Task 2: Content-Based Recommendation

Implement a new recommendation algorithm that uses the MovieLens Tag Genome to recommend movies that are similar to a user's rated movies. The Tag Genome comes with the ML-33M data set, in the genome-scores.csv file. This file contains three columns: movie ID, tag ID, and relevance (the meaning of the tag IDs is in genome-tags.csv). It is complete, in that it has relevance scores for every movie for a large number of tags, although it does not cover every movie in the MovieLens data. You can think of this file as defining a vector for every movie: an approximately 1100-dimensional vector describing the movie in terms of Tag Genome tags.

Objectives

2.1. Write an algorithm by:

- Computing a user tag vector by taking the average of the tag vectors for the movies the user has watched, weighted by their rating (that is, the user's value for a tag is the average of their movies' value for that tag. You can do this relatively efficiently with NumPy vectorized operations.).
- Select a user and get the top 10 recommendations that have not been rated by that user before. This is done by computing the Pearson correlation between the user tag vector and each of the movie's tag vector not rated by the user.
- Suppose that the content based filtering of Task 1 is an ideal state-of-the-art system where it always outputs an ideal relevant top 10 recommendation movies. How do these recommendations of this task compare to the content-based filtering done in Task 1 for the same user you selected? Implement and compute the nDCG@10 from scratch of the ranking of that user. Example:

  Ideal system ranking for user i (fixed): `[1,1,1,1,1,1,1,1,1,1]`
  An example of your algorithm output ranking for user i: `[0,1,0,1,1,0,1,0,1,1]`

  where 1 in the ideal ranking indicates that the item is relevant, and 1 in your algorithm indicates that the item is relevant because it is found in the rankings of the ideal system regardless of the position of the item in the ranking@10.

In [7]:
```
import numpy as np
import pandas as pd
```

In [8]:
```
genome_scores = pd.read_csv("ml-latest/genome-scores.csv")
genome_tags = pd.read_csv("ml-latest/genome-tags.csv")
ratings = pd.read_csv("ml-latest/ratings.csv")
```

In [9]:
```
def compute_user_tag_vector(user_id, ratings, genome_scores):
    # Filter ratings for user
    user_ratings = ratings[ratings['userId'] == user_id]

    # merge ratings with genome scores on movieId
    user_genome_scores = user_ratings.merge(genome_scores, on='movieId')

    # weighted sum of tag relevances
    weighted_scores = user_genome_scores['rating'] * user_genome_scores['relevance']
    total_weight = user_ratings['rating'].sum()

    #compute average tag relevance per tag
```

```
    user_vector = weighted_scores.groupby(user_genome_scores['tagId']).sum() / total_wei
    return user_vector
```

In [10]:
```
from scipy.stats import pearsonr

def recommend_movies(user_id, ratings, genome_scores, n=10):
    user_vector = compute_user_vector(user_id, ratings, genome_scores)

    # list of user rated movies
    rated_movies = ratings[ratings['userId'] == user_id]['movieId'].values

    # list of movies that havnt been rated
    unrated_movies = genome_scores[~genome_scores['movieId'].isin(rated_movies)].drop_du

    correlations = []
    for movie in unrated_movies['movieId'].values:
        movie_vector = genome_scores[genome_scores['movieId'] == movie]['relevance'].val
        corr, _ = pearsonr(user_vector, movie_vector)
        correlations.append((movie, corr))

    # Sort based on correlation values
    sorted_movies = sorted(correlations, key=lambda x: x[1], reverse=True)

    # Top N movie recommendations
    recommended_movies = [movie[0] for movie in sorted_movies[:n]]

    return recommended_movies
```

In [11]:
```
def compute_ndcg(ideal, predicted):
    def dcg(scores):
        return sum([(2**scores[i] - 1) / np.log2(i + 2) for i in range(len(scores))])

    idcg = dcg(ideal)
    actual_dcg = dcg(predicted)
    return actual_dcg / idcg

ideal_ranking = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
predicted_ranking = [0, 1, 0, 1, 1, 0, 1, 0, 1, 1]####copy in the predicted results from

ndcg = compute_ndcg(ideal_ranking, predicted_ranking)
print(f"nDCG@10: {ndcg}")
```

```
nDCG@10: 0.522032859637427
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: