Luke Hong, Naomi Kim, Max Freeman, Esther Anemeje (Group 5)

Professor Rachlin

DS 4300

20 April 2021

**DS4300 Final Project: Movie Recommendation Engine**

Currently, consumers are finding themselves caught in the crossfire of the Video Streaming Wars. This essentially consists of large corporations, such as Netflix, HBO, and Disney, vying for users’ attention and subscription dollars with proprietary streaming platforms offering exclusive and licensed content across large catalogs of TV shows and movies. In 2020 alone, Netflix spent over $17 billion on producing original content to compete. Given the massive amounts of spending happening to generate new user subscriptions and maintain existing ones, recommendation engines for these platforms are becoming increasingly vital to success. If a streaming service can successfully keep users locked in with content tailored to their specific viewing habits, they have a much better chance at maintaining and growing subscriptions. Since this is so important, we aim to create our own recommendation engine for movies specifically to gain insights into how these large corporations keep viewers logged in for hours on end.

**Project Description and Significance**

We will create a recommendation engine for a streaming service that recommends movies based on a movies rating dataset for its users. We will use Neo4j as the primary NoSQL database to query the relationships between the movies and users. By using the ratings of movies and user data, the engine will find unique relationships incorporating different approaches to find said relationships. The recommendation of movies based on how users have rated other movies will be an interesting dimension to look at and combining that with other user and movie metadata will lead to a more holistic recommendation engine.

There are different approaches to recommendation engineering that we want to explore through our project and evaluate their performances. We want to explore approaches to building content-based vs collaborative filtering recommendation engines and evaluate their performances. We want to use this to create our overall recommendation engine.

Graphs allow flexibility in data structure which allows us to represent movies as a network of different users, movies, and others.

**Data Cleaning/Wrangling**

Out of 45449 rows, there are many rows with nulls or missing/bad data. We first dropped insignificant columns such as homepage, poster\_paath, video, imdb\_id, original\_title, spoken languages, and adult that will not help us with our recommendation engine.The belongs\_to\_collection had many blanks so it was converted to a boolean data type that will be used to reference a separate df. Other columns like budget, revenue, and runtime had blank rows that were deleted. Some values in the revenue and budget were not in the same units and had to be scaled up. Columns such as genres, production\_comapnies, credits, keywords had to be cleaned up into a list string format for analysis in python to get values such as director and cast. We also created a weighted rating column to normalize the average ratings by the number of votes. Some movie data also did not have user ratings associated with them and had to be dropped.Ultimately, we ended with clean data consisting of 3857 rows.

**Collaborative-Based Approach**

The first collaborative-based approach differs from the others because it generates movie recommendations based on user similarity. For this approach, we used Neo4j to create a “Rated” relationship between users and movies with their rating for that movie as a property of the relationship. We then used a procedure called Pearson similarity to find the users most similar to a specific user. The Pearson correlation coefficient finds users that rated movies similarly. It accounts for the fact that some users have a tendency to give higher ratings than others and normalizes the ratings based on the standard deviation of the ratings specific to a user. Once the most similar users are found, we use the k-Nearest Neighbor algorithm to get the movie recommendations. We find the 10 most similar users and get the movies those users have watched that the given user has not. For each of the 10 users, we multiply their Pearson coefficient by their rating for each movie. This produces a score which allows us to recommend movies in descending order of how likely the given user is to enjoy that movie.

Another approach in collaborative recommendation is looking at item-based wise or movie wise like how the content-based approach looked at the cosine similarity. Unlike looking at how users rated in the previous method, here we look at how movies were rated by the users. To do this, I took the columns userId and movieId from ratings.csv file that represents how users rated the movie. To look at how similar movies are, we manipulated the dataframe to have userId as the index and each movieId as a column. We ran a cosine similarity test on the transpose of this matrix to get a matrix of each movie to movie with a value ranging from 0 to 1, 1 being the exact same. These values were added to the graph as properties of a relationship similarity between 2 movies. To retrieve a list of movies based on this approach, we returned a list of movies in descending order of the cosine similarity value since the larger number means they are more similar.

**Content-Based Approach**

For the first content-based approach, we decided to categorize the values in the metadata file based on whether they were static values (do not change, no matter which movie the function is supplied) or dynamic values (change based on the movie supplied in the function). The static values in the file we decided to use were ROI (a column derived from dividing revenue by budget; good indicator of a movie’s success), Popularity, and Weighted Rating (weighted average of vote rating and vote count). The dynamic values we decided to use were Belongs To Collection (whether or not a movie belongs to a series), Genre(s), Production Companies, Cast Member(s), Director, Runtime, and Release Year.

In order to normalize all three static values to values between 0 and 1, we applied a preprocessing MinMax scaler to the data frame columns from scikit-learn. Following this, we created a new data frame column “Total Static Value” in which all three normalized static values were added up for each movie. Following this, we created a new data frame column “Dynamic Point Counter” in which points are added to a movie if they match the supplied movie’s metadata. For example, if a movie belongs to the supplied movie’s collection, five points were added due to the importance of sequels, three points were added if the director matched, one point was added for every genre that matched, etc. Following this, a new data frame column “Total Point Value” adds up a movie’s dynamic and static points, and then the entire dataframe is sorted based on this, finally yielding the top 10 similar movies to the movie supplied to the function.

For the second content-based approach, we created a metadata text analyzer that uses a count vectorizer to create a matrix that is converted into cosine similarity scores to calculate a numeric quantity that denotes the similarity between two movies. The score is ranged from -1 to 1, two vectors with the same orientation have a cosine similarity of 1 and vice versa.The get\_recommendations function inputs a title of the movie and a dataframe and outputs a descending sorted list of movie ids and their respective similarity scores. The Text metadata column consists of Keywords (Tagline/Overview columns), Director, Cast, Genres values.

**Combined Approach**

In order to combine both content-based approaches into one, we converted the second content-based approach into a function (“get\_recommendations”) that takes in the movie and metadata dataframe and returns a similarity point value for each movie\_id between zero and one. After adding this as a column to the dataframe as “luke\_points”, we went on to use the normalize function on Total Point Value so that it could be added to luke\_points in order to create the column “Final Point Value.” The data frame then sorts based on this and yields the top ten similar movies to the movie supplied based on a combination of both content-based engines.

In order to combine Naomi’s collaborative-based approach with the combined content-based approaches, we converted it into a function (“get\_movie\_rating\_recommendations”) that takes in the movie and metadata dataframe and returns a similarity point value for each movie\_id between zero and one. After adding this as a column to the dataframe as “naomi\_points”, we went on to add these values to the total collaborative point column to return a finalized version of the “Final Point Value” column. The data frame then sorts based on this and yields the top ten similar movies to the movie supplied based on a combination of both content-based engines and Naomi’s collaborative-based engine.

**Conclusion**

In conclusion, we were able to create a non-trivial movie recommendation engine that incorporated some of our own rating system for movies. We tested our engine by recommending movies for “Toy Story” since it was a movie that is well known. It recommended sequel movies and other similar Pixar animation movies like “Bug’s Life” and “Cars” which tells us that our engine is able to recommend similar movies based on a given movie. For the user-based approach, the recommendations varied in genres and storylines which is probably due to the variety in user ratings. Our combination method yielded better results than the single approach for collaborative based.

This project allowed us to explore and understand how complicated these recommendations are that are built by large companies like Netflix, Amazon, Hulu, etc. It was interesting to look at the different approaches each of us took what aspects of the data we actually had to use for these approaches. In the future, we would like to explore building a recommendation engine that takes in a user since our content-based approaches focused on taking in a movie title.