Final Project: Building a Movie Recommendation System

Nanyang Technological University (NTU)

November 2022
By Maëlys Boudier (N2202193F)

Table of Contents

Problem Statement:

Netflix, Roku, Disney+ and other digital streaming platforms strive to keep users engaged by providing content recommendations. The more users they gain, the higher their revenue, which incentivizes these companies to build accurate and reliable recommendation systems. This study will provide insights on how to build a movie recommendation system which infers whether a user will like a specific movie based on a variety of features. The first algorithm will rely on Bayesian Probability which will determine the probability of a user liking a movie based on their watch history and movie metadata. However, other algorithms including Logistic Regression and Decision Tree Classifiers will also be tested. To achieve high accuracy, multiple methods will be compared in this paper and the best features will be combined to create the final algorithm.

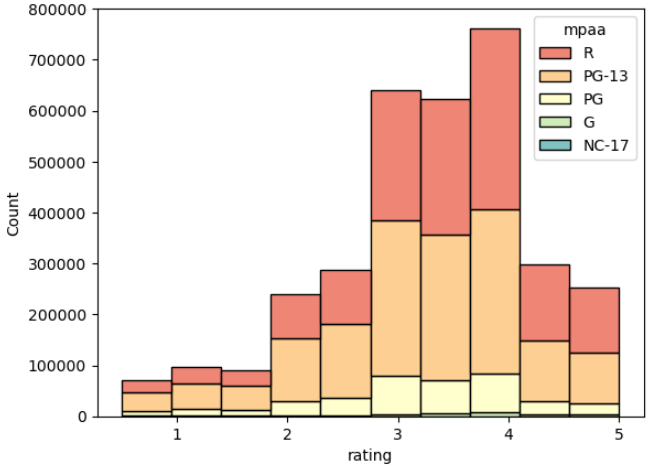
First, an overview of the dataset will be presented, followed by a detailed methodology. Then, the study will highlight the evaluation of the algorithm, results, and a discussion to expand on the limitations, possible improvements, and applications of the model.

Data:

To build the movie recommendation system, two datasets were combined: data_rating.csv and data_movies.csv. The first file, data_rating.csv, provides users' watch history and associated rating assuming that rated movies have been watched by the user and contains 3,363,773 rows. The three columns in the ratings file are "user_id", "movie_id" and "rating". This study will consider any rating of 4 and above as a 'liked' movie, and anything below as disliked for recommendation purposes. The second file, data_movie.csv, consists of 22 columns with metadata about the movie including the title, description, genres, actors and more and contains 1,144 rows. Each row corresponds to a different movie with its own unique movie id. The two datasets were merged thanks to the "movie_id" feature present in both datasets. Despite all the movies being unique, one movie title appeared twice: "The Girl with the Dragon Tattoo", but this was left untouched as it did not seem to present an issue to future analysis and models.

After merging the two datasets, a few key statistics were computed. Notably, 2,050,762 movies have a low rating (less than 4.0/5) against 1,313,011 movies with a high rating. This is likely due to choosing 4 as the cut-off value instead of using the top 10% of values as high per user given that some users may be harsher than others. The histogram below plots the distribution of the ratings variable with MPAA (film ratings for recommended viewer age) as the hue.

Histogram: Distribution of Ratings (out of 5) with MPAA as the hue



Each user id is associated with a list of movies which have been reviewed; the minimum amount of movies viewed by a user is 199 and the maximum is 1127 movies. The mean and median are at 328 and 286 movies respectively with a standard deviation of 129 movies. The outer bounds are particularly interesting, as users with a large watch history provide more specific data and context on their preferences than those around the minimum; however, the minimum values are already quite high.

After looking through the data, some columns seemed correlated or could provide further insights when combined. Indeed, each movie's profitability was computed by using the 'worldwide' revenue minus the 'budget' features; 1039 movies were profitable against 105 movies which generated a loss. The tables below show the key statistics in the merged datasets (with ratings & movie metadata).

Key Statistics of Numerical Variables

	*Rating**	*Year*	*Run Time*	*Budget*	*Domestic*	*Worldwide*	*Profit*
Mean	0.39	2003	118	7.26E7	1.27E8	2.98E8	2.25E8
Median	0	2003	116	6E7	9.71E7	2.03E8	1.40E8
Std	0.49	5.91	22	5.69E7	1.13E8	3.12E8	2.76E8
Min	0	1990	77	700	3047	3047	-9.83E7
Max	1	2019	242	3.56E8	9.37E8	2.80E9	2.55E9

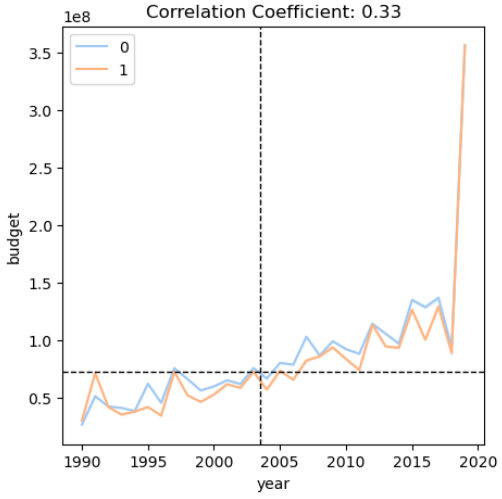
**Rating: binary rating is used to compute metrics*

Key Statistics of Categorical Variables

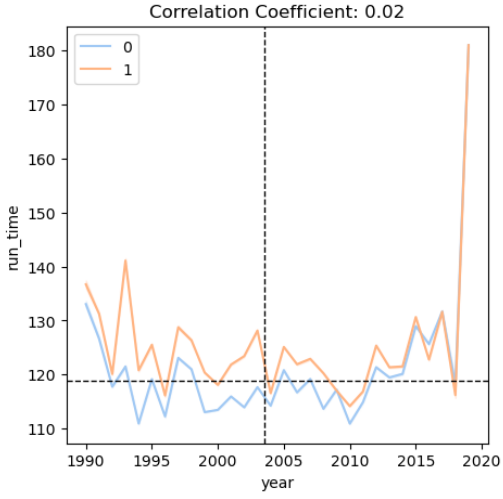
	User ID	*Movie ID*	*Title*	*MPAA*	*Director*	*Composer*
Unique	10249	1144	1141	5	543	291
Top	u03656	m0199	The Matrix	PG-13	Steven Spielberg	John Williams
Count	1127	9807	9807	1538215	74961	130500
Frequency	0.03%	0.29%	0.29%	45.72%	2.23%	3.88%

After building line plots and computing the correlation coefficient for various variables a few interesting patterns became noticeable. The relationship between budget and year is positive, as the movie gets newer, budget increases; this pattern makes sense as blockbuster movies have gotten more budget over the years. This same theory was tested for run time and year, however, there doesn't seem to be a significant trend except for the jump in 2020 (found on both line plots).

Line Plot: Budget & Year



Line Plot: Run Time & Year



Finally, the dataset (once merged) provided data on the 4 main actors and 4 main genres in a movie alongside the names of the director, writer, producer, composer. The actor and genre columns had to be processed and transformed into Booleans ignoring the placement (as users care more having Tom Cruise in the cast, than which position he was given in the dataset). With further analysis, the top 5 actors present in the dataset were: Tom Cruise, Brad Pitt, Matt Damon, Tom Hanks, and Will Smith (in order) which was a very male-dominated podium. Meanwhile, the top 5 genres were: Drama, Action, Adventure, Comedy, and Thriller.

Method:

To build a powerful recommendation system, multiple models with different features will be created and a final 'ensemble' model combining the best features will be applied to a validation set. First, the data will be preprocessed to create a baseline model using categorical and numerical variables. Second, a user-based model using text analytics on movie description and titles and applying Bayesian probability.

Feature Engineering:

In this section, the model will be made using solely numerical and categorical variables. After looking through all the variables, the categorical variables were turned into dummy variables to facilitate usage. Dummy variables are columns for each category with 0 and 1; for instance, there is a column for each genre category with 0 and 1 depending on if the movie belongs to that category. For genres and actors, we combined all the labels found across the 4 columns and ignored the order or positioning found in the original dataset. Given the large amount of actors, only a select top actors were kept as dummy variables and the exact number was 'tuned': tested different numbers to find the optimal amount. Once all the categorical variables had been processed, an iterative process was used to find the best mix of features. This included using all the features and making changes to the combination and noting down whether the error metrics went up or down.

A key concern was how to use the user_id as there were multiple possibilities. On the one hand, a model could be built per user based only on their watch history severely restricting data per model but providing a more individualized model. On the other hand, user_id could be used as a dummy variable and interact with other variables in the model. In this study, the latter was found to be the best to access all the data.

To make the predictions, multiple models in the *sklearn* package were tested but only two ran quickly enough to be deemed efficient algorithms (under 5mins of run time). The two models were Logistic Regression (the most *basic* model) and Decision Tree Classifier. Logistic Regression initially performed poorly, and the issue stemmed from the different data scales. Therefore, the data was scaled using a SimpleScaler() (also found in the *sklearn* package) and the performance jumped up. The scaler was fitted to the train data and the same scaling was applied to the test data as well as the questions data (results were submitted in the CS4050_RESULTS.csv file). The scaled train data was then used to fit a Logistic Regression model and used to predict test and questions data.

Text Analytics: Movie Description & Title

Alternatively, this section presents a model made using solely text data present in the dataset: movie description called 'trivia' and title variables separately. This was initially made as a standalone model to gage how the runtime and accuracy. If the runtime was low and accuracy reasonably high, further features could be added at a later stage. For this approach, a new model was built for each user and the algorithm was only run on the users present in the validation set but the same algorithm could be applied to other users. The goal was to look at users individually, create a dictionary of words found in the synopsis or title of all the movies they had watched and compute the probability of a movie being liked given the appearance of specific words in the movie description and/or title. These probabilities would in turn identify potential patterns in the appearance of similar words across movies a user liked. Stop words – very common words with little added values such as 'the' – were removed from the dictionary to reduce the amount of

parameters. However, this model resulted in a considerably long run time (almost 5 minutes for a basic form of the algorithm run across only 1000 users – those present in the question_movie.csv dataset). This algorithm also only ran 1 simulation of the algorithm per user instead of multiple simulations which to generate a more reliable estimate of the accuracy. Another major downside of this models was that it limited the quantity data is used to make predictions; some predictions were based on ~200 data points instead of across the similar users. Given the long runtime, this strategy was not explored any further.

Evaluation:

The metrics considered to evaluate each algorithm were: precision, recall, f score and the run time of the code. The first three measures measure the performance of the algorithm whilst the final metric determines the efficiency of the algorithm. The maximum run time allowed 5mins and any models above were not considered.

The first 'model' – random model - in the table below consists of using the prior probability and guessing only the class found in most of the dataset. Although this method could have a seemingly high accuracy it generated a very low average F score but was used as a baseline for a 'bad model'. The text analytics-based models performed quite poorly but had a slightly higher F score than the random guessing, despite taking significantly more time. Interestingly, whether the text analytics methodology was applied to the title or description (e.g., trivia) of a movie made minimal difference in the result scores but using title reduced run time by 1 minute. This may be due to titles using strategic key words or allowing the algorithm to predict movies in the same series (e.g., The Matrix, The Matrix Reloaded, The Matrix Revolutions, The Matrix Resurrections…).

Notable Model Results:

Model Type	Features	Precision	Recall	F Score	Run Time
Random	Guessing all 0s (on all data)	0: 0.61 1: 1.00	0: 1.00 1: 0.00	0: 0.76 1: 0	0:13min
Bayes	Text Analytics: Title	0: 0.65 1: 0.48	0: 0.70 1: 0.36	0: 0.64 1: 0.36	3:30min
Bayes	Text Analytics: Description	0: 0.64 1: 0.47	0: 0.68 1: 0.37	0: 0.65 1: 0.39	4:30min
Logistic Reg	Features: Genre, Actors (top 100), Budget, Year, Run Time	0: 0.62 1: 1.00	0: 1.00 1: 0.00	0: 0.77 1: 0.00	0:37min
Decision Tree	Features: Genre, Actors (top 100), Budget, Year, Run Time	0: 0.73 1: 0.60	0: 0.80 1: 0.50	0: 0.76 1: 0.54	3:16min
Logistic Reg + Scaler	Features: Genre, Actors (top 100), Budget, Year, Run Time	0: 0.74 1: 0.67	0: 0.85 1: 0.5	0: 0.79 1: 0.58	0:49min
Logistic Reg + Scaler	Features: Genre, Actors (top 400), Budget, Year, Run Time	0: 0.75 1: 0.68	0: 0.84 1: 0.54	0: 0.80 1: 0.60	1:28min

Note: when calculating the features, all zero divisions were replaced by 1 for simplicity.

The text analytics-based models using Bayes theorem performed quite poorly compared to the feature engineered algorithms and took a considerable amount of time to run. The main benefit of the text analytics algorithm is to explore an alternative way of creating features; it implies creativity but also takes a considerable amount of effort to clean the data compared to regular categorical and numerical data.

However, this model took a while to run and performed poorly compared to the Logistic Regression and Decision Tree Classifier. The Decision Tree's main weakness was overfitting the data and achieving a train accuracy of 1 whilst having a test accuracy of 0.64. The Logistic Regression Model (with scaled data) did not encounter such issue and resulted in a train and test accuracy of ~0.74. Overall, the Logistic Regression with scaled data with the features in the above table performed the best.

The main benefit of the logistic regression was its simplicity. The final choice to make was which version of the logistic regression to use, more specifically determining how many actors to use as predictors. Using 100 actors has a run time of 49 seconds while 400 actors achieved a slightly higher F score in 88 seconds. Given that the goal of this study is to have accurate results and be efficient (defined by Prof. Poong Oh as 'under 5mins'); the slightly more complex model was deemed the best as it met both criteria.

Results:

The "question_movie.csv" rating predictions were computed based on the Logistic Regression algorithm with scaled data (the only code submitted for review). The features used were genre, actors (top 400), budget, year, and run time. This dataset consisted of 1000 user-movie pairs and were filled with True/False depending on whether a specific movie should be recommended to a specific user. Notably, all the users and all the movies present in this dataset also appeared in the main dataset which should lead to similar results as in the train set.

Discussion:

Limitations

Part of the strength of the algorithm lies in accessing each user's watch history and movie metadata. Recommending movies to a new user may have poor results given that there would be no user data. To remedy this problem, user metadata (e.g., age, gender, location…) would be necessary to find similar profiles based on demographics. Also, if the streaming platform were to add new features on their platform such as new genre categories (different names or entirely new category), the model would generate poor results as the new data wouldn't match the data used when building the model.

Another limitation would arise if two separate users share the same profile and have very different tastes. The algorithm may have difficulty recommending movies but platforms like Netflix have overcome this challenge by allowing users sharing an account to create different watch profiles. Indeed, users may share an account but have two separate profiles and 1 shared profile (to watch movies together), but this separation is user specific.

Strategies to Improve Model

The first strategy to improve results would be to normalize each users' scores and only consider the top percentage instead of setting a pre-defined cut-off (at 4/5) as some users are harsher than others when rating movies. For instance, a specific user may rate movies between 3 and 5 and truly only 'like' movies from 4.5 to 5. Alternatively, a user may rate movies in the lower portion of the range (1-4) and have very select

movies around 4 despite liking all the movies above 3.5. In this project, the algorithm only considers a set cut-off, but this strategy could be pursued to create a more reliable model.

The second strategy to improve accuracy from a long-term perspective would be to implement a 'rolling' algorithm and constantly update the base probabilities. In this study's case, the train and test sets are fixed, and the final model is trained across all the data before being applied to the 'question_movie.csv' set. However, if a streaming platform where to implement this algorithm they would be able to update the user's watch history and confirm whether a user watched a recommended movie (and liked it).

Finally, accessing user metadata such as their demographics could be useful to create user profiles. Using these profiles, user similarity could be a stronger feature in the recommendation system. Most streaming platforms ask for this information when users sign up; however, this would be limited to the creator of the account and not sub profiles (typically shared across family members of different ages).

Use Cases

Overall, a recommendation algorithm can used to optimize dating app matches by providing users with profiles they are likely to swipe on or matching users with job opportunities on LinkedIn. The algorithm could be improved by optimizing mutual swipes. An alternative use case for a powerful recommendation system is the autocorrect/word predictors many phones have on the virtual keyboards; most phones provide a prediction of the next word to be typed based on a series of previous words.