# Amy Hatman DSC 630 - 10.2

February 15, 2025

[1]:
```
# Import needed packages
import pandas as pd
import os
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_extraction.text import TfidfVectorizer

# Remove warnings
import warnings
warnings.filterwarnings("ignore")

# Truncate code to display in PDF
from IPython.core.display import display, HTML
display(HTML("<style>.jp-OutputArea { overflow: visible !important; }</style>"))
```

```
<IPython.core.display.HTML object>
```

[2]:
```
# Dataset path
dataset_path = r"C:\Users\Amy\ml-latest-small"

# Load all 4 csv files
movies = pd.read_csv(os.path.join(dataset_path, 'movies.csv'))
ratings = pd.read_csv(os.path.join(dataset_path, 'ratings.csv'))
tags = pd.read_csv(os.path.join(dataset_path, 'tags.csv'))
links = pd.read_csv(os.path.join(dataset_path, 'links.csv'))
```

A movie recommender system helps users find similar movies based on their preferences. The process begins by loading four datasets: one containing movie details, another with user ratings, a third movie tags, and last move links. The read_csv function from pandas is used for efficient data handling, ensuring smooth processing of large datasets.

[3]:
```
# Merge datasets on 'movieId'
merged = ratings.merge(movies, on='movieId', how='left')
merged = merged.merge(tags, on=['movieId', 'userId'], how='left')
merged = merged.merge(links, on='movieId', how='left')

# Save merged dataset as CSV
merged_file_path = os.path.join(dataset_path, "combined_dataset.csv")
merged.to_csv(merged_file_path, index=False)
```

The next step is merging the movie and rating data using 'movieId' to connect each rating, tag, and link to its corresponding movie. This is crucial because it allows me to understand user preferences in the context of specific movie titles, making recommendations more meaningful.

[4]:
```
# Create a pivot table
user_movie_matrix = merged.pivot_table(index='movieId', columns='userId',␣
 ↪values='rating').fillna(0)
user_tag_matrix = merged.pivot_table(index='movieId', columns='userId',␣
 ↪values='tag', aggfunc=lambda x:' '.join(str(v) for v in x if pd.notna(v))).
 ↪fillna('')
user_links_matrix = merged.pivot_table(index='movieId', columns='userId',␣
 ↪values='imdbId').fillna(0)

# Combine all matrices into one matrix
final_matrix = user_movie_matrix.astype(str) + ' ' + user_tag_matrix + ' ' +␣
 ↪user_links_matrix.astype(str)
```

A user-movie table is created with movies as rows, users as columns, and ratings as values, filling missing ratings with zero. Tags are processed with TF-IDF for text-based similarity, and IMDb links are included separately. Combining these features improves recommendations by considering ratings, tags, and external references.

[5]:
```
# Use cosine similarity between movies
movie_similarity = cosine_similarity(user_movie_matrix)
```

Cosine similarity is used to determine movie similarities by analyzing user ratings, processed tags, and IMDb links. Instead of relying solely on genres or actors, this method identifies patterns in user behavior and textual metadata to generate more precise recommendations. By integrating these diverse features, the system ensures that suggested movies align with both user interests and content attributes.

[6]:
```
# Map movieId to indices for easy lookup
movie_indices = pd.Series(movies.index, index=movies['title']).drop_duplicates()

def recommend_movies(movie_title, num_recommendations=10):
    if movie_title not in movie_indices:
        return ["Movie not found. Please try another title."]

    # Get index of the input movie
    idx = movie_indices[movie_title]

    # Get similarity scores for the input movie
    sim_scores = list(enumerate(movie_similarity[idx]))

    # Sort movies by similarity score (excluding itself)
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:
 ↪num_recommendations+1]
```

```
    # Get recommended movie indices
    movie_ids = [movies.iloc[i[0]]['title'] for i in sim_scores]

    return movie_ids
```

A title-to-index mapping is implemented to allow quick retrieval of similarity scores. This improves efficiency and makes the system scalable for larger datasets, ensuring that recommendations are generated quickly even with thousands of movies.

The recommender function works by accepting a movie title as input, finding its index in the dataset, and retrieving the top ten most similar movies, excluding the input movie itself. This ensures that users receive fresh recommendations instead of simply being presented with the same movie again.

[7]:
```
# Testing the model
movie_name = "Homeward Bound: The Incredible Journey (1993)"
recommendations = recommend_movies(movie_name)
print(f"Movies recommended for '{movie_name}':\n", recommendations)
```

```
Movies recommended for 'Homeward Bound: The Incredible Journey (1993)':
 ['Matilda (1996)', 'Angels in the Outfield (1994)', 'Newsies (1992)', '101
Dalmatians (One Hundred and One Dalmatians) (1961)', 'Negotiator, The (1998)',
'Dumbo (1941)', 'Love Bug, The (1969)', 'Dark Portals: The Chronicles of Vidocq
(Vidocq) (2001)', 'Ghosts of the Abyss (2003)', 'Twilight Saga: Eclipse, The
(2010)']
```

To test the system, "Homeward Bound: The Incredible Journey (1993)" was used as an example. The function successfully returns a list of similar movies based on user behavior. Cosine similarity is widely used in recommendation systems and is an effective technique, as demonstrated in real-world applications such as Netflix and Amazon (Towards Data Science). This approach is scalable and ensures personalized movie suggestions based on user preferences and content similarities.

Reference: Towards Data Science. How Cosine Similarity Works in Recommendation Systems. Towards Data Science. Published 2021. Accessed Feb 15, 2025.https://towardsdatascience.com/recommender-systems-explained-with-python-and-movie-data-7ff8b9cb81fd.