# Alleviating Choice Paralysis for Cinephiles

John F. Desiderio, Jarod S. Gilliam, Jordy F. Klustner

May 2022

## 1 Abstract

With the high volumes of movies that exist in the world today, simply choosing a movie to watch has become a daunting task for cinephiles. Fortunately, recommendation systems could be a potential solution. In this report we attempt to build and employ 3 unique recommendation algorithms to recommend movies for cinephiles. Among these algorithms include both content-based and collaborative filtering algorithms. Data were obtained from an extensive movie rating set via Kaggle.

## 2 Introduction

Information has long been the foundation of human progress. From the beginnings of written language to Claude Shannon's discovery of information entropy, humankind's collaborative advantage is somehow still improving. Computers have since been able to capture and maintain information at a magnitude unbearable for a single human's mind. Instead of allocating our time utilizing what information we have, we now spend most our time filtering the massive amounts of information produced and stored in computers. It is no wonder why recommendation systems have become a vital part of our society.

First introduced in the late 1970's, recommendation systems attempt to alleviate the task of retrieving the information that most closely pertains to an individual's wants and needs. Today, recommendation systems are a part of every day life. Consider your daily news feed, or the products suggested for you on Amazon, or even the songs Spotify says you might like; all attempt to filter the high volume of noise an individual experiences each day.

Recommendation systems are primarily classified into two types of algorithms; content-based and collaborative filtering. Content based algorithms take a users history with certain items and finds other items similar using the different attributes of those given items. Collaborative filtering algorithms work in a slightly different way. These recommenders, find other users, or in some cases items, that are similar to the given user. Items highly rated by the set of similar users are then recommended to the user at hand.

In this report, we will explore the two kinds of recommendation systems via three implementations. More specifically we will test these systems' ability to recommend movies to cinephiles.

## 3 Problem Definition

Over the last century, movies and television have replaced books and theater as the most popular medium for entertainment. During this period, over 600,000 movies have been made and documented on databases such as IMDB. This high volume of movies has led to a common problem among cinephiles; "what movie should I watch next?".

With the help of three recommendation algorithms, we will attempt to alleviate this choice paralysis in this report. The dataset we will be utilizing for this report comes from the "Movies" dataset on Kaggle. It contains information about 45,000 unique movies. These movies are described by user IDs and user ratings which will be the framework for our collaborative filtering algorithms. We will additionally build a content-based algorithm that will use additional features like genre, rating frequency, cast, directors, etc. In total there are 26 million ratings generated from 270,000 users [1]. We hope to successfully employ these three algorithms alongside the data to generate recommendations that all movie-lovers could use and appreciate.

## 4 Proposed Methods

In order to determine how to best solve this problem, we chose a sample dataset [1] and picked and then evaluated on said dataset three different data science methods. We believe that this dataset is a good representation of the kind of data any system built around solving this problem would have access to and we believe it has enough values to both be a valid representation of how much data such a real world system would have and be enough to fully and accurately check the different recommender systems we have in mind. Said methods are Neural Graph Collaborative Filtering, Factorization Machines, and Content-Based Recommendations. You can view the code we wrote for all three here: https://github.com/JohnDesiderio/Data_Mining_Final_Proj.

### 4.1 Neural Graph Collaborative Filtering

Neural Graph Collaborative Filtering was designed to fix a perceived problem in most other recommender systems. The authors state that each other method has an "embedding function [that] lacks an explicit encoding of the crucial collaborative signal" [4]. The authors go on to explain how the collaborative signal "reveal[s] the behavioral similarity between users (or items)" [4] and how they believe that including the collaborative signal in the embedding function is vital to giving the user accurate recommendations. The author's proposed solution is utilizing high-order connectivity in a number of embedding propagation layers. Given a graph of the connections between a set of users and a set of movies they have watched, high-order connectivity describes the connection between a user and any other movie or user as long as the path to get from the original user to this other user or movie passes along the lines of the graph and through at least one node before arriving at that that other user or movie. As an example, the set of users that have watched the same movie as the target user are high-order connected to the target user. The set of movies that each of those connected users have watched are also high-order connected to the target user, just with three movements along the graph required instead of two.

In order to embed this functionality into Neural Graph Collaborative Filtering, the system builds on top of traditional recommender systems like Matrix Factorization and Neural Collaborative Filtering by refining the normal embeddings. It does this by "propagating them on the user-item interaction graph" [4]. In order to do so, it implements the following function:

$$E^{(l)} = LeakyReLU\left((\mathcal{L}+I)E^{(l-1)}w_1^{(l)} + \mathcal{L}E^{(l-1)} \odot E^{(l-1)}w_2^{(l)}\right)$$

Where $E$ is the array containing the user and item representations, $l$ is the number of steps of the embedding propagation, thus $E^{(l)}$ is the user and item representation at that step. In this context, a step corresponds to a movement along the connections in the graph as described above. Thus $E^{(1)}$ represents the movies a target user has watched and $E^{(2)}$ represents the users who have watched the same movies as the target user. $W_1^{(l)}$ and $W_2^{(2)}$ are the trainable transformation matrices, $I$ is the identity matrix and, $L$ is the Laplacian Matrix. In the paper it is stated that the Laplacian Matrix is calculated by:

$$\mathcal{L} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \; and \; A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix}$$

Where $D$ is the diagonal degree matrix and $A$ is the adjacency matrix which is calculated using $R$, the user-item interaction matrix. However, in the author's implementation [5], this is not how they chose to implement the Laplacian matrix. Assuming this was an optimization, and since this part of the code does not use tensorflow, a technology that was avoided because of technical problems, the author's implementation's calculations for the Laplacian matrix was mirrored instead of the paper's calculations.

Neural Graph Collaborative Filtering uses BPR as the loss function:

$$Loss = \sum_{(u,i,j)\in 0} -ln\ \sigma(\hat{y}_{ui} - \hat{y}_{uj}) + \lambda\|\mathbf{p}\|_2^2$$

Where $u$, is a user, $i$ is an item they interacted with, $j$ is an item they did not interact with, $O$ is the set of all such groups of three items, $\sigma$ is the sigmoid function, $p$ is the set of all trainable model parameters, and $\lambda$ controls regularization strength. The combination of these three systems allows for the collaborative signal to be introduced into the model's embeddings, theoretically allowing for a better final prediction.

## 4.2 Factorization Machines

Factorization Machines (FMs) are a sub-sect of collaborative filtering recommendation systems that have been utilized in many real-world applications like Netflix and Instacart. FMs are simply an extension of general linear models and use interactions between features to make predictions. As it pertains to recommendations, the interactions between users and items within a given domain will be used. FMs have the following form:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} < v_i, v_j > x_i x_j$$

In the equation $w_0$ is the global bias. $w_i$ describe the weights of the ith variable. As it pertains to our project, our weights will be simply be the ratings of users for each user-movie interaction normalized to $w \in [0, 1]$. For example, if a user $u$ rated a movie $m$ 3.5 out of 5 stars, $w_{u,m} = .7$. $v_i$ denotes the the $i^{th}$ row of the feature embeddings matrix $V$. This algorithm yields a time complexity of $O(kd^2)$, however, the matrix factorization term can be reformulated to:

$$\sum_{i=1}^{d} \sum_{j=i+1}^{d} < v_i, v_j > x_i x_j = \frac{1}{1} \sum_{l=1}^{d} ((\sum_{i=1}^{d} v_{i,l} x_i)^2 - \sum_{i=1}^{d} v_{i,l}^2 x_i^2)$$

Which leads to a time complexity of $O(kd)$, greatly improving the efficiency of the algorithm [3].

We included this algorithm in the project because it has several advantages that could serve us well with application to movie ratings. The matrix factorization element of the model equation allows it to handle sparse data. This is useful for movie ratings because not every user has a rating for every movie, even among movies a user has seen. FMs have the ability to train with linear complexity which, in turn, allows them to handle very large data like the movies data with which we are working. Finally, FM's are easy to implement because of their availability in open-source software. Even Amazon Sagemaker has an FM library. We will be using "RankFM" which is a factorization machines library available in Python.

## 4.3 Content-Based Recommendation System

The content-based recommendation system is a proposed method to overcome the choice paralysis with choosing a movie. The idea behind the content-based recommendation system is that the recommender does not need users' data to help return an array of options. The algorithm takes the existing metadata of the movie dataset to perform the Term Frequency–Inverse Document

Frequency algorithm on the set of metadata, and the algorithm returns a similarity score with all the movies in the database. After the algorithm finishes fitting, a user submits a movie id for identification, and the results yield multiple recommendations. The arrangement of the results is in descending order. The upside of the content-based recommendation is that no user data, like ratings, are contingent on the algorithm's success. The metadata is enough to construct strong recommendations. A downside to the algorithm is the lack of overall ratings. There exists the possibility that an algorithm recommends unpopular movies, despite the similar existing metadata. The user does not receive tailored results for their preferences either. They receive results based on the most similar film they searched.

## 5 Experiments

### 5.1 Neural Graph Collaborative Filtering

The implementation of this Neural Graph Collaborative Filtering was completed in Python using torch, numpy, and pandas. Since the size of the data was so massive, the paper indicated that on smaller datasets the system might perform especially well, and the dataset had handily provided an easily downloadable subset of its data as an extra dataset, this smaller dataset was used to run the calculations.

The Neural Graph Collaborative Filtering implementation takes in the dataset, splits it on a 70, 30 split into a testing and training set. As mentioned in the Neural Graph Collaborative Filtering paper, the split is not simply done by cutting the dataset itself. Instead, the split is accomplished by splitting the data for each user. The result is that both the training and testing groups have data points for each user. The number of unique users and movies are also calculated, allowing the system to build an user-item interaction matrix that has an index for all items and users. The program also builds a list of positive examples from the training set, which is all items that a user has interacted with organized into a vector that can be more easily used in the BPR loss. Later the negative examples are created by simply taking a random item in the user-item interaction matrix that is not on this positive list. Finally the adjacency matrix is created and normalized as discussed in a previous section. After this point, the model is run and the results are calculated.

There were many possible experiments that could be run on this implementation, for example varying batch size. However, there are multiple steps in this system in which randomness is introduced which obfuscated the results. This made it very difficult to determine if an increase or decrease in the final output's precision and recall was caused by the change to a variable or simply said randomness. On top of this the paper describing Neural Graph Collaborative Filtering already calculated that 1024 was the best so it was decided to simply use that setting in order to focus on the most interesting and important part of Neural Graph Collaborative Filtering: the amount of high-order connectivity that is utilized in the model. The whole point of Neural Graph Collaborative Filtering is adding in the collaborative signal using said connectivity and what the optimal amount of connectivity was, seemed to be the priority.

We ran the program multiple times and produced the results in Table 1 when the high-order connectivity was set to 1, 2, 3, and 4.

| | **Precision** | **Recall** |
|---|---|---|
| $l = 1$ | 0.11326 | 0.00819 |
| $l = 2$ | 0.12667 | 0.01163 |
| $l = 3$ | 0.13263 | 0.01171 |
| $l = 4$ | 0.09538 | 0.00539 |

Table 1: Comparing $l$ values.

As expected, the connectivity setting of $l = 1$, which represents looking at the movies the users watched, performed badly. $l = 2$ improved upon this. The fact that $l = 3$, which represents looking at the movies the users who watched the same movies as the target user watched, performed the best is considered intuitive by the group. However, based on the paper, the original assumption was that as $l$ increased so would the model accuracy no matter how large $l$ was. However, the paper disproves this showing two of their dataset tests performing better when $l = 3$ than $l = 4$. We are unsure why this happens here, but since it occurs in the paper as well, believe that this is merely a quirk of our implementation or the dataset with which we are working, not an error.

These results show that Neural Graph Collaborative Filtering has a high enough precision to be a viable solution to the cinephile problem.

## 5.2 Factorization Machines

To implement the Factorization Machines algorithm to the data, we made use of the "RankFM" library available in python. For this model we used the "ratings" dataset which includes the user ID, movie ID, and that user's rating of the given movie. The ratings are given on a 5 star scale which we divided by 5 to normalize and use as weights for the algorithm. We additionally added a new feature, "random", which randomly samples a value from the uniform distribution, $U(0, 1)$, which we will utilize for splitting our data.

Considering that our data set is very large, and that my laptop only has 4 GB of memory, we decided to only take a random sample of 20,000 users for use in the algorithm. Doing this results in just under 2 million remaining user-movie interactions. From there, we split the data 3 ways; 70 percent for training, 15 percent for validation, and 15 percent for testing.

We fit the FMs algorithm to the training set using the default hyperparameters to see how it performs using three metrics; hit rate, precision, and recall. Additionally, I asked my cinephilic roommate to rate 20 randomly chosen movies to add to the data, let the algorithm recommend 5 movies, and see how satisfied he is first hand with the output.

| | **Hit Rate** | **Precision** | **Accuracy** |
|---|---|---|---|
| **Training** | 0.668 | 0.196 | 0.071 |
| **Validation** | 0.348 | 0.049 | 0.068 |

Table 2: Metrics on Default FMs

The table above shows an obvious degradation in both hit rate and precision. Naturally, we were concerned, so we continued to tune the hyperparameters. After a long process of trial and error, we found no improvement in degradation. We decided to simply proceed. In the end, our best model used the values presented in the table below. Warp loss function has three facets; user, positive items, and negative items. When a score of a negative item, or, in our case, a poorly rated movie, yields a higher rated prediction than a positive item, the gradient updates to allow for the positive item to be rated higher.

| **HP** | **Tuning** |
|---|---|
| Loss Function | Warp |
| Factors | 6 |
| Max Samples | 40 |
| Learn Schedule | Inv. Scaling |
| Learn Rate | .02 |

Table 3: Optimal Hyperparameter Tuning

Using this set of hyperparameters, our model yielded the following results:

| | **Hit Rate** | **Precision** | **Recall** |
|---|---|---|---|
| **Training** | 0.673 | 0.277 | 0.081 |
| **Validation** | 0.408 | 0.063 | 0.075 |
| **Testing** | 0.409 | 0.062 | 0.075 |

Table 4: Final FMs Metrics

Although again we see a heavy degradation from the training set to the validation set, the validation and testing set have near identical results indicating that our final algorithm is consistent among unseen data. Additionally, the recommendations generated for my roommate yielded two thumbs up suggesting this recommendation system is useful in practice.

## 5.3 Content-Based Recommendation System

Another approach to recommending movies to users is suggesting choices based on similar metadata shared in other films through a content-based recommendation algorithm. These new, suggested options could include relevant information such as the movie's cast, director, genres, and description based on the information in another movie's metadata to help guide the user in their decision. The idea behind the content-based algorithm is to take the metadata and feed it into a Term Frequency–Inverse Document Frequency algorithm that identifies the pertinent information and constructs similarity scores between the different movies. The recommender can produce a variety of options to recommend to the user using the movie's similarity scores.

The process of building the recommender system posed problems not considered in the abstract process of working with the idea. The first problem came with the proposed dataset, The Movie Lens Dataset. The dimensions of The Movie Lens Dataset are 46628 rows by 27 columns. In fitting the model during the creation, the kernel failed every time to fit the model and crashed, unable to work correctly. The solution to the problem was to limit the number of movies selected to help fit the model. There were two options in cutting down to a usable dataset: a random sample selection of the films or a filter to cut down on movies. A random sampling of films would yield the widest variety of results as far as options go, except some of the results recommended are not available on major streaming platforms available to the users, which means the top-choice movie could become useless. A different approach to filtering films is to use one of the tags, the 'vote count' tag, as a tag to sift through the metadata. The 'vote count' tag is where anonymous users voted on a movie, and the movie has a popularity tag. Filtering through movies that held a 'vote count' value of 25 votes or more yielded enough results that were good to work with but would also not break the recommender in the fitting process. The idea behind the threshold was that there were people who watched the film, which meant that the film was not too obscure that it would be too hard to find.

There were problems in evaluating the performance of the content-based recommender system concerning how well it recommended movies. The scale of the project limited the thoroughness of its testing. One solution could address the problem: an author compared his favorite movies with the recommendations and subjected those recommendations to a Mean Average Precision metric. The evaluation began with compiling a list of personal favorite films and looking for recommended movies based on each film. The recommender system returned the top 5 results, and the author picked out movies of interest. The author did this for ten movies they enjoyed and compared the results of the Mean Average Precision. The calculated results of the Mean Average Precision came out to be about 0.0843. From the author's perspective, the algorithm worked well to identify other movies that may be of interest.

The content-based learning algorithm worked well to recommend movies based on similar films. One interesting note regarding the algorithm's ability to recommend movies was how there was a higher similarity score when the algorithm returned films by the same director. The highest similarity score is Woody Allen's Crimes and Misdemeanors. The first movie the algorithm recommends is another Woody Allen film titled Manhattan. The next four returned films also had Woody Allen as the director. Those motion pictures all posted higher similarity scores than the best-recommended films for other movies chosen in the sample. There are two explanations for this phenomenon: Woody Allen's produced many movies that belong to the same genre. The selection of movies becomes concentrated, and when a recommender system performs a similarity score on the movies, those movies' similarity scores reflect a higher similarity. If a director produces films in different genres, their movies do not compile concentrated similarity scores, and the algorithm will recommend a variety of directors.

## 6 Conclusion

After looking at all the results, we were able to compare the three models. Even though we were unable to get results for the content-based recommendation implementation, we believe we can safely rule this out as the best of the three given the fact that it is the simplest and thus the others would not have been invented if this version was accurate enough. Between the other two, Neural Graph Collaborative Filtering has higher precision in the validation stage, however it has much lower recall across the board and a higher precision in the training stages. For this reason, we would use Factorization Machines as our system of choice to solve the target problem. We hope this solution will help aid both hardcore cinephiles and the average person who loves movies to sort through the high volumes of content that exists in the world today and overcome the daunting task of choosing the next movie for one to watch.

## References

[1] Banik, Rounak, 2017, *The Movies Dataset*

https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

[2] Gintare, K., Dziugaite, Roy, D., 2016, *Neural Network Matrix Factorization*

https://arxiv.org/pdf/1511.06443.pdf

[3] Rendle, S., 2008, *Factorization Machines*

https://www.csie.ntu.edu.tw/ b97053/paper/Rendle2010FM.pdf

[4] Wang, X., He, X., Wang, M., Feng, F., Chua, T.-S. , 2019, *Neural Graph Collaborative Filtering*

https://doi.org/10.1145/3331184.3331267

[5] Wang, X., He, X., Wang, M., Feng, F., Chua, T.-S. , 2019, *https://github.com/xiangwang1223/neural_graph_collaborative_filtering*