Netflix Movie Rating Prediction

Akshaya Anand
University of Maryland, College Park
akshaya@korionhealth.com

***Abstract*—Recommendation systems help businesses decide which content to showcase to their user base as well as help users decide what to consume. This paper looks at how to manage large datasets, process the data, and parallelize the training of a prediction model. We then show how we can use the final model to predict the rating of movies using the cosine similarity model. We then analyze the results and the accuracy of the model.**

***Index Terms*—recommendation system, movie rating, Netflix dataset**

I. Introduction

Recommendation systems have emerged as prominent tools that leverage patterns in large datasets to enhance user experience, increase revenue streams, and optimize operations. In health care, for example, several recommendation systems have been developed to enhance diet/exercise compliance or provide personalized educational content to patients [1]. Beyond healthcare, recommendation systems have been used to propose related articles to aid researchers, or new movies for an individual to watch.

To aid in the development of new recommendation algorithms, Netflix released a large challenge dataset consisting of ratings from over 480,000 users across 17,000 movies between 1890 and 2005. This dataset, along with a prize of $1 million dollars, was released to the public as part of an open challenge to promote the development of novel recommendation algorithms. The Netflix dataset is a challenging and interesting dataset due to its size (2GB) and sparseness.

Historically, several algorithms have been employed for designing recommendation systems. Collaborative filtering relies on the hypothesis that users who behave similarly in the past are likely to behave similarly in the future. In a collaborative filtering approach, techniques such as cosine similarity are used to group similar users together and use the information from the similar group to provide recommendations [5].

Within collaborative filtering, there are user-based and item-based filtering approaches. User-based filtering works by identifying users who give similar ratings on shared movies. Item-based filtering works by identifying items that are similar to each other [2]. While both methods of collaborative filtering are used, user-based tends to work better when the number of users is greater than the number of items. By a similar rationale, item-based filtering tends to work better when there are more items than users. However, a notable drawback of user-based filtering in the context of the Netflix dataset is that there are very few movies shared between any group of users. Therefore, successful solutions tend to use a combination of user-based and item-based filtering [5].

In contrast, content-based filtering utilizes inherent features of the product and individual user preference to predict user interaction with the item. For example, various features about a movie, such as the actors, genre, release date, language, title, and other metadata will be mapped into a latent feature space. User preferences will also be mapped to a feature space, and a model will learn how to associate user preferences with the feature embedding of movies [5]. One drawback to content-based filtering is that it relies heavily on user preferences, so it is not very accurate when a new user without a preference history joins.

Finally, knowledge-based recommendation systems use known rules and heuristics to identify items to recommend to a user. For example, a heuristic may be to recommend the top-rated movies across the platform to all users.

For the context of this project, a hybrid approach was used. For a given user-movie rating query, cosine similarity was used to identify similar movies. The weighted average of the users' ratings on the top 10 similar movies was used to predict the user-movie average. This approach utilized both item-filtering and user behavior to output a final prediction. The proposed model had an average RMSE of 1.1. Preliminary sentiment analysis of the movie titles was also conducted as a proof-of-concept for future engineering, and showed that most movie titles fell under neutral, fear, or admiration.

II. Methodology

A. About the dataset

The dataset is from the Netflix Prize open competition, found on Kaggle[2]. The competition was held to see who could create the most accurate model for predicting user ratings. The dataset was divided into 4 files, each around 500MB, due to file size limits on the Kaggle website. For the project, a single file, which contained 24 Million rows, was selected for analysis. The selected file was organized as a movieID line followed by a series of comma-separated values representing the customerID, rating, and data. A similar file was provided for evaluation purposes. Figure 1 shows the distribution of 999,775 user ratings over 225 movies. As shown, there is a clear bias towards higher-rated movies (4 or 5).

In addition to customer rating information, there was a follow-up document that mapped the MovieID's to Title names. For the analysis reported in this paper, the date and title information were not used. In order to streamline architecture development and account for hardware and time limitations, only the first 1 million lines of data were extracted

to show proof-of-concept. This data subset consisted of 225 movies, 283,670 users, and 999,775 movie ratings. A custom train/val/test split was used to evaluate the performance of the proposed algorithm since the official test dataset contained many users/movies that were not in the first 1 million rows we extracted.

TABLE I
NETFLIX CHALLENGE DATASET

Variable Name	Description
MovieID	The ID of the Movie
CustomerID	ID of the customer who gave the rating
Rating	Rating given by the customer for the movie
Date	Date the movie released
Title dataset	File that contains mappings between MovieID and movie title
Test dataset	File containing movieID,userID,andDate

Table 1 describes the data available as part of the Netflix challenge dataset. MovieID, CustomerID, and Rating were key features used in the development of a (user, movie) rating prediction algorithm.

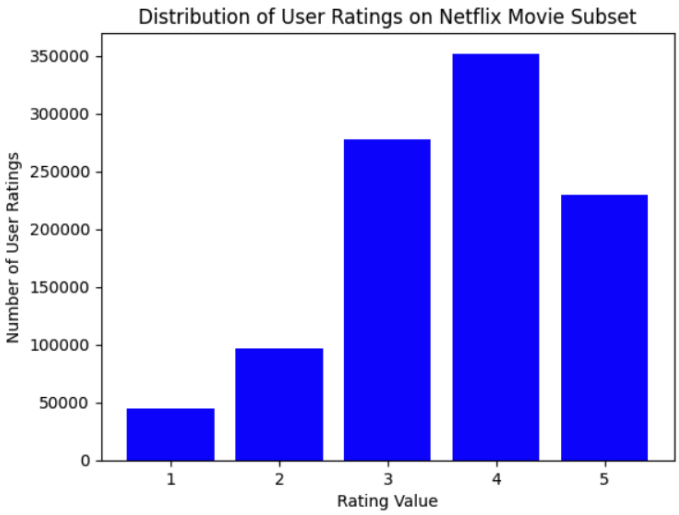

Fig. 1. Distribution of rating scores across first 255 Netflix Movies

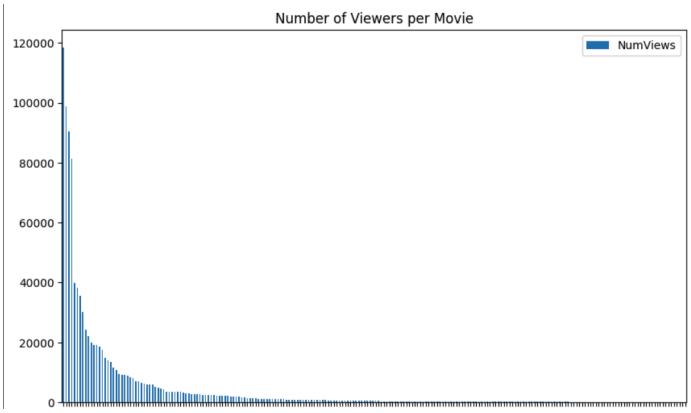

Fig. 2. Popularity distribution of movies in the dataset

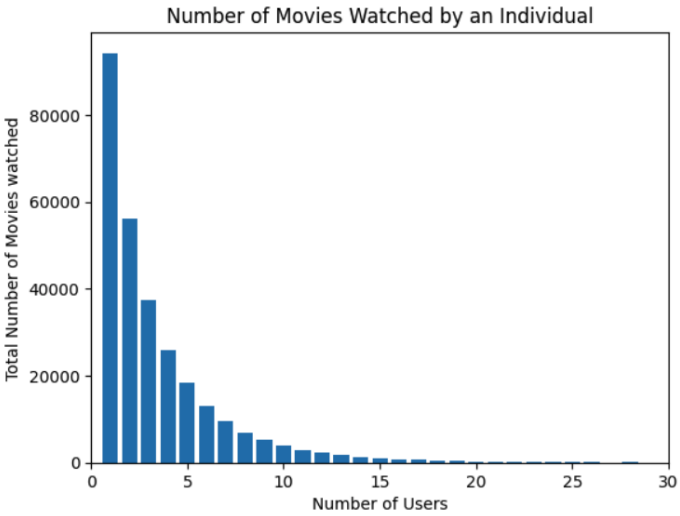

Fig. 3. Number of Movies a User Watches

As shown in Figure 2 and Figure 3, the distribution of movie popularity and the distribution of number of movies a user watches is highly skewed. There are few movies that are watched by a very large number of people, but for any given individual, they probably watch less than 10 movies. There is a higher chance of finding more shared users for a given movies than it is to find shared movies for given users. This insight suggests that grouping the dataset by movieID rather than UserID will allow for better model training since this data representation won't be as sparse.

B. Architecture

Figure 2 describes the architecture flow of processing the Netflix challenge dataset and building a (user, movie) rating prediction algorithm. First, 1M rows are extracted from the large dataset using Linux command head -1000000 [input file]. Data is stored as a CSV, in the format of:

MovieID:
CustomerId, rating, date
CustomerID, rating, date
...
...
...
MovieID:
CustomerId, rating, date
CustomerID, rating, date
...
...
...
(255 MovieIDs...)

This data was not an easy form to work with, since it did not fit the expected format of a typical CSV file, where every row is structured in the same column format. In order to reformat the data, a Python script was used to read the file line-by-line and group all ratings for a given movie. The

result was 225 individual CSV files labeled as the movieID, and containing all the customer's ratings corresponding to the movie.

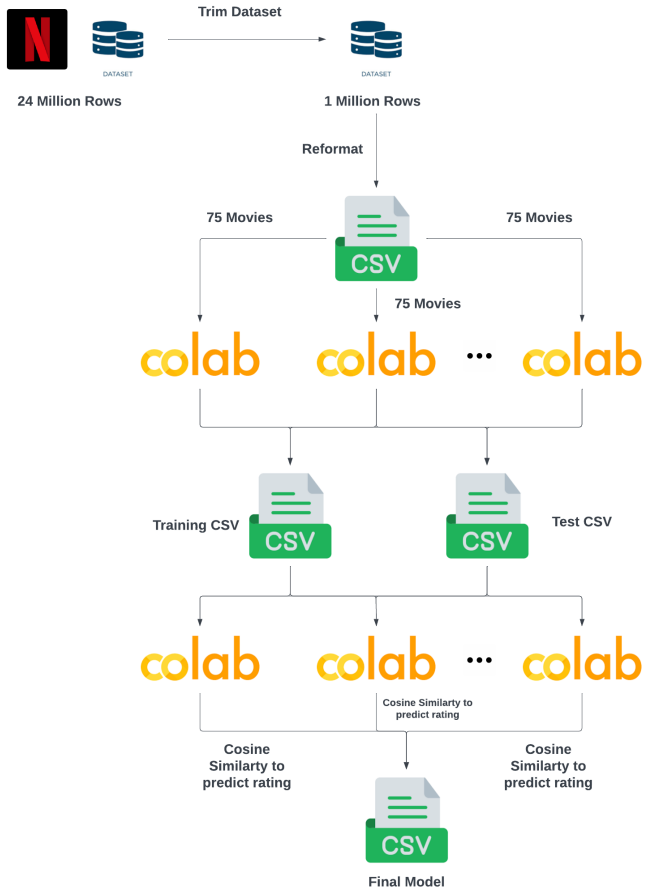

Fig. 4. Architecture diagram of rating prediction algorithm

The next step was to vectorize the data so that each movie had columns as customerID and values as ratings. Initially, dask was used to load the entire dataset and create a pivot table. However, I found it was more time-efficient to distribute the data among several collab notebooks and run the vectorizations in parallel. to vectorize the data, I took each movie, pivoted the userID as the column, and normalized the ratings. Ratings were normalized by subtracting the average rating for of the user from all the user's ratings. The vectors were saved to intermediate files, and train (80%, 180 movie vectors), val(10%, 23 movie vectors), and test(10%, 22 movie vectors) sets were created using sklearn package.

To train the classification model, I used adjusted cosine similarity [3] to identify top similar movies and then used a weighted average to predict the rating for the given (user, movie). The validation set was used to determine an optimum number of similar users (k) to include in the weighted average. The cosine similarity values were used as weights. My initial attempt was to use Pyspark and a UDF function to calculate cosine similarity and weighted average, however, I struggled to implement that solution (colab disk memory exceeded). Ultimately, I utilized a similar approach as before and distributed the calculations across multiple colab instances. Each colab notebook had read access to all the vectors in the training dataset but only had to process write calculation results from a small fraction of the validation dataset.

For each movie vector in the validation set, I calculated the cosine similarity against all other movies in the training dataset. The cosine similarity was calculated using the formula shown in Equation 1.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|} \tag{1}$$

In the equation, A represents the dot product between the two movie vectors. The denominator is the product of the magnitudes of each. One thing to note is that all the "empty" columns were dropped in order to save storage space and reduce computational complexity. I did consider using imputation to "fill in" empty values according to this article [4], however, the process was too computationally costly and did not make much sense to me given how many "empty" values there were in the dataset. Additionally, the intersection between the two vectors was calculated prior to the dot product. These optimizations helped reduce the sparseness in the dataset and sped up my calculations, although it meant I could not use mass operations on dataframes and had to distribute small, individual computations across colab notebooks. Next, I sorted the movies in descending order based on cosine similarity. Then I iterated through several values of k (5,10,15,20) to determine the optimal number of movies to use in the weighted average calculation (Equation 2).

$$\text{Pred} = \frac{\sum_{i=1}^{k} \text{CosineSimilarity}_i \times \text{NormalizedRating}_i}{\sum_{i=1}^{k} \text{CosineSimilarity}_i} \tag{2}$$

The end-to-end architecture diagram is shown in Figure 2. For a dataset of 1M rows (225 movies), the training of the rating classifier took about 2 hours when distributed between 3 colab notebook sessions. A preliminary exploration into the integration of movie titles is described. First, 1M rows from the movie title dataset were loaded. Then, a pre-trained sentiment analysis model, which was based on RoBERTa was loaded [7].

III. Results

Root Mean Squared Error (RMSE) was used as a metric to evaluate the performance of the model and select an optimum k-value. The Root Mean Squared Error (RMSE) is a metric commonly used to assess the accuracy of predictions, particularly in the context of (movie, user) rating prediction [8]. Prior to the Netflix public challenge, the best in-house RMSE value was 0.9514. RMSE measures the square root of the average squared differences between the predicted ratings ($\hat{y}_i$) and the actual ratings (y_i) for a total of n data points. The RMSE is calculated using the following formula:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Here, n represents the total number of ratings. The formula computes the squared differences between actual and predicted ratings, averages them, and then takes the square root to obtain a measure of the average error. Table 5 describes several statistics about the RMSE. The average RMSE was calculated by taking the mean of all the validation set RMSE (23 movies in the validation set). The minimum and maximum RMSE i nthe validation set is shown to describe the range in performance across different movies.

k	avg RMSE	min RMSE	max RMSE
5	1.49	1.2	2.05
10	1.43	1.19	1.79
15	1.4	1.15	1.68
20	1.39	1.09	1.67
0	1.13	0.92	1.45

Fig. 5. Validation set RMSE for different k values

The baseline value, denoted as k=0 represents the simple baseline algorithm of just predicting the avg movie rating for aall (movie,user) pairs.

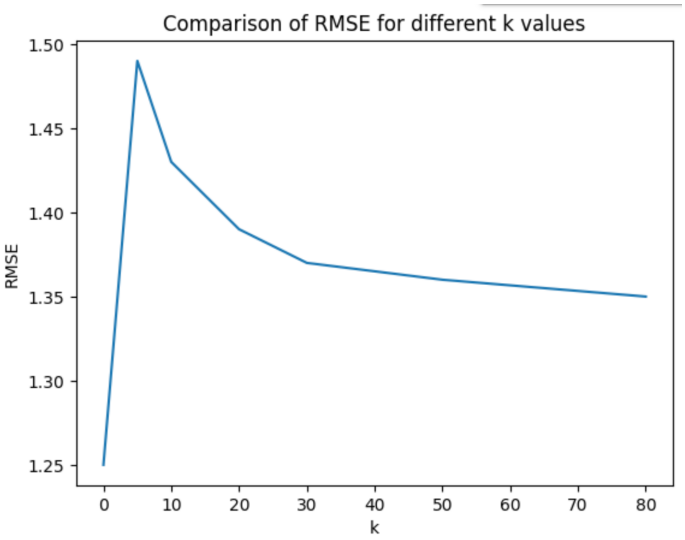

Fig. 6. Relationship between RMSE and k

As shown in Figure 6, he RMSE decreases sharply between 10-30 users in the weighted average set and then flattens out as more users beyond 30 are added. The best performing model was the baseline model. For the test set, (movie,user) pairs were evaluated using a k-value of 20 since this seemed to be an optimum value to trade off computation and RMSE. The results of the final model and the baseline model are shown in Figure 7.

k	avg RMSE	min RMSE	max RMSE
20	1.35	1.07	1.85
0	1.1	0.86	1.42

Fig. 7. Test set RMSE

As an interesting "side exploration", an additional feature was engineered too include information about the movie title that could potentially enhance the rating prediction algorithm. For each movie title, all significant emotions associated with the sentiment analysis was tabulated. About 86% of the titles were neutral. Of the 14% non-neutral emotions identified, the two dominant emotions were fear or admiration. Figure 5 how many movies were associated with each of the diverse emotions.

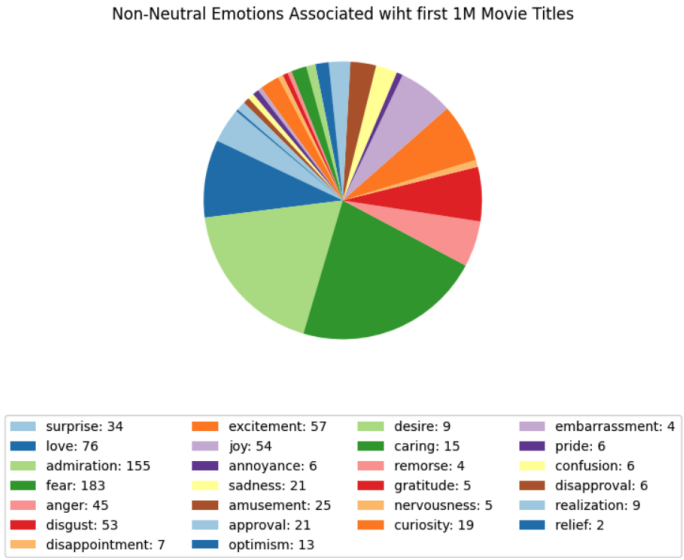

Fig. 8. Emotions Associated with Movie Titles

IV. Discussion

Interestingly, none of the models performs better than the baseline algorithm. This implies that patterns in user group behaviors does not provide additional information better than simply taking the average movie rating. One reason for this could be that only 225 movies were represented in the data subset. Trends in evaluation metrics might differ significantly on the entire dataset which has about 17,000 movies. Furthermore, additional features such as movie title information could be included in the model to potentially increase performance by providing more item-centric information. Since the baseline model is completely item-centric, including more information related to the movie itself seems like a promising area of further exploration. Indeed, it is very interesting to note that the baseline model actually performs better than the reported "in house" RMSE that Netflix announced in 2009. One possible avenue of further exploration is to see if the movie rating prediction model performs better for some users than others. Another question is if the model is better at accurately predicting ratings at the extreme end (ex: rating of 1 or 5). It may be

the case that the baseline model is performing so well because there is a significant bias in the dataset for ratings of 4 or 5. Based on psychological research, individuals who respond to surveys may be inherently more likely to rate something higher [10]. Many factors can influence if a user responds to a survey and how they respond. Therefore, it would be interested to learn the context on how survey results were collected to determine if there are any biases at play [11]. This project focused on designing a movie rating classified on a large dataset of 1M rows. Distributed and parallel architectures were described as solutions to effectively perform EDA analysis, transform data, and train/evaluate a simple weighted cosine similarity rating algorithm. While pyspark and dask have been shown to offer significant performance gains in processing large datasets [9], a combination of writing chunks of data to memory and running colab notebooks in parallel was much faster. This is likely due to the fact that maintaining pyspark and Dask operations on dataframes required the dataset to maintain all columns, even if the resulting matrix was very sparse. Breaking the table into individual row vectors and computing vector operations one at a time was much faster since the 0/null values could be dropped. While this solution worked for the resources available, it is important to note that the proposed algorithm is very storage-heavy, and likely will not scale well if storage cannot be easily increased.

The proposed model shows some promise in predicting user movie ratings for Netflix movies. Future studies should focus on enhancing performance through additional features, more sophisticated algorithms, and creative data representation (for example clustering to find user groups and then computing similarity only in that group). Additionally, the rating systems developed can easily be integrated into movie recommendation systems. A proposed pipeline would be to evaluate ratings for 100 unseen movies and return a list of the top 10 predicted movies for the user. More sophisticated techniques could be used such as user profiling analysis and pulling in additional information from other sources such as what songs the user listened to that day (tells you information about their mood which may correlate with the emotions tied to a particular movie).

While recommendation systems have many benefits such as enhancing user experience, decreasing feeling of overwhelming options, and increasing company profits, there are potential concerns that should be discussed. Recommendation systems that do a really good job of finding similar users and recommended content that is very similar to what a user has previously liked can contribute to in-group mentality and reduce an individuals ability to explore new areas and learn new information. In the context of social media, recommendation systems have been linked to increasing polarization, which leads to tensions between groups since they lack overlap in shared experiences/knowledge. One solution is to purposefully build in avenues for recommendation systems to recommend out-group content [12], or decrease the similarity threshold.

V. Conclusion

The dataset, originating from the Netflix Prize open competition on Kaggle, provided a robust foundation for our experiments. The selection of a single file containing 24 million rows allowed us to navigate the challenges associated with large-scale data. Our choice to ignore date and title information aimed to streamline architecture development and address hardware constraints. Focusing on the first 1 million rows for proof-of-concept allowed us to create a customized train/validation/test split, ensuring a rigorous evaluation of our proposed algorithm.

The architecture flow, as depicted in Figure 2, outlines the processing steps for the Netflix challenge dataset. The original format of the dataset posed a challenge since it was not in the standard dataframe format. Leveraging a Python script facilitated the transformation, resulting in 225 individual CSV files, each labeled with a movie ID and containing corresponding customer ratings that could be easily loaded into a dataframe and processed in a distributed manner.

The subsequent step involved vectorizing the data, where each movie had columns for customer IDs and values as normalized ratings. To optimize efficiency, parallelization across multiple Colab notebooks proved more time-efficient than using Dask for the entire dataset. This is likely due to the fact that the entire dataset is very sparse, so using dataframe operations, which require all data rows to have the same number of columns, would have been 255x 280,000 matrix computations. By breaking down the calculations into individual ones, we were able to only consider non-null columns, which reduced the matrix calculations to maximum 1x118,413. However, most computations were actually much smaller, the average matrix operation size was 1x4,411. The vectorization resulted in train, validation, and test sets, crucial for the subsequent training of the classification model.

Several similarity algorithms were researched. Singular value decomposition (SVD) is often used in matrix-like datasets to identify compact representations of the dataset. SVD may help simplify a complex dataset and reveal patterns and relationships between the user and movie groups. However, SVD does not work well on sparse datasets [2]. Pearson's correlation coefficient is another metric that can be used to measure the similarity between vectors. Pearson is useful when the scale of the data matters. Since our dataset has very variable column sizes and there are many 0/null values which would impact the calculation of Pearson's correlation, it would not be a good choice [3]. Cosine similarity is a measurement that computes the angle between two vectors and is not sensitive to the magnitude or null/zero values. Based on the analysis, cosine similarity was the most appropriate for this task. The validation set played a pivotal role in determining the optimal number of similar users (k) for the weighted average calculation (Equation 2). The cosine similarity values, serving as weights, were computed through distributed calculations across multiple Colab instances.

In summary, the experimental methodology, architecture,

and results collectively underline the feasibility and efficacy of our movie rating classification system. Challenges in data preprocessing, parallelization, and optimization were addressed, culminating in a system capable of making accurate predictions in a scaleable manner. Item-based filtering was applied instead of user-based filtering for three reasons. First, item-based filtering resulted in more dense feature vectors, since a given movie has many watchers, but a given individual would only watch a few movies. Filtering based on the user would result in very small intersection vectors. Second, much of the user data was incomplete, since we selected only a portion of the entire Netflix dataset. On the other hand, the movie grouping was complete for 255 movies. Lastly, the user information was completely randomized, so that there was no other useful information that could be used to develop a user persona. In contrast, the movies had titles that could be used in clustering algorithms or as additional features for comparison.

An additional point to note is the trade-off between storage and RAM. The chosen architecture leverages storage-intensive operations, making it less reliant on high RAM capacity. This design choice aligns with the resources available in the Google Colab environment. However, for contexts where RAM limitations are more restrictive than storage considerations, alternative architectures might be explored.

The dataset used in our experiments excluded title and date information for the sake of streamlining development and addressing hardware limitations. However, incorporating these features has the potential to elevate predictive accuracy. Movie titles, when transformed into a word embedding space through techniques like bag-of-words or word2vec, could capture semantic relationships that describe user ratings [6]. Additionally, the temporal dimension introduced by date information might unveil patterns linked to changing viewer preferences over time.

As a next step, employing dimensionality reduction techniques such as clustering and Principal Component Analysis (PCA) holds promise. The inclusion of title and date features could expand the feature space considerably. Applying clustering methods might identify latent structures within this expanded feature space, while PCA could effectively reduce dimensionality, potentially leading to a more streamlined and efficient model. Additionally, the rating system could be used to provide recommendations. For example, the algorithm could randomly select unseen movies from the movie database and predict the user's rating for the movie. If a movie is predicted to be 5, then automatically return that movie as a recommendation. Otherwise, return the top result out of x number of movies analyzed. If more information about the user's preferences were known, the algorithm could be modified to return several movie recommendations from different genres that the user has historically enjoyed. However, this requires that the movie dataset be sorted or grouped in certain ways, or if not that filtering step would take additional computational resources.

REFERENCES

[1] Cai Y, Yu F, Kumar M, Gladney R, Mostafa J. Health Recommender Systems Development, Usage, and Evaluation from 2010 to 2022: A Scoping Review. Int J Environ Res Public Health. 2022 Nov 16;19(22):15115.

[2] Postmus, S. (2018). Recommender system techniques applied to Netflix movie data Research Paper Business Analytics.

[3] Prasad. (2023, August 25). Understanding Pearson and Cosine similarity - prasad - medium. Medium. https://prrasad.medium.com/understanding-pearson-and-cosine-similarity-f7a9afc22e1a

[4] Díaz Romo (2018) Application of Imputation techniques in Collaborative Filtering-based Recommender Systems

[5] Collaborative filtering. Google for Developers.

[6] Vadloori, K., Sanghishetty, S. M. (2021). Exploratory and sentiment analysis of Netflix data. ResearchGate.

[7] arpanghoshal/EmoRoBERTa · Hugging Face. (2001, June 1). https://huggingface.co/arpanghoshal/EmoRoBERTa

[8] Danofer. (2021, February 24). Deep learning for Netflix Prize challenge. Kaggle. https://www.kaggle.com/code/danofer/deep-learning-for-netflix-prize-challenge/notebook

[9] A performance comparison of Dask and Apache Spark for Data-Intensive Neuroimaging Pipelines. (n.d.). IEEE Conference Publication — IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8943502

[10] Mazor, K. M., Clauser, B. E., Field, T. S., Yood, R. A., Gurwitz, J. H. (2002). A demonstration of the impact of response bias on the results of patient satisfaction surveys. Health Services Research, 37(5), 1403–1417. https://doi.org/10.1111/1475-6773.11194

[11] Response Bias - Biases and Heuristics — The Decision Lab. (2023, August 31). The Decision Lab. https://thedecisionlab.com/biases/response-bias

[12] Santos, F. P., Lelkes, Y., and Levin, S. A. (2021). Link recommendation algorithms and dynamics of polarization in online social networks. Proceedings of the National Academy of Sciences of the United States of America, 118(50). https://doi.org/10.1073/pnas.2102141118