Movie Recommendation Engine and Feedback Analysis

Alexander Karl, Ankit Kumar, Edwin Amponsah, Ross Brancati, Sathvik Thogaru[*]

December 9, 2021

Abstract

This project aims to understand the self-modifying effects of recommendation algorithms, which, by recommending movies to rate, effectively create their own data to use in subsequent predictions, which affect those predictions in unknown ways. This project included the design of several Machine Learning recommendation algorithms, the best of which was selected for use in a simulated experiment modeling user viewership behavior under the effect of that recommendation algorithm. Analysis has shown that the feedback loop in recommendation algorithms results in shifting the initial preferences of users over time. We also show that more popular movie genres dominate the movie recommendations due to popularity bias while less popular movies wither out over time.

Keywords: Recommendation Algorithms, Machine Learning, Simulated, Feedback loop, Popularity bias.

1 Introduction

Recommendation engines, as in Netflix's movie recommendations, prompt the user to provide ratings on the recommended movies, which then feeds back into the engine for future

[*]University of Massachusetts Amherst

predictions. When a person watches and rates a recommended movie, the data available to the engine changes, and the next recommendation is based on that new data. In this way, recommendation engines create their own feedback, which could plausibly change their own recommendations over time.

Any recommendation engine effectively performs data collection via recommending specific movies for viewers to watch. This is non-random data, and is hallmarked by the selection bias of the algorithm, as well as the initial conditions as represented by the viewer's starting data. That this selection bias in gathered data is an effect is clear. The design of this experiment is to analyze the extent of that effect.

There are a couple of different approaches to generating recommendations. One of these approaches uses characteristics or features of users to recommend movies that the user highly rated to similar users. Another approach is collaborative filtering, which exploits a user's past behaviors to generate the recommendations [2]. In the sense of movie recommendations, this past behavior would be the ratings that the user provides for movies that they have viewed. One disadvantage of the content-based approach is that it does not account for the dynamic nature of users as they can change their interests over time. Since our project revolves around simulating new user ratings, it was more appropriate to use a collaborative filtering approach. A drawback of this approach is the cold-start problem, described as difficulty with providing recommendations to new users [3], but fortunately we were provided quite a bit of initial user data so this was avoided.

The two most common methods in collaborative filtering are the nearest neighbor and matrix factorization approaches. The nearest neighbor algorithm is a relatively straightforward item based approach - it generates predictions of new data points by voting on the number of data points that are closest to it, but is only an item-based approach that uses movie ratings to generate predictions. Our matrix factorization method incorporates characteristics of the users such as demographic information. Ultimately, we chose the matrix factorization approach for our recommendation engine.

Our project sought to answer a few different questions: Do the initial preferences of users change as more movies are seen? Are the users recommended with a varying set of movies over time? Do recommendations converge to highly rated genres as more movies are viewed?

The paper is organized as follows. Our methods are outlined in section 2, experimental results are in section 3, and the conclusions/summary follow in section 4.

2 Methods

This section contains details on building the recommendation engine and the subsequent experimental simulation.

2.1 Recommendation Engine

The first part of this project was to build a recommendation engine with the goal of generating recommended movies for a given user's rated movies. This engine was then used in the simulation portion of the project.

2.1.1 Item-Based K-Nearest Neighbors (KNN)

In machine learning, K-Nearest Neighbors labels a new data point by taking a vote with its nearest neighbors, where the number of neighbors selected is denoted by the hyperparameter, K. The metric used to determine the nearest neighbor is typically euclidean distance, however this metric can suffer from the curse of dimensionality [1] as the matrix grows to large dimensions. Instead, cosine similarity evaluates the similarity of the angle between two vectors because it tends to work better in higher dimensions. The cosine similarity between two non-zero vectors (A and B) can be determined by the dot product, where θ is the angle between the two vectors [5].

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

$$CosineSimilarity = cos(\theta) = A \cdot B||A||||B|| = \Sigma_{i=1}^{n} A_{\text{i}} B_{\text{i}} / (\sqrt{\Sigma_{i=1}^{n} A_{\text{i}}{}^2} \sqrt{\Sigma_{i=1}^{n} B_{\text{i}}{}^2}) \tag{2}$$

The algorithm is trained with the existing data and a vector of movies consisting of the user's top n rated movies is passed through the function, where n represents the desired number of movies to use in determining the recommendations. The recommended movie is calculated using the following equation

$$F_{\text{KNN(x)}} = argmax_{y \epsilon Y} \Sigma [y_{\text{i}} = y] \tag{3}$$

Where y is the known data point and y_{i} is the data point that is being classified. This equation is taking a majority vote of the nearest number of neighbors.

Our goal was to find the ratings and recommend movies using the content based or collaborative based filtering techniques. Being an item-based collaborative filtering technique, KNN suffers from limitations arising due to absence of explicit information about the user's preference for the item. It is highly likely that users do not rate a movie after watching it, or they do not watch a movie at all, thus creating a situation where we have highly sparse data on movie ratings. We would also like to incorporate the user's demographic and related information into our analysis. Thus, the natural way forward was to somehow utilize both the user and item based features into our analysis. This led us to develop a recommendation engine based on Matrix factorization.

2.1.2 Matrix Factorization

Suppose we have n number of users and r number of movies(items). We define a matrix M to be the utility matrix that has user ratings for movies (n x r), U to be the matrix that represents users (n x k), and V to be the matrix that represents items or movies (k x r). The matrix M can be decomposed using Singular Value Decomposition [4] [6] into the following:

$$M = U\Sigma V^T \tag{4}$$

where Σ is a diagonal matrix which contains weights corresponding to the strength of each latent feature. Decomposing the utility matrix allows us to capture the strength of the underlying latent features corresponding to the preferences of the users and movies. While users may exert their preferences through demographic characteristics and other information, movies may wield their influence through genre, year of release, etc. We then used Matrix Factorization to recommend movies to the users.

NA values were imputed in the utility matrix with 0 and then normalized with respect to the average rating for each movie. Matrix factorization was applied on the matrix M using the svds function from the scipy library. After decomposing, the process of normalisation was reversed to generate actual ratings for each movie. An iteration using a for loop with K from 1 to 40 was used to choose an appropriate K value by plotting the error rate and the K corresponding to each iteration. For K greater than 30, it was found that the error rate tapers. Thus, 30 became a natural choice for K. The top 10 movies based on ratings from the matrix factorization were selected for the recommendation.

2.2 Simulation

Simulation was used to generate data on how the recommendation engine results change after repeated viewings and ratings. 20 iterations of the simulation were run, to model a series of 20 sequential movie viewings. Each simulation involves all 2,354 users in the base data. All users data are fed back into the model after every movie rating, which allowed every user's updated ratings to effect every other user's recommendations from the next run of the model.

Each simulation event consisted of running the recommendation engine, finding the five highest predicted movies for every user, and selecting one of those movies per person to apply a random rating to. That set of ratings was then appended to the data available to the recommendation engine, which it to use in all future generations of that iteration.

To mimic real world behavior around viewer choice, five recommended movies were produced per person, and one chosen at random. This also had the important effect of adding variance between simulation iterations, ensuring alternate sets of movies were reviewed.

That movie was given a random rating between 1-5, modeled by the equation

$$Max(1, Min(5, 5.5 - \exp(4 * (1 - distance)))) \tag{5}$$

This formula assumes that recommended movies will generally be highly rated. To allow for a chance at some middling ratings, and a few very low ratings, the formula includes a term to subtract a random exponential value. By incorporating the distance variable calculated by the Matrix Factorization, the rating assumes that recommendations made with more confidence will also tend to be rated higher, whereas weaker predictions were allowed to skew lower.

3 Experimental Results

Here we present our analysis of the simulation data with focus on the impact of feedback loop on the recommendation engine and how it shapes user preferences over time as more movies are seen. We discuss key issues including changes in group preferences over time, homogenization of user experience and popularity bias amplification.

3.1 Changes to Group Preferences Over Time

We define the preference of the group using the genre of the movie seen. The genre distribution is calculated as the ratio of the movies watched associated with the genre over the total movies watched grouped according to genre. We then calculate the Kullback-Leibler divergence between the initial preference and the genre distribution in each iteration for the group. From figure1 there was a higher deviation from the initial preference over time. This suggests that the taste of the users shift over time as more movies are watched by users which is a direct consequence of the feedback loop in the recommendation engine.

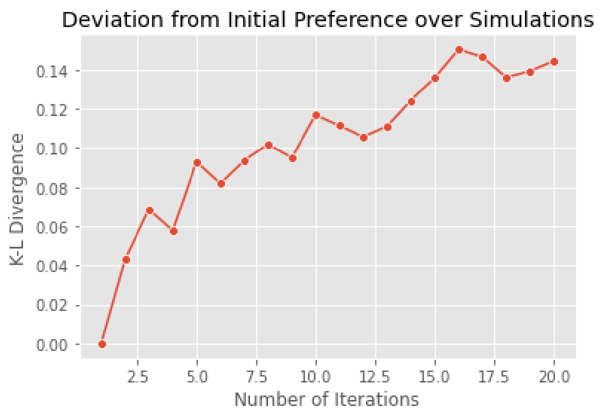

Figure 1: Deviation from Initial Preference over Simulations

3.2 Homogenization of User Experience

It is important for a recommendation engine to balance the recommended items to cover the user's whole set of interests. We assess the impact of feedback loop on this key need by studying the number of unique sets of movies that are recommended to the group over time as more movies are seen. We also compute the Simpson's Index of Diversity, a metric originally developed to assess biodiversity in ecology, for each iteration. From figure2 and figure3, it shows that more unique sets of movies are recommended over time as more movies are watched by users.

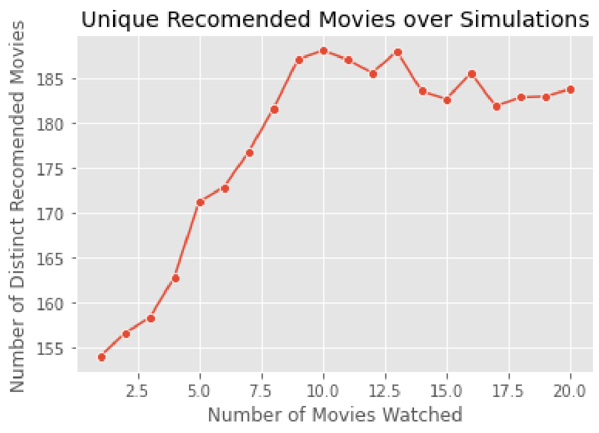

Figure 2: Unique Recommended Movies over Simulations

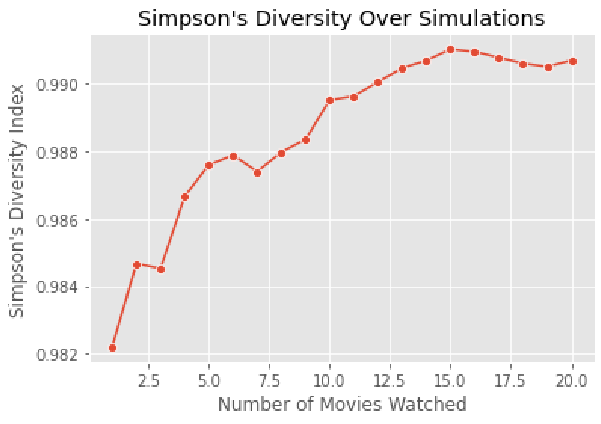

Figure 3: Simpson's Diversity over Simulations

3.3 Popularity Bias Amplification

Recommendation models can intensify the popularity bias in input data over time due to the feedback loop. From figure4, comedy (which is the most popular genre) is more recommended over time. On the other hand, Documentary (which is the least popular movie) is less recommended over time and dominated by more popular genres. We conclude that more popular genres are more recommended over time while less popular genres are likely to wither out as more movies are watched by users.

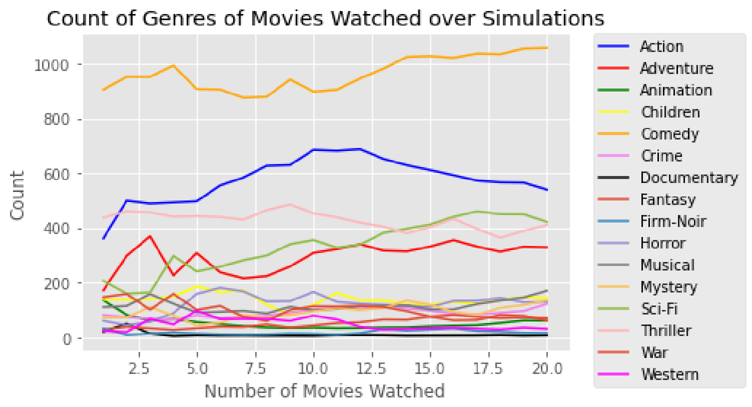

Figure 4: Counts of Genres of Movies Watched over Simulations

4 Conclusions and Summary

Here we present data from a simulation of movie ratings. Our results show that there was a higher deviation from the initial preference over time. This suggest that the taste of the users shift over time as more movies are watched by users. Moreover, more unique set of movies are recommended over time as more movies are watched by users. Lastly, more popular genres are more recommended over time as more movies are watched by users.

Acknowledgments

We would like to thank Professor Flaherty and Zhou Tang for their help throughout this project and the semester.

References

[1] J. Grus, *Data Science from Scratch: First Principles with Python*, O'Reilly Media, Sebastopol, CA, 2019.

[2] S. Kangas, *Collaborative filtering and recommendation systems.* VTT Information Technology, Jan. 2002, http://virtual.vtt.fi/virtual/datamining/publications/collaborativefiltering.pdf (accessed 2021-08-12).

[3] X. N. Lam, T. Vu, T. D. Le, and A. D. Duong, *Addressing cold-start problem in recommendation systems*, ICUIMC '08: Proceedings of the 2nd international conference on Ubiquitous information management and communication, (2008), pp. 208–211, https://doi.org/10.1145/1352793.1352837.

[4] J. leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*, Cambridge University Press, Cambridge, United Kingdom, 2020.

[5] N. I. of Standards and Technology, *Cosine distance, cosine similarity, angular cosine distance, angular cosine similarity.* Statistical Engineering Division Dataplot, July 2017, https://www.itl.nist.gov/div898/software/dataplot/refman2/auxillar/cosdist.htm (accessed 2021-08-12).

[6] Vaibhav Kumar, *Singular Value Decomposition (SVD) and Its Application In Recommender System*, 2020, https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/ (accessed 2020/08/12).