index:

```
from sklearn.metrics import davies_bouldin_score

db_score = davies_bouldin_score(X, data['cluster'])
print("Davies-Bouldin Score (K=6):", db_score)
```

```
Davies-Bouldin Score (K=6): 1.6945545620832612
```

S_i: Average distance within cluster (i) (scatter).

- $S_i = \frac{1}{|C_i|} \sum_{x \in C_i} d(x, c_i)$

D_ij: Distance between centroids of (i) and (j) (separatior

- $D_ij = d(c_i, c_j)$.

R_ij: Similarity ratio.

- $R_ij = \frac{S_i + S_j}{D_ij}$

DB Index: Average max similarity over all clusters.

- $DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} R_ij$
- Lower DB = tighter, better-separated clusters.

Conclusion:

In Experiment 8, we built a hybrid recommendation system using K-Means clustering on the MovieLens 100K dataset. We fetched data with wget, preprocessed by extracting 19 genre features and calculating average ratings per movie, then clustered movies into 6 groups (K=6) based on genres. Our hybrid approach recommended top-rated movies within each cluster, blending content-based (genre similarity) and collaborative (ratings) methods. We evaluated using PCA visualization (showing distinct clusters with some overlap), genre profiles (e.g., Cluster 0 = Animation/Children's, Cluster 5 = Sci-Fi/Action), and average ratings per cluster (~2.95–3.20). Recommendations like "Toy Story" → "Lion King" (4.2) and "Star Wars" → "Empire Strikes Back" (4.2) confirmed relevance, with ratings well above cluster averages. The silhouette score of 0.354 validated decent clustering quality, making this a robust, impactful system for movie recommendations.