EECS 486 Final Project Report

Anisha Aggarwal, Shruti Jain, Ananya Menon, Ajay Sumanth, and Richard Wang

EECS 486 Info Retrieval, University of Michigan

1 Project Description

At a high-level, our project is a movie recommender that takes in a movie as the query and returns the top 25 similar movies.

Consider the following scenario: you've just finished an amazing movie, something outside of your typical movie-watching wheelhouse, that has left you wanting more. Unfortunately, since you're breaking into a new genre, your current streaming services fail to provide you with good suggestions similar to this new movie. Existing recommendation systems can prove to be inadequate since they rely on user preferences and data such as genre, cast, and director. These recommendations often fail to capture the essence of the original movie's plot and tend to be only loosely related. They often fail to suggest movies in genres that don't fit the current user profile. The motivation for our movie recommender stems from the universal feeling of wanting more after you finish a great movie. Rather than looking at secondary predictors of movie similarity, our system analyzes the Wikipedia plot of the query to find closely related movies, while also considering factors like language, genre, release year, cast, and director.

1.1 Idea

We've found various datasets that provide useful information for this task, but have decided to use the vishnupriyavr/wiki-movie-plots-with-summaries dataset from HuggingFace as it provides most of the features we desire.

Our initial idea is to return movies by a calculated similarity score based on database features (namely plot and genre). With this method, the user would query a movie and our recommender would calculate the similarity scores between the queried movie and the movies in our database. Our recommender would then output the top 25 similar movies. We plan to test this manually, by querying movies that we are already familiar with and assessing the accuracy of the returned suggestions based on prior knowledge about the suggested movies and/or the summary of the movie. We plan to collect user feedback ourselves and with volunteers that use our recommender. Our thought is that this initial implementation would provide better recommendations than current recommender systems as ours would look at a single movie and compare it to other movies based on specific defining features of the plot and not biasing the results with current user preferences or rating as these can potentially prevent the user from discovering relevant movies or bog them down with trending films. Furthermore, our implementation allows users to reasonably tune the weights used for coming up with recommendations, which gives them more control over the recommendations that they get.

2 Related Works

Over the past several years, especially after the pandemic, consumers' reliance on on-demand entertainment has increased significantly. The days of channel surfing have been replaced by scrolling through social media to find the best shows to watch and then pulling that up on whatever streaming platform you use. The fact that 82% of households subscribe to 4 or more video streaming platforms shows the inherent need people have for finding the things they like quickly and efficiently (Westcott et al., 2021). Thus we have identified the need for recommendation systems like ours.

2.1 Existing Methods

This being said, there are plenty of recommendation systems that one could find already, and even some that come built-in on streaming platforms. There are different popular methods that these systems use to run - the main ones we will look at in this section are Collaborative Filtering strategies, Machine Learning Algorithms, and Metaheuristic Algorithms. We will analyze previous studies that have used these techniques.

2.1.1 Collaborative Filtering

One of the most common umbrella techniques for recommendation systems is using Collaborative filtering (CF). This type of classification leverages previous user interaction and the data collected on those interactions. Deldjoo, Schedl, Cremonesi, and Pasi mention that the interaction data includes information about implicit preferences clicks and check-ins and also about explicit preferences using user ratings (Deldjoo et al., 2020). The data can be combined with other user data like demographics to match recommendations further by user (Jayalakshmi et al., 2022). We originally considered using collaborative filtering, but ultimately decided against it because we don't have user data and weren't planning on building a large-scale recommender. Additionally, we acknowledged that user preferences change and that CF takes a long time to pick up on the change. By recommending movies based off a query movie, we shouldn't run into this problem. Lastly, by not using CF, we avoid the cold-start problem, which occurs when there isn't enough information about a new user, making it difficult to give personalized recommendations.

2.1.2 Machine Learning Algorithms

The idea of a machine learning algorithm in these recommender systems is to help choose the right algorithm for different tasks (Jayalakshmi et al., 2022). In this paper they use algorithms like K-Means clustering which inherently is a collaborative filtering method focusing on the users. This allows them to look at different aspects of a bunch of users and categorize them based on specific things like age group, gender, previously watched movies, and more. They also include Principal Component analysis K-Means which diverges from collaborative filtering by focusing on content. The computations conducted in this algorithm are derived using covariance matrices. The last algorithm this paper implements is Principal Component Analysis Self-Organizing Maps (PCA-SOM). This SOM is an unsupervised learning algorithm that detects features of movies and categorizes them accordingly. Some significant drawbacks to most of these algorithms are about how reliant they might be on previous user data, time taken for computation especially if calculated at runtime, and reformulating the clustering methods when conducting unsupervised learning. The main reason we decided against ML algorithms is because we calculate similarities at runtime based on the query movie and doing these calculations at runtime with ML takes a long time.

2.1.3 Netflix

The Netflix recommender system is one of the most efficient movie recommender systems used by the public. Historically, recommendations were based on predicting how many stars a user would rate a video, but with the shift from DVDs to streaming, Netflix now utilizes a complex array of data, including viewing habits and interactions with the service, to improve recommendations. Gomez-Uribe and Hunt explore this system and talk about how the system comprises various algorithms, each serving different functions like personalizing video rankings, identifying trending content, and facilitating continued watching. These algorithms work together to generate a highly personalized homepage for each user, featuring rows of videos categorized by theme or viewing history, and are designed to balance relevance and diversity, optimizing the user's experience by presenting a mix of personalized and popular content (Gomez-Uribe and Hunt, 2016). The combining of different methods is called a Hybrid Recommendation System as mentioned in the paper by Kumar, De, and Roy (Kumar et al., 2020). Netflix's recommendation system certainly uses this hybrid model as it combines features from Collaborative Filtering and Content-Based Filtering. While many people like the personalization Netflix provides, our goal is to take user preferences out of the picture in order to provide movie recommendations that won't be biased by what the user is currently watching in case they want to branch out. We did, however, draw inspiration from the fact that Netflix uses various algorithms and decided to use a combination of jaccard similarity, word embeddings, and different weighting schemes to come up with our final similarity scores.

2.2 Challenges

Jayalakshmi also identified several challenges that come with the creation of a movie recommender system. These include the cold-start problem, accuracy, diversity, scalability, and sparsity (Jayalakshmi et al., 2022):

1. **Accuracy**: As stated in Jayalakshmi's paper, if the underlying data is imbalanced, then ac-

curacy is not a good evaluation metric. Jayalakshmi provides the following scenario: “if the data are made of 100 movies, and only 5 movies are comedies, the rest are different genres such as thrillers. If the algorithm wrongly predicts all the movies as thrillers, it will be 95% accurate because 95% are thrillers, and only five are comedies. However, from a rational standpoint, the algorithm failed to classify comedy movies”. While we do have an underlying dataset that has a slightly imbalanced genre distribution, we shouldn’t run into this problem because we aren’t focused on classifying movies based on genre. Rather, we focus more on providing several recommendations for a movie based on similarity scores of summaries, rating, director, cast, and genre, meaning.

2. **Diversity**: New and unrated movies often are not recommended as much in recommender systems. Recommender systems can introduce diversity of results by prioritizing these movies and increase their chances of being noticed. To avoid this issue, we plan to avoid taking ratings into account so as to avoid recommending only popular or older movies.

3. **Scalability**: Recommender systems also face the challenge of scalability, such that there is a balance between the size of the search space and how long it takes for the recommender system to provide results. Scalability can be achieved by performing certain computations in advance, allowing for quick suggestions when users are finding movies to watch. This is definitely an area of concern. We attempt to mitigate this by doing as many calculations prior to runtime as possible, such as embeddings and jaccard similarity calculations. Furthermore, the data structures that we use are optimized to support constant-time access for computations.

4. **Sparsity**: Sparsity in recommender systems are caused by a large amount of movie data and features, but users engaging only a limited subset of the features available. This can lead to potential biases towards highly rated or more popular movies. There are algorithms that allow recommender systems to explore a broader range of features in order to resolve this. this shouldn’t be a concern with our recommender since we don’t take user interaction/engagement or ratings into consideration. This means all movies will have an equal chance of being displayed so long as they are similar enough to the query movie.

2.3 Interesting Methods

An unusual technique for recommendation systems uses data on the user’s emotion and physical state (Wakil et al., 2015). This would be advantageous for many people because then the recommender could provide movie recommendations accordingly for each user after studying their reactions over time while watching certain things. The article discusses many features of a user to analyze their facial expressions, heart rate, and other physiological responses. However in practice this could prove to be more work than it’s worth. If a user is self describing their emotions or there’s a misstep in their facial expressions while watching something that makes the computer misread it then there’s no guarantee that this system will provide users with what they want. Additionally these methods assume that a person is going to respond the same way on different watching sessions. As they describe, someone in a bad mood might react better to an action movie than a rom-com. But that same person might prefer a rom-com on a different day when they are in love and happy. If a user has to put in this much effort and still possibly end up with mediocre results the whole thing would be fruitless. Additionally this type of effort is something that someone looking for a movie recommendation is unlikely to do due to how long it could take to set up an account and go through all the phases before finally getting results. In the world of “on-demand” these types of methods are being phased out. We hope that by recommending movies based on a query, we can provide good suggestions that align with the user’s current mood without having to measure secondary indicators of enjoyment.

3 Data Handling

The following section discusses the data sets that we used in the recommender system as well as any annotation and pre-processing that we performed on the data before using it.

3.1 Data Collection Method

For our project, we gathered data from a HuggingFace dataset that contains 35 thousand movies and their summaries from Wikipedia. The Wikipedia summaries offer concise descriptions of the movie's storyline, making them suitable for our similarity comparison purposes. Additionally, the summaries are accompanied by metadata such as genre tags, release year, and other factors. This allows us to filter and categorize movies based on genre similarity.

3.2 Data Annotation and Preprocessing Method

As our project revolves around movie recommendation based on similarity, the data annotation process primarily involves categorizing movies based on various criteria such as genre, language, plot summary, full plot description, cast, director, and ratings. For genre classification, we rely on the genre tags provided within the datasets. We use these tags to assign each movie to one or more genres, facilitating genre-based filtering during the recommendation process. To ensure accurate similarity comparison based on plot summaries and full plot descriptions, we preprocess the text data by cleaning it of noise such as contractions, punctuation, numbers, and dates. We also tokenize the text to prepare it for further analysis. For cast and director information, we extract and organize the relevant data from the datasets. This involves creating lists of cast members and identifying the director associated with each movie.

3.3 Interesting Data Samples

As part of our data collection process, we noticed some movies that are particularly interesting. For example, there are films that have a high popularity rating but are not as critically acclaimed and vice versa. We also see a large number of films that have multiple genre labels. We were interested to see how accurately our recommender was able to distinguish between these and if it could create a connection between a certain combination of genres and what the user might like. There are also a large number of foreign films in our database since our dataset contains movies in around 20 different languages. We were curious to investigate how this element would impact the quality of recommendations given by our system.

4 Method

The following section discusses the recommender and evaluation methods for the movie recommender system, as depicted in Figure 1. To reference our code, see our GitHub Repo.

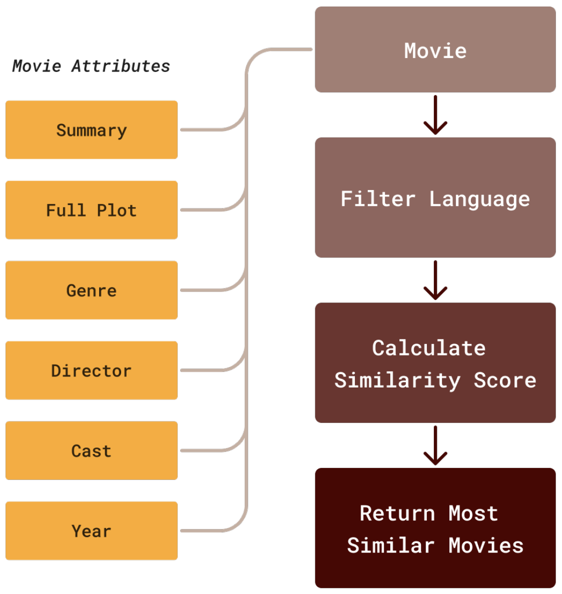

Figure 1: High-level look at our recommender system

4.1 Recommender Method

The recommender method is a multi-step process and is outlined in the following subsection.

4.1.1 Overall Algorithm

For our algorithm, we evaluate similarity and rank suggestions based on the criterias of (1) genre, (2) plot summary, (3) full plot, (4) language, (5) cast (6) director, and (7) release year. First, we preprocess all of our data from our dataset to reduce computation time. We assign each movie an ID, and store attributes such as cast, genre, director, language, and release year in dictionaries. We also tokenize the plot summaries and full plots and find a fixed size vector representation for the embeddings of the plot summary and full plot. To do so, we use the GloVe pretrained embeddings that accurately capture the meaning of the full plot and summary, and allow for comparison across movies.

Since similar movies tend to fall in the same genre, when a user enters a query, we first filter movies based on their genre, only keeping those that have at least one overlapping genre with the genre of the queried movie. We keep all movies that are of an unknown genre since we have no information about whether or not they overlap. We also filter our recommendations on language since most users tend to want movies in the same language as the movie that they are finding recommendations for. Narrowing down based on these two factors also allows us to greatly reduce our computation time, since we do not need to iterate through the pruned movies and calculate their score.

4.1.2 Weighting Scheme

We use the following novel weighting scheme for our algorithm: full plot (0.36), summary (0.48), director (0.15), cast (0.05), year (0.05), and genre (0.3). We choose these weights because the defining factor of our recommender is its ability to analyze plot and suggest based on that similarity, so we give the highest weights to full plot and plot summary. Since many movies have differing length of full plot but all movies in dataset have an approximately fixed length summary, we give more weight to the summary than the full plot due to the length being a key factor in sizing embeddings. Our next highest weight is genre because we want to give more weight to movies with the most overlapping genre(s) since our filtering will retain all movies whose genre(s) are unknown. We consider director to be an important factor as many people are fans of movies by a particular director. Cast and year are weighed slightly less as similar movies can be made across different decades and with different actors. The exact values for these weights were gradually adjusted using trial and error until we reached a point where our recommendation algorithm seemed to perform best.

4.1.3 Calculating Similarity Scores

We now explain how we calculated the scores that were then multiplied by their corresponding weights to find the final similarity score.

For the full plot as well as summary, we first create fixed length embedding representations using GloVe as explained previously. We store these in a dictionary. When a user inputs a movie name, we find the corresponding movie ID, narrow down our search space based on genre and language filters, and then compute the cosine similarity between the queried movie and other movies for both the full plot and summary. We find the weighted sum for the full plot and summary similarity, and keep the top 25 highest movies. We know that this methodology may rule out movies that are relevant for tertiary factors beyond plot, however, we chose to make this trade-off to reduce the computation time. All further calculations are only done for the top 25 movies.

To calculate the scores for cast, we find the intersection between the query and the candidate movie and divide it by the size of the query's cast. We use this same method for director scores. We use this method over Jaccard similarity as it prevents us from penalizing movies with a similar but larger cast.

To calculate the release year score, we use the formula $\frac{-\log(\frac{1}{x+1})}{\log(\frac{1}{117})}$, where x is the absolute difference between the release years of the two movies. We use the inverse as we want to penalize movies with a higher difference in release year. We use add-one smoothing to prevent any zeros in the denominator. We divide by $\log(\frac{1}{117})$ and multiply by -1 since $x = 116$ is the maximum difference in years in our dataset (earliest movie year: 1901, latest movie year: 2017), and we want values to be between -1 and 0. If two movies are of the same release years, they get a 0 score (no penalty). The farther apart the release years are, the heavier they are penalized. We believe that this curve more accurately represents the relevancy and release year difference relationship than a linear one.

As for the genre, although we already filtered on it, we want to weigh movies with the most overlapping genres most highly. We compute the Jaccard similarity between the query and other movie's genres and use this as our genre score.

4.1.4 User Adjusted Weights

We also allow the user to manually adjust the weights to change the rankings. Although due to the nature of our algorithm the 25 suggested movies remain the same, this adjustment feature allows users to find more relevant movies a lot more quickly (ie. increase precision at 5). Since we want a level of abstraction, we do not show our actual weights to the user, we simply let them choose a level between 1-5 where 1 has the least weight and 5 has the most (with the default being 3). For each change in level, we change the weight by 25 percent of the default value (ex. for director, an increase by 1 level means increasing the weight by 0.15*0.25). This allows the increments to be proportional based on the preset weights. After the user adjusts weights, they may not add up to 1, however, this does not make a difference as all movies are affected equally.

4.2 Evaluation Method

We tested the performance of our recommender based on user-feedback to calculate the precision of the recommendations they were given. We had a user type in a movie from our database that they knew as the query and then had them go through the first ten recommendations, which are provided with their respective summaries, and decide how many were similar to their query movie. As a group, we found 10 participants to each test 5 movies. We then logged how many out of the first five recommendations they felt were similar or "accurate" and then how many out of the first ten recommendations were "accurate." We then made an average Top 5 and Top 10 value to come up with our final accuracy scores.

It is important to note that our evaluation method could be improved since users are biased, which could make it difficult to get an objective measure of our accuracy.

5 Results & Discussion

From our user evaluations, we found that we had an average Precision@5 score of 0.8 and an average Precision@10 score of 0.68 across the 50 films tested. This means that, on average, 80% of the top 5 movie suggestions are relevant and about 68% of the top 10 movie suggestions are relevant. Overall, this is a very positive result as it shows that our recommender system is able to return a majority of relevant films up front. Thus, the user should be able to quickly find at least 1 good film suggestion based on their query. It is important to note the decrease in the average precision score from 5 to 10. This decrease is logical and follows the trends of the standard precision and recall curve.

As can be seen from Table 1, our evaluation results can be further analyzed by breaking the movies down by genre. It seems that Horror and Romantic Comedy movies had the best Precision values while Adventure, Fantasy, Musical, and War had the worst. At first glance, this means that our recommender suggests more relevant or similar movies depending on the genre of the query film, which is interesting since we filer by genre before calculating similarity scores in the backend. This could be due to the fact that certain genres are more niche and consist of movies with very similar plots while other genres are more broad making it hard to find patterns in the types of movies that fall into the category based on the movie characteristics we looked at. Additionally, it could be the case that the labels in our dataset are incorrect (after all, the films are labeled by humans, leading to them being subjective).

It is important to note the imbalance in the genres of movies being tested. Our participants tested significantly more Drama movies than any another genre and tested Adventure, Horror, Musical, Mystery, and Romantic Comedy the least. This definitely impacts the Precision@5 and Precision@10 values for each genre as adding more Adventure movies could have possibly bumped up the scores. Similarly, it's possible that testing more Horror films could result in lower scores. Additionally, our evaluation method is not ideal since each user is biased and has different opinions on how similar a movie is to the query movie based on how well they know the suggested films, how well they remember the query film, and what they liked best about the query film.

6 Experiments

In this section we will go through some things that had to be modified over the course of the project's timeline through trial and error.

6.1 Preprocessing

When we began working with preprocessing we set up the initial data files as dictionaries in txt forms so that it was human readable and made the most sense in linking movie IDs to their value of whatever the dictionary was supposed to hold such as their summaries or cast. The issue we ran into was that keeping these as human readable in txt files was causing a very long runtime. To fix this, we translated all of these txt dictionaries into pkl files so that the data is already in computer-readable language and the program gets the information it needs faster.

6.2 Vector Representation

We made a significant adjustment to the vector representation method for the Full Plot feature from the Hugging Face dataset on movies from Wikipedia. Originally, our plan was to utilize a TF-IDF-based approach for the Full Plots feature and only using embeddings for the Summaries. This decision was made because we saw that Full Plots had a lot of variability in length, making us want to mitigate potential errors because of padding issues – given that embeddings typically require a fixed length input. However, after recognizing that embeddings were better at capturing semantic information from the data compared to TF-IDF, we opted

Genre	Number of Movies	Average Precision@5	Average Precision@10
Action	6	0.8	0.72
Adventure	1	0.6	0.5
Animated	4	0.9	0.73
Comedy	6	0.83	0.73
Drama	15	0.8	0.67
Fantasy	3	0.6	0.53
Horror	1	1	0.9
Musical	1	0.6	0.4
Mystery	1	0.8	0.8
Romantic Comedy	1	1	1
Science Fiction	7	0.8	0.64
Superhero	2	0.9	0.75
War	2	0.6	0.55

Table 1: Average Precision@5 and Precision@10 by genre

to change our strategy and utilize embeddings for both Full Plots and Summaries. This improved the overall semantic understanding of our dataset by our recommender system. Furthermore since embeddings are fixed-length, using embeddings also helped us save space and time during computation.

6.3 Weighting Schemes & Filtering

Weighting schemes changed through the course of the project. We set our own values for the baseline model in terms of what weight each factor is given with no user interaction. With a lot of back and forth, we put the summary weight as the highest (0.48) and the release year and cast as the lowest (0.05) in the final weights. We started with genre as our highest, but after discussing the distinction between our engine and others – emphasizing the substance of the story using plots, we decided to modify this. Another point that we ended up changing was how the final score of the release year would be calculated. We understood that we would need the score to be inversely correlated with how far away the release year of a possible recommended movie was to the release year of the query movie. Meaning that the further apart two movies are, the lower the release year score should be. what we didn't know was how to show this correlation. A simple linear option would give us this result but after considering that movies that are 50 years and 80 years away from the query movie should have similar final scores. Due to this, we ultimately decided to use a logarithmic function to have a factor between 0 and 1 to multiply by the preset weight to return the final similarity score. To continue, in our filtering methods, we knew that we wanted to filter our system on genre since typically users want recommendations in the same genre. When we kept this as the only filter, we ran into the issue where the program suggested movies in the same genre with the same name, but in different languages. For instance when we queried "Frozen (2013)," we were recommended two other movies also titled "Frozen" in different languages. This led us to recognize that while genre is an important filter for a recommendation system, so is language. Thus we added language as a built-in filter rather than as a weight as we had originally planned.

7 Conclusion

Our recommendation system differs from other mainstream recommenders in that it focuses on suggesting movies that are similar based on characteristics like plot, summary, genre, release year, directors, cast, and language without taking user preference into consideration. We developed a novel weighting scheme for calculating similarity scores between movies that has led to fairly good suggestions for most genres. Additionally, our system has a very intuitive user interface allowing users to modify the relative weights of various movie characteristics such as full plot, summary, director, cast, release year, and genre. This gives users a lot more flexibility in the types of recommendations the system returns and enables them to customize the movie suggestions based on what best fits their needs.

Based on our evaluations, we noticed that our system failed to provide good recommendations for drama films as compared to other genres like comedy, horror, and superhero. That being said, we also feel that our evaluation didn't work as well as we had hoped since users are biased, making it hard to get an objective estimate on the quality of the movies recommended by our system. Additionally, while removing user preference can be beneficial when recommending movies as it allows users to avoid getting pigeonholed into a specific category, this also means that we can't give extremely personalized recommendations.

7.1 Future Considerations

In the future it would be nice to enhance this system so that it can automatically test different weights and adjust the weighting scheme based on the tests and simple user feedback (ex. asking how good the suggestions were). This would reduce the manual work in making sure the algorithm is optimal. Additionally, this recommender could be extended to other forms of media (such as books, TV shows, music, and podcasts). Books and TV shows should be fairly similar to movies, and therefore, wouldn't require too many changes besides expanding our database. Music and podcasts may be a bit more difficult, since most songs don't have a short description associated with them, and may require further thinking on the features most important for recommending these types of media. Another feature that we thought could incorporate in the future is to have the engine generate a sentence explaining why a recommendation was given (ie. This movie is similar because it mentions x in the summary). We also considered that it would be useful to fine-tune default weights over time. This would be done using simple surveys asking random users to rate whether they feel that the recommendations are accurate or not. Doing A/B testing with different versions with slightly different weights could help determine which performs better and how to fine tune the weights over time. Another feature we wanted to add was to allow the user to add a keyword to their search if they choose. We thought that adding this feature to our current model would hurt the user experience currently. We recognized that keeping the base model simple and intuitive for the user would be better so that adding functionality later would be easier.

8 Individual contributions

Below are the overall contributions of each team member made throughout the project's timeline.

1. **Anisha Aggarwal**: Anisha Aggarwal worked on the initial project proposal poster with the rest of the group. Anisha also initially worked on the preprocessing.py file. She completed the project description section for the second checkpoint. Anisha also edited and added to the Final Report in all sections. She also worked on the final poster.

2. **Shruti Jain**: Shruti Jain worked on the initial project proposal poster with the rest of the group. Shruti wrote the skeleton code for the similarities.py and calculate.py file. She wrote the methodology section in the second checkpoint. She wrote and implemented functions for similarities.py and calculate.py. She also worked on the final poster.

3. **Ananya Menon**: Ananya Menon worked on the initial project proposal poster with the rest of the group. Ananya wrote the related works section of the second checkpoint. She wrote implementations in similarities.py. Ananya also modified and added to the Final Report in all sections. She also worked on the final poster.

4. **Ajay Sumanth**: Ajay Sumanth worked on the initial project proposal poster with the rest of the group. He wrote the Data Handling section of the second checkpoint. He also wrote implementations for preprocessing.py, calculate.py, and dataset.py. Ajay also worked on the final poster.

5. **Richard Wang**: Richard Wang worked on the initial project proposal poster with the rest of the group. Richard wrote the related works section of the second checkpoint. He wrote implementations for preprocessing.py and calculate.py. He also worked on the preprocessing and front-end portion of the project. Richard also worked on the final poster.

References

Yashar Deldjoo, Markus Schedl, Paolo Cremonesi, and Gabriella Pasi. 2020. Recommender systems leveraging multimedia content. *ACM Comput. Surv.*, 53(5).

Carlos A. Gomez-Uribe and Neil Hunt. 2016. The netflix recommender system: Algorithms, business value, and innovation. *ACM Trans. Manage. Inf. Syst.*, 6(4).

Sambandam Jayalakshmi, Narayanan Ganesh, Robert Čep, and Janakiraman Senthil Murugan. 2022. Movie recommender systems: Concepts, methods, challenges, and future directions. *Sensors*, 22(13):4904.

Sudhanshu Kumar, Kanjar De, and Partha Pratim Roy. 2020. Movie recommendation system using sentiment analysis from microblogging data. *IEEE Transactions on Computational Social Systems*, 7(4):915–923.

Karzan Wakil, Rebwar Bakhtyar, Karwan Ali, and Kozhin Alaadin. 2015. Improving web movie recommender system based on emotions. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 6(2).

Kevin Westcott et al. 2021. Digital media trends, 15th edition: Courting the consumer in a world of choice. Deloitte.