Experiment 8

To develop a recommendation system

Laboratory Exercise

A. Procedure: Commands related to the experiments

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Load datasets
movies = pd.read_csv("/content/movies.csv")
ratings = pd.read_csv("/content/ratings.csv")

# Check the first few rows
movies.head(5)
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
# Check the first few rows
ratings.head(5)
```

	userId	movieId	rating	timestamp
0	1	110	1.0	1425941529
1	1	147	4.5	1425942435
2	1	858	5.0	1425941523
3	1	1221	5.0	1425941546
4	1	1246	5.0	1425941556

Collaborative Filtering With Cosine Similarity

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
# Load data
ratings = pd.read_csv('ratings.csv')
movies = pd.read_csv('movies.csv')

# Create user-item matrix
user_item_matrix = ratings.pivot_table(index='userId', columns='movieId', values='rating')

# Fill NaN with 0 (means no rating)
user_item_matrix = user_item_matrix.fillna(0)

# Calculate item-item similarity
item_similarity = cosine_similarity(user_item_matrix.T)  # Transpose to get items as rows

# Convert to DataFrame for easier handling
item_similarity_df = pd.DataFrame(item_similarity, index=user_item_matrix.columns,
columns=user_item_matrix.columns)
# Function to get recommendations
def get_simple_collab_recommendations(movie_id, n=10):
    # Get similarity scores for the movie
    similar_scores = item_similarity_df[movie_id]
    # Sort by similarity
    similar_scores = similar_scores.sort_values(ascending=False)

    # Exclude the movie itself
    similar_scores = similar_scores.iloc[1:n+1]

    # Get movie titles
    recommended_movies = movies[movies['movieId'].isin(similar_scores.index)]['title']
    return recommended_movies

# Example usage
print(get_simple_collab_recommendations(1))  # Toy Story (1995)
Adventure|Animation|Children|Comedy|Fantasy
```

```
257             Star Wars: Episode IV - A New Hope (1977)
352                                  Forrest Gump (1994)
360                                Lion King, The (1994)
476                                 Jurassic Park (1993)
582                                       Aladdin (1992)
767                   Independence Day (a.k.a. ID4) (1996)
1184    Star Wars: Episode VI - Return of the Jedi (1983)
1242                             Back to the Future (1985)
3027                                   Toy Story 2 (1999)
4211                                        Shrek (2001)
Name: title, dtype: object
```

Algorithm: It uses memory-based collaborative filtering with cosine similarity metric
Process: Creates a user-item rating matrix (users × movies), Calculates how similar movies are based on user rating patterns (cosine similarity between item vectors), Recommends movies that are most similar to the target movie
Key Point: "Users who liked this movie also liked these similar movies"

Content-Based Filtering using TF-IDF and Cosine Similarity

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
# Load data
movies = pd.read_csv('movies.csv')
ratings = pd.read_csv('ratings.csv')
# Preprocess genres for content-based filtering
movies['genres'] = movies['genres'].str.replace('|', ' ')
# Create TF-IDF matrix
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['genres'])

# Compute cosine similarity
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Function to get recommendations
def get_content_based_recommendations(title, cosine_sim=cosine_sim, movies=movies):
    # Get the index of the movie
    idx = movies[movies['title'] == title].index[0]
    # Get pairwise similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Sort movies based on similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    # Get top 10 similar movies (skip the first one as it's the same movie)
    sim_scores = sim_scores[1:11]
    # Get movie indices
    movie_indices = [i[0] for i in sim_scores]
    # Return top 10 similar movies
    return movies['title'].iloc[movie_indices]

# Example usage
print(get_content_based_recommendations('Pocahontas (1995)'))
```

```
770                    Hunchback of Notre Dame, The (1996)
24345    Jungle Creature: Hugo, The (Jungledyret) (Go H...
24385    Amazon Jack 2: The Movie Star (a.k.a. Hugo the...
1997                            Little Mermaid, The (1989)
1010                                        Dumbo (1941)
1004           Winnie the Pooh and the Blustery Day (1968)
1005                         Three Caballeros, The (1945)
1449                              Cats Don't Dance (1997)
2012                               Sleeping Beauty (1959)
3668                           Fun and Fancy Free (1947)
Name: title, dtype: object
```

Algorithm: Content-based filtering using: TF-IDF (Term Frequency-Inverse Document Frequency) for text processing, Cosine similarity for measuring similarity between movies

What it does: Analyzes movie genres (converted to text: "Adventure Animation Children..."), Creates numerical vectors representing each movie's genre profile, Finds movies with similar genre patterns to your input movie

For "Pocahontas (1995)" (genres: Adventure|Animation|Children|Musical|Romance), it will likely recommend other Disney-style animated musicals

User-User Collaborative Filtering with Cosine Similarity

```python
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

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

# Create user-movie matrix
user_movie = ratings.pivot_table(index='userId', columns='movieId', values='rating').fillna(0)
user_sim = pd.DataFrame(cosine_similarity(user_movie), index=user_movie.index, columns=user_movie.index)

def collab_recommend(user_id, n=5):
    # Find similar users
    similar_users = user_sim[user_id].sort_values(ascending=False)[1:6].index

    # Get their highly rated movies
    similar_ratings = user_movie.loc[similar_users]
    avg_ratings = similar_ratings.mean(axis=0)

    # Filter out movies user already rated
    user_rated = user_movie.loc[user_id][user_movie.loc[user_id] > 0].index
    recommendations = avg_ratings[~avg_ratings.index.isin(user_rated)].sort_values(ascending=False)[:n]

    return movies[movies['movieId'].isin(recommendations.index)][['title', 'genres']]

# Test
collab_recommend(47)#Seven (a.k.a. Se7en) (1995) Mystery|Thriller
```

	title	genres
737	Dr. Strangelove or: How I Learned to Stop Worr...	Comedy|War
886	Vertigo (1958)	Drama|Mystery|Romance|Thriller
896	Maltese Falcon, The (1941)	Film-Noir|Mystery
1113	Monty Python and the Holy Grail (1975)	Adventure|Comedy|Fantasy
1220	Touch of Evil (1958)	Crime|Film-Noir|Thriller

Algorithm: User-User Collaborative Filtering
Approach: Memory-based (neighborhood) collaborative filtering
Core Technique: Cosine similarity between users' rating patterns
What It Does: Creates a user-movie rating matrix (users × movies), Finds 5 most similar users to your target user (based on rating behavior), Recommends movies that these similar users rated highly (excluding movies the target user already watched)

8. Post-Experiments Exercise

A. Extended Theory: (Soft Copy)

Explain in detail the Problem statement and methodology used to perform recommendation in the experiment

Problem Statement

The experiment aims to design and implement a movie recommendation system to address the challenge of personalized content discovery in large datasets. Traditional methods struggle to efficiently filter relevant items for users due to information overload. This project explores three machine learning approaches:

- **Content-Based Filtering:** Leverages movie genres to recommend similar items based on intrinsic features.
- **Collaborative Filtering (User-User)**: Predicts preferences by identifying users with similar rating patterns.
- **Collaborative Filtering (Item-Item)**: Recommends movies frequently rated together by users.

Key challenges include handling data sparsity, ensuring recommendation diversity, and mitigating cold-start problems for new users/items.

Methodology

1. **Content-Based Filtering**

Input: Movie genres (movies.csv).
Process: Preprocess genres into TF-IDF vectors. Compute cosine similarity between movies.
Output: Top 10 genre-similar movies.

2. **Collaborative Filtering (User-User)**

Input: User ratings (ratings.csv).
Process: Create user-movie rating matrix. Calculate user-user cosine similarity. Aggregate ratings from similar users.
Output: Top 5 unwatched movies from similar users.

3. **Collaborative Filtering (Item-Item)**

Input: User ratings (ratings.csv).
Process: Create item-item similarity matrix. Identify movies frequently co-rated.
Output: Top 10 movies similar to the target item.
Tools: Python, pandas, scikit-learn (TfidfVectorizer, cosine_similarity).

Key Takeaways:

Content-Based:

Pros: Works without user ratings (good for new items).
Cons: Limited to genre similarity (may lack serendipity).
Collaborative:
Pros: Captures nuanced preferences (e.g., Seven → Dr. Strangelove).
Cons: Fails for new users/items (cold-start problem).

The experiment demonstrates three foundational recommendation techniques, highlighting their respective strengths in handling feature-based and behavioral data. Future work could explore hybrid models for improved accuracy.