10.2 Exercise: Recommender System

Data Loading:

The small MovieLens dataset includes files for movies (with titles and genres) and ratings.

In [11]:
```
# Import necessary libraries
from zipfile import ZipFile
import os
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

In [12]:
```
# Define the path where the extracted CSV files are located
extract_to_path = r"C:\Users\salin\OneDrive\Desktop\DSC630 Predictive Analytics\Week 10 - Recommender Systems"

# Load movies and ratings data
movies_df = pd.read_csv(os.path.join(extract_to_path, 'movies.csv'))
ratings_df = pd.read_csv(os.path.join(extract_to_path, 'ratings.csv'))
```

Preprocessing:

Prepare the data by merging movie titles with their ratings and extracting relevant features. I used movie genres as the primary feature.

In [13]:
```
# Preprocess movie genres for vectorization
tfidf = TfidfVectorizer(stop_words='english')
movies_df['genres'] = movies_df['genres'].str.replace('|', ' ', regex=True)
tfidf_matrix = tfidf.fit_transform(movies_df['genres'])
```

Similarity Measure:

Compute a similarity score between movies. This is done using cosine similarity on the genres, where each genre is treated as a vector dimension.

In [14]:
```
# Compute cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [15]:
```
# Create a reverse map of indices and movie titles
indices = pd.Series(movies_df.index, index=movies_df['title']).drop_duplicates()
```

Recommendations:

For a given movie input by the user, find the top 10 movies with the highest similarity scores (excluding the movie itself).

In [16]:
```
# Function to get recommendations
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwise similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return movies_df['title'].iloc[movie_indices]
```

Output:

Present the recommended movies to the user.

In [17]:
```
# Example usage
print(get_recommendations('Jumanji (1995)'))
```

```
53                         Indian in the Cupboard, The (1995)
109                          NeverEnding Story III, The (1994)
767                            Escape to Witch Mountain (1975)
1514              Darby O'Gill and the Little People (1959)
1556                                     Return to Oz (1985)
1617                           NeverEnding Story, The (1984)
1618    NeverEnding Story II: The Next Chapter, The (1...
1799                           Santa Claus: The Movie (1985)
3574    Harry Potter and the Sorcerer's Stone (a.k.a. ...
6075    Chronicles of Narnia: The Lion, the Witch and ...
Name: title, dtype: object
```

Additional Comments and Explanation of Approach

To create a recommender system using the small MovieLens dataset, I followed a simple content-based filtering approach. This process involves:

1. **Data Loading**: Loaded the dataset using Python. The small MovieLens dataset typically includes files for movies (with titles and genres) and ratings.
2. **Preprocessing**: Prepared the data by merging movie titles with their ratings and extracting relevant features. For simplicity, I used movie genres as the primary feature.
3. **Similarity Measure**: Compute a similarity score between movies. This was done using cosine similarity on the genres, where each genre is treated as a vector dimension.
4. **Recommendations**: For a given movie input by the user, find the top 10 movies with the highest similarity scores (excluding the movie itself).
5. **Output**: Present the recommended movies to the user.

This method focuses on recommending movies similar in content (genre) to the user's input. It's a basic approach suitable for demonstrating the concept of recommender systems without diving into more complex methods like collaborative filtering or deep learning models.

References:

GroupLens Research. (2024). MovieLens Dataset. Retrieved from https://grouplens.org/datasets/movielens/

```
In [ ]:
```