```
# Dalia Tavizon-Dykstra
# DSC 630
# Week 10
```

Blackboard Instructions

- Using the small MovieLens data set, create a recommender system that allows users to input a movie they like (in the data set) and recommends ten other movies for them to watch. In your write-up, clearly explain the recommender system process and all steps performed. If you are using a method found online, be sure to reference the source. You can use R or Python to complete this assignment. Submit your code and output to the submission link. Make sure to add comments to all of your code and to document your steps, process, and analysis.

Introduction

- In this project, I developed a recommender system using the MovieLens dataset. The goal of the recommender system is to suggest ten movies to users based on a movie they like. I used content-based filtering techniques, focusing on movie genres and tags, to achieve this.

Importing the data

- Importing necessary libraries and load the CSV files into Pandas DataFrames.

```
# 1. Importing the libraries
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import CountVectorizer

# Enabling Data Table Functionality in Google Colab for better
DataFrame visualization:
from google.colab import data_table
data_table.enable_dataframe_formatter()

import warnings
# Suppress all warnings
warnings.filterwarnings("ignore")

# Setting Pandas display options to avoid scientific notation in float
numbers for better readability:
pd.options.display.float_format = '{:.2f}'.format
```

Loading the data

- Loading the CSV files into DataFrames to work with the data. This includes movies, ratings, tags, and links datasets.

```
# 2. Loading the datasets
movies =
pd.read_csv('https://raw.githubusercontent.com/dtavizondykstra/BU_Data
```

```
sets/main/DSC630_PredictiveAnalytics/Week11/links.csv')
links =
pd.read_csv('https://raw.githubusercontent.com/dtavizondykstra/BU_Data
sets/main/DSC630_PredictiveAnalytics/Week11/movies.csv')
ratings =
pd.read_csv('https://raw.githubusercontent.com/dtavizondykstra/BU_Data
sets/main/DSC630_PredictiveAnalytics/Week11/ratings.csv')
tags =
pd.read_csv('https://raw.githubusercontent.com/dtavizondykstra/BU_Data
sets/main/DSC630_PredictiveAnalytics/Week11/tags.csv')
```

Merging the datasets

- Merging the datasets to create a unified DataFrame with all necessary information. Merging the datasets helps combine all relevant information into one place, making it easier to work with and analyze.
  - I aggregated the tags for each movie to create a combined feature column for building the recommender system. Aggregating tags and combining them with genres creates a comprehensive feature set that describes each movie, which is crucial for calculating similarities.
  - I calculated the average rating for each movie and merged this information with the main DataFrame. Including average ratings allows us to consider the popularity of movies, which can be useful for recommendation purposes.

```
# 4. Merge links and movies dataframes to get titles and genres
movie_details = pd.merge(links, movies, on='movieId')

# 5. Merge tags with movie details
movie_tags = pd.merge(tags, movie_details, on='movieId')

# 6. Aggregate tags for each movie
tags_agg = movie_tags.groupby('title')['tag'].apply(lambda x: '
'.join(x)).reset_index()

# 7. Merge the aggregated tags with the movie details dataframe
movies_with_tags = pd.merge(movie_details, tags_agg, on='title',
how='left')

# 8. Fill NaN values in the 'tag' column with an empty string
movies_with_tags['tag'].fillna('', inplace=True)

# 9. Create a combined feature column with 'genres' and 'tag'
movies_with_tags['combined_features'] = movies_with_tags['genres'] + '
' + movies_with_tags['tag']

# 10. Calculate the average rating for each movie
avg_ratings = ratings.groupby('movieId')
['rating'].mean().reset_index()

# 11. Merge the average ratings with the movies_with_tags dataframe
```

```
movies_with_tags = pd.merge(movies_with_tags, avg_ratings,
on='movieId', how='left')

# 12. Display the first few rows of the combined dataframe
# movies_with_tags.head()
```

Building the Recommender System

To build the recommender system, I focused on using cosine similarity based on combined features of genres and tags.

Step 1: Vectorizing Features with CountVectorizer

- The CountVectorizer serves as a sort of translator. Each movie is described by a combination of words (features) that tell us about its genre and the tags associated with it, like "comedy," "drama," "adventure," or "space," "robot," "romantic." The CountVectorizer takes all these words and creates a numerical code that represents each movie based on how frequently each word appears.
  - Translating movie descriptions into numbers helps compare movies systematically.

How It Works

1. **Collects Words:** The CountVectorizer first looks at all the words used to describe all the movies.
2. **Counts Words:** It then counts how many times each word appears in the description of each movie.
3. **Creates a Matrix:** Finally, it creates a big table (or matrix) where each row represents a movie, each column represents a word, and each cell contains the count of how many times that word appears in that movie's description.

For example, take the data below, to which the CountVectorizer might create a matrix like this:

- Movie A: "comedy adventure"
- Movie B: "drama romantic"
- Movie C: "comedy drama"

Movie	Comedy	Adventure	Drama	Romantic
Movie A	1	1	0	0
Movie B	0	0	1	1
Movie C	1	0	1	0

- *Each row is a numerical representation of a movie based on the presence of descriptive words.*

Step 2: Calculating Cosine Similarity

- After the matrix of numbers has been created, a way to compare the rows (movies) to each other to see how similar they are is needed -- this is where cosine similarity comes in.

What is Cosine Similarity?

- To understand how Cosine Similarity is helpful in this project, it may help to imagine each movie's row of numbers as a point in a multi-dimensional space. Cosine similarity measures the angle between the points. If two movies have similar descriptions, the angle between them will be small, and the cosine of this angle will be close to 1. If their descriptions are very different, then the angle will be larger, and the cosine will be closer to 0.
  - Cosine similarity measures the similarity between two movies based on the overlap in their descriptive keywords rather than the total number of words used. Sort of like comparing the angles of two spotlights – cosine similarity cares where they point, not how bright they are.

Step 3: Creating the Recommender System

- With cosine similarity, I created a function to find and recommend movies similar to a given movie.
  - **Input:** A movie title that the user likes.
  - **Process:**
    - Find the row corresponding to this movie in the similarity matrix.
    - Sort the movies based on their similarity scores.
    - Exclude the input movie from the recommendations.
    - Return the top 10 most similar movies.

```
# 13. Create a CountVectorizer object
count_vectorizer = CountVectorizer(stop_words='english')

# 14. Fit and transform the combined features column
count_matrix =
count_vectorizer.fit_transform(movies_with_tags['combined_features'])

# 15. Compute the cosine similarity matrix
cosine_sim = cosine_similarity(count_matrix, count_matrix)

# 16. Create a DataFrame for the cosine similarity matrix
cosine_sim_df = pd.DataFrame(cosine_sim,
index=movies_with_tags['title'], columns=movies_with_tags['title'])
```

Creating the Recommendation Function

- I created a function that takes a movie title as input and returns the top 10 recommended movies.This function allows users to input a movie they like and get a list of similar movies, leveraging the cosine similarity scores.

```
# 17. Function to get movie recommendations
def recommend_movies(movie_title, num_recommendations=10):
    # Get the similarity scores for the input movie
    sim_scores =
cosine_sim_df[movie_title].sort_values(ascending=False)
```

```
    # Exclude the input movie from the the top 10 recommendations
    sim_scores =
sim_scores.drop(movie_title).head(num_recommendations)

    # Return the recommended movies
    return sim_scores.index.tolist()
```

Test the Function with Real User Input

- I tested the recommendation function with a real movie title to verify its correctness. Testing the function ensures that the recommender system works as expected and provides relevant recommendations. Testing the function ensures that the recommender system works as expected and provides relevant recommendations. The movie recommendations returned, seem congruent with the chosen movie.

```
# 18. Test the recommendation function
recommend_movies('Toy Story (1995)')
```

```
["Bug's Life, A (1998)",
 'Toy Story 2 (1999)',
 "Emperor's New Groove, The (2000)",
 'Asterix and the Vikings (Astérix et les Vikings) (2006)',
 'Antz (1998)',
 'Adventures of Rocky and Bullwinkle, The (2000)',
 'Turbo (2013)',
 'Moana (2016)',
 'Monsters, Inc. (2001)',
 'The Good Dinosaur (2015)']
```

Conclusion

- In this project, I developed a movie recommender system using the MovieLens dataset. By leveraging content-based filtering techniques, I can suggest movies that users are likely to enjoy based on their preferences. This system can be further enhanced by incorporating collaborative filtering or hybrid methods to improve the quality of recommendations.