Movie Recommendation Systems Using Content-Based and Collaborative Filtering

Exam project in Data Science, Prediction, and Forecasting 2023

Lærke Brædder (201906517@post.au.dk)
School of Communication and Cognition, University of Aarhus,
Jens Chr. Skous Vej 2, 8000 Aarhus, Denmark

02.06.2023

Professor: Chris Mathys

Abstract

This project aims to present and investigate various approaches for movie recommendation including a recommendation system with a content-based filtering approach, and three systems with a collaborative filtering approach (and item-based, a user-based, and a model-based system). The purpose of a movie recommendation system is to predict a user's preferences for items in order to provide recommendations for items that the user will like. Collaborative filtering, content-based, and hybrid movie recommendation systems are the most commonly used types of recommendation systems. Content based recommendation systems simply uses comparison of the features of the items, such as genre, cast, or director, to recommend, whereas collaborative filtering uses the previous interactions between users and movies to provide suggestions. Findings of this project are not conclusive, but we discuss some of the advantages and limitations of each approach, and look at how they function in practice.

Keywords: *Recommendation system – Machine learning – Content-based filtering – Collaborative filtering – Matrix factorization*

All code used for this paper is available on Github at: https://github.com/laerkebraedder/datascience2023

TABLE OF CONTENTS

1. Introduction

With the rise of online streaming platforms such as Netflix, HBO, and Amazon Prime, just to mention a few, our options when it comes to movies are rich. There are movies for entertainment or for education, for children or for adults, there are action movies, horror movies or romance movies, and much more. But that also means that manually browsing through an endless amount of content leaves little chance of successfully choosing just the right movie for that Friday night. That is why streaming platforms use recommendation systems. A recommendation system uses various filtering algorithms to predict which items (movies, music, books, etc.) a user is more likely to enjoy. Movies can be easily distinguished and categorized based on features such as genre, language, cast, director, year of release, etc. Using these features, a recommendation system should be able to reliably recommend movies that are an exact match or that have the most in common with our preferred content. However both the number of existing movies and the number of online streaming service users are increasing every day, which means that the quality of the movie recommendation systems is compromised.

1.1. Research Question

In this study we offer 4 different approaches to creating a movie recommendation system varying in complexity. The first three methods will employ content-based filtering and collaborative filtering to create simple recommendation systems. The purpose of these systems in this project is to investigate and gain a better understanding of the different types of features in the data that one can utilize when recommending items. The fourth and final recommendation system uses model-based collaborative filtering approach. It uses machine learning to predict user ratings using a weighted average method for the collaborative filtering approach. To evaluate the accuracy of the model prediction, we use the measures root mean square error (RMSE) and mean average error (MAE). The overall aim of the study is to investigate whether we can reliably employ these methods to recommend movies that a given user will like.

1.2. Content-Based Filtering

A content-based recommendation system analyzes only the items and their features in order to identify similarities between items and recommend them to users. For a given user, an interest profile is created in order to match recommended items to the user's interests. It is thus necessary, when creating a content-based system, that we can properly a) represent the items, b) produce user profiles,

and c) that we have good strategies for comparing the user profile to the representation of the items (Mngomezulu & Ajoodha, 2022).

However, as with all recommender systems, content-based filtering has its limitations. First of all, obtaining data on a larger scale can be difficult, since all of the items to be recommended must be tagged of a variety of features, which can be consuming of both time and resources. Secondly, content-based recommender systems often suffer from a lack of novelty and diversity in their suggestions. Chances are that you already know about the movies that look like the ones you like on a superficial level. A good recommendation system should be able to provide diverse and unexpected results – suggestions that you would not have thought of yourself (The Upwork Team, 2021).

1.3. Collaborative Filtering

Collaborative filtering deals with some of the limitations of content-based filtering by taking into consideration not only similarities between items (cast, director, budget, etc.) but also similarities between users when providing recommendations. In this manner, models that use collaborative filtering can recommend a movie to user A based on the interests (ratings, for example) of a similar user B. Another advantage of collaborative filtering is, that the embeddings are learned automatically by the model, i.e. there is no need to hand-engineer the features as with content-based filtering (Google for Developers, 2022).

2. Methods

2.1. About the dataset

For the majority of this analysis, I used the MovieLens 20M Dataset[1] (Maxwell Harper & Konstan, 2015), which describes ratings and free-text tagging activities form the movie recommendation service MovieLens. This is one of the most widely used benchmark datasets within the field of movie recommendation. It contains about 20 million ratings of 27,000 movies by 138,000 users between January 1995 and March 2015. Every user included in the dataset has rated a minimum of 20 movies.

The dataset contains no demographic information outside of unique user IDs, however that is not an issue with the approaches that we are using in this project. For the content-based filtering, we are not taking the individual users into account at all, since we are working with an item-item matrix, and for

[1] https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset

the collaborative filtering approach, similarity between users is calculated based solely on user behaviour, rather than on any demographic factors such as the sex of the user or the genre of the movie. Two users are considered similar in spite of age, sex, etc. if they give the same ratings to the same movies (Abhinav, n.d.).

However, the MovieLens 20M Dataset does not contain the movie descriptions which are needed for the content-based recommender, so for this system we use The Movies Dataset from Kaggle[2]. This dataset contains metadata on over 45,000 movies released in or before July 2017 and 26 million ratings from over 270,000 users. Additionally, it also contains information on the cast, crew, plot keywords, budget, release dates, languages, etc. of the individual movies

2.2. Content-Based Filtering Recommendation System

With content-based filtering, we create mathematical representations (vectors) of text data. Content-based filtering approaches will often times use Term Frequency (TF) and Inverse Document Frequency (IDF) to define the objects of the data (a collection of descriptors). TF simply describes the frequency of a given term in a given document. IDF describes the prevalence of a given term across all documents in the corpus. TF-IDF, then, is the product of TF and IDF, and it is an expression of how important a term is within a given document relative to other documents, i.e. is the term meaningful in the document given its prevalence across the corpus. The greater the TF-IDF is, the more important the term is for the document (Mngomezulu & Ajoodha, 2022).

For our recommender, we can use the IF-IDF to identify key terms in the descriptions of the movies. We will be working with the "overview" variable in the metadata, which provides short descriptions of the plots of all of the movies. From the overview descriptions, we create a TF-IDF matrix using the TfidfVectorizer from the Scikit-learn feature extraction library (Pedregosa et al., 2011). We then create a cosine similarity matrix and recommend movies to the user based on similarities. The cosine similarity is a metric which focuses on the similarity of two vectors. For this task, we are also going to need the Euclidian distance to find the distance between two vectors.

This way, we are able to recommend movies based on an input movie title. Since this is not a model-based machine learning method, we cannot directly evaluate the accuracy of our recommendation

[2] https://www.kaggle.com/rounakbanik/the-movies-dataset. The original dataset was released by MovieLens website, https://grouplens.org/datasets/movielens/latest/. It is a project run by GroupLens, a NLP research lab at the Univeristy of Minnesota.

system. However, we can use the ratings data to check if the users who rated, say Jumanji, highly did indeed also rate the recommended movies highly, so regression plots were made visualizing the relationship between the ranking of recommended movies and the user ratings. If the recommender is effective, then we would expect that the lover the recommendation ranking number is the higher the rating will be. I.e. we would expect the rating of the number 1 recommended movie to be higher than the number 1000 recommended movie.

2.3. Item-Based Collaborative Filtering Recommendation System

For the item-based collaborative filtering approach, recommendations are made based on item similarity. For example, if we liked Finding Nemo, then we might want to be recommended other movies that show the same liking structure, or that have the same director, cast, etc.

2.4. User-Based Collaborative Filtering Recommendation System

For this approach, movie recommendations are made based on similarities between different users. The process goes by determining which movies the user of interest has already watched, and then accessing the data from other users who watched the same movies. We can then identify other users, who display similar behaviour to our user of interest, to suggest. Finally, we calculate the weighted average recommendation score. In order to identify users who behave similarly to our user of interest, we have three steps to go through: 1) aggregating the data from our random user and other users, 2) creating a correlation matrix, and 3) finding the most similar users (Top Users).

After identifying user similarity, we encounter a problem; Now, there are two levels in the data, namely correlation and rating. Some users will have high correlation with our random user of interest, but low rating, etc. So, which one of these levels is it most important to consider? In order to figure that out, we make a weighting based on the two levels (we calculate the weighted average recommendation score). Now, we create a single score by considering both correlation (i.e. the impact of the users that behave similarly to our user of interest) and rating simultaneously. This results in a total of 8071 recommended movies, so we choose to recommend just the ones that have a weighted score above 3.5

2.4. Model-Based Collaborative Filtering Recommendation System

We can use collaborative filtering to recommend items that a given user might like based on the ratings of other similar users. In our case, this method works by searching the whole dataset for users

who gave similar ratings to the same movies as our given user. When we have found the items that these similar users liked, then we can create a list of movies that our user of interest will probably like (Abhinav, n.d.). This can be done in different ways, but we do need data that contains a set of users and a set of items (movies, in this case). The users' reactions can be explicit (e.g. ratings on a scale) or implicit (like time spent on the item, viewing, etc.). Our data has explicit ratings on a scale from 1-5. We set it up as a matrix of users and items. Some of the data will almost always be missing from the matrix, because not every user rated every movie. However, the missing data can be predicted and filled out if we train a machine learning algorithm on the data we do have (Abhinav, n.d.).

Three important questions that need answering now are: How do we determine which users or items are similar? Once that is done, how do we determine the rating that a user would give based on similar users (the weights)? And finally, which measure do we use to evaluate the accuracy of our predicted ratings? With regards to the question of evaluation measure, one possible measure of model accuracy is the root mean squared error (RMSE), which indicates the difference between a known value in the dataset and a predicted value. Another measure is the mean absolute error (MAE) which indicates the magnitude of error by taking the mean of the absolute value of all error values. As to the other questions, we will be using matrix factorization for our model-based recommendation system.

2.4.1. Matrix Factorization

An assumption of matrix factorization is that there exist some latent factors when a user likes a movie. The same type of hidden features are also present for the movies. Latent factors are a type of features in machine learning. In this case, they will be the underlying reasons as to why a given user likes a given movie, e.g. the genre of the movie, the director, the duration, the production company, the language in which the film was shot, etc.

Firstly, in order to fill in the gaps in our data, we find the weights of the latent factors for users and movies over our data. We can then use these weights to make predictions for non-existent observations. The approach here is to decompose the user-item matrix (which we made for the previous systems) into 2 separate matrices of fewer dimensions. It is assumed that the transition between these different types of matrices occurs with latent factors, and we can thus assume the latent variables. Now, that we have filled in (assumed) missing observations, we can find the weights of the latent factors.

So, to summarize, we assume that our rating matrix is the dot product of two factor matrices – one separate matrix for user latent factors and one separate matrix for movie latent factors. Both users and movies are then considered to have weights (scores) for these latent features. We begin by finding weights on the data that we have, and then we fill out the gaps according to these weights.

2.4.2. The Singular Value Decomposition (SVD) Algorithm

For our model-based recommender we use the Surprise library in Python (Hug, 2020), which was build specifically for the purpose of analyzing recommender systems that are dealing with explicit rating data. Amongst other things, it provides a number of various ready-to-use prediction algorithms. One of these is the probabilistic matrix factorization-based SVD model. The prediction of the model $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \, p_u$$

Where if the user *u* is unknown, then the bias term b_u and the factors p_u are assumed to be zero. The same is the case for *i* with b_i and q_i. In order to estimate the unknown variables (b_u, b_i, p_u, and q_i) we want to minimize this regularized squared error:

$$\sum_{r_{ui} \in R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

We do this by a stochastic gradient descent as follows:

$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$$
$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$$
$$p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u)$$
$$q_i \leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i)$$

In this, $e_{ui} = r_{ui} - \hat{r}_{ui}$. This process is performed and repeated over all of the weights (ratings) of the training data set for a specified number of epochs. γ is the learning rate and has a default setting of 0.005, and λ is the regularization term and has the default setting of 0.02 (Hug, 2015).

2.4.3. Hyperparameter Tuning:

Now, that we know how to estimate the weights of the null observations, we need to figure out how the long process of changing the values should be. Should we update the weights 10 times? 100 times? How does this affect the learning speed of out model? Our model will have two interactive hyperparameters that need tuning: the number of epochs over which we train our model, and the learning rate of the model.

For the tuning of these hyperparameters, we first define a grid with the different hyperparameter combinations that we are wanting to try out. We set a combination 3 epoch values and 3 learning rate values. This makes for 9 possible combinations. We use the Surprise function GridSearchCV, where we set the cross-validation parameter (cv) to 3 to cross validate over 3 folds. This means that we will run the model 27 times in total.

3. Results

Since this is not a model-based machine learning method, we cannot directly evaluate the accuracy of our recommendation system. However, we can use the ratings data to check if the users who rated the input movie highly did indeed also rate the recommended movies highly. Figure 1 shows The relationship between the ranking of the recommended movies (lower numbers mean top recommendations) and rating.

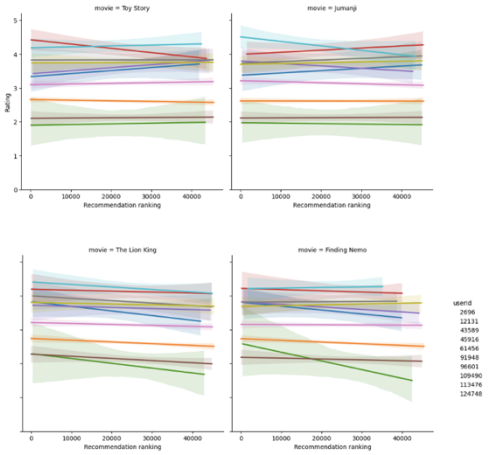

Figure 1: *The relationship between recommendation ranking and movie rating when using the content-based recommender. A lower recommendation value means that it is highly recommended (number 1 recommended movie, and so on).*

On the plots in figure 2 for the item-based recommender we see results that generally resemble those of the content-based recommender, except for one movie (Fight Club (1999)) where there does seem to be a tendency towards a negative relationship between recommendation ranking and rating.

After tuning the hyperparameters of the model-based recommender, we found that running it with a learning rate of 0.01 over 15 epochs resulted in the most accurate predictions. If we look at an example, say Blade Runner, we see that the model predicts that the user will give a rating of 4.21 – a movie which the user in reality rated 4.0 (since the actual ratings are integers), so that is pretty good. If we then look at a movie which the user did not rate, Cries and Whispers, the model predicts that the user will give a rating of 4.04. These predictions are made with a root mean squared error of 0.9 and a mean average error of 0.7.

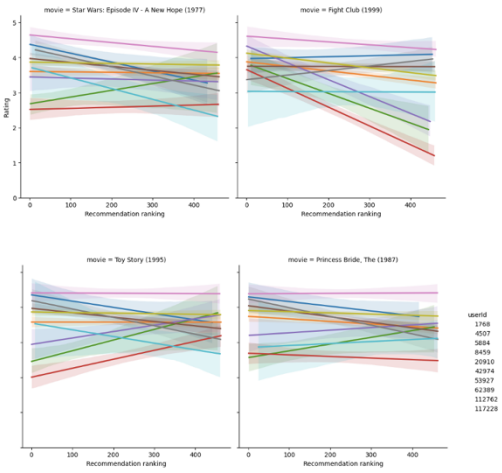

Figure 2: *The relationship between recommendation ranking and movie rating when using the item-based recommender.*

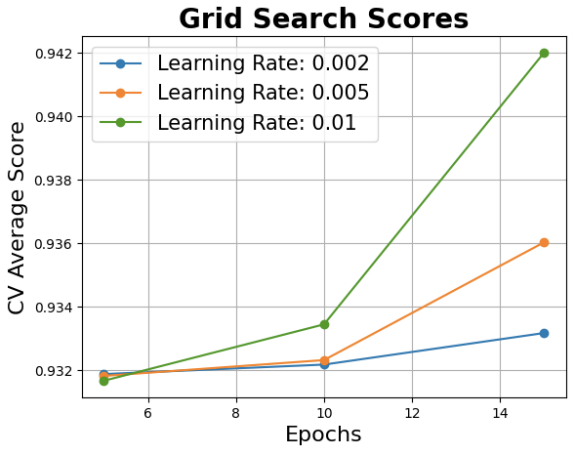

Figure 3: *Results of the grid search for the hyperparameter tuning.*

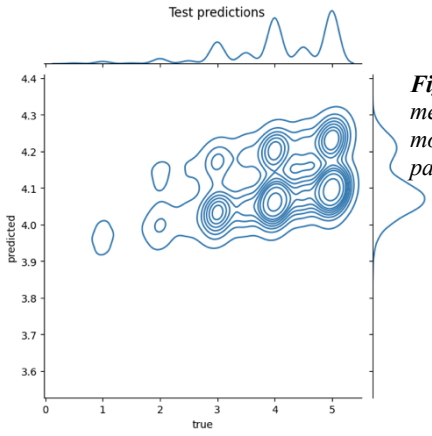

Figure 4: *Accuracy measures for the SVD model with the best parameters*

RMSE	MSE
0.93	0.72

Table 1: *Accuracy measures for the SVD model with the best parameters*

4. Discussion

It is difficult to make definitive conclusions about which movie recommendation system is superior based on the results of this project, since the evaluation measures of the four systems were different. If this project were to be continued, it would be of interest to obtain similar quantitative evaluation metrics from each of the four model types. This is not only because it would allow us to quantitatively compare the systems, but also because the error metrics, which we have for the model-based recommender system, are mostly meaningful to discuss when compared to the values for other models.

However, there were still noticeable differences between the outputs of the different recommender systems. Upon qualitative assessment of the content-based recommender, we can confirm that the system is indeed recommending movies that share the same overarching topics, and would thus be described using similar key terms. For instance, if we input the movie "The Lion King", recommended movies include "How the Lion Cub and the Turtle Sang a Song", "The Lion King 2: Simba's Pride", "African Cats", etc. These are all movies about lions, however, now the question becomes whether this is actually a useful way of recommending movies. The recommended items include movies of various genres – all from animated movies to documentaries. If a user liked "The Lion King" it seems unlikely, that they would also, on that basis, like a documentary about lions. This is a clear limitation of the content-based recommender, and as we also see from the plots in figure 1, high ranking on the recommendation list produced by this system does not seem to be predictive of higher user ratings.

Compare this to the same plot for the item-based collaborative filtering recommender, and we seem to have slightly more of a tendency towards a negative relationship. These plots are however only to provide a general idea of what the recommendations look like for randomly chosen movies, and are to be taken with a grain of salt. However, if we qualitatively assess the output of this system and have it recommend movies from the input "Fight Club", recommendations include movies such as Pulp Fiction, Twelve Monkeys, Snatch, and Memento. These are arguably movies that one with just a little knowledge about movies could imagine recommending to a film-naive friend who liked Fight Club. They are of a more streamlined sort (with regards to genre, etc.) that the movies recommended by the content-based recommender. Ideally we would have had a similar plot for the user-based recommender, but due to technical issues and a lack of time we do not.

With regards to the model-based approach, when we apply this method for all the full data set, then we can fill out all of the missing values where users haven't rated movies with rating estimates predicted by our model. When we have the full dataset, we can then recommend the top movies for a specific user. We see from the plot in figure 4 and from the examples that we ran, that the predictions are looking quite good. Figure 4 shows that the predicted values tend to lie close to the true values for the ratings, which indicates that the model is performing well.

5. Conclusion

We have now seen how we can use different approaches to provide movie recommendations. We created four different recommender systems: a recommendation system with a content-based filtering approach, and three systems with a collaborative filtering approach (and item-based, a user-based, and a model-based system). We found that the purely content-based approach is arguably the weakest one, due to its simple nature of recommending movies based on superficial item features. The stronger results are found, when we combine features of the items with knowledge from the data about user behaviour. The path to a good recommendation system lies in continuously observing and learning real human behaviour, which is why a machine learning-based approach is ideal for this task.

6. Literature

Abhinav, A. (n.d.). *Build a Recommendation Engine With Collaborative Filtering*. Real Python. Retrieved 25 May 2023, from https://realpython.com/build-recommendation-engine-collaborative-filtering/

Google for Developers. (2022). *Advanced Machine Learning Course in Recommendation Systems—Collaborative Filtering and Matrix Factorization*. https://developers.google.com/machine-learning/recommendation/collaborative/basics

Hug, N. (2015). *Welcome to Surprise' Documentation*. https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVD

Hug, N. (2020). Surprise: A Python library for recommender systems. *Journal of Open Source Software*, 2174.

Maxwell Harper, F., & Konstan, J. A. (2015). *The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages.* http://dx.doi.org/10.1145/2827872

Mngomezulu, M., & Ajoodha, R. (2022). A Content-Based Collaborative Filtering Movie Recommendation System using Keywords Extractions. *2022 International Conference on Engineering and Emerging Technologies (ICEET)*, 1–6. https://doi.org/10.1109/ICEET56468.2022.10007345

Pedregosa, F., Varoquaux, Ga"el, Gramford, A., Michel, Thirion, B., Grisel, O., & et al. (2011). Scikit-learn: Machine Learning in Python. *JMLR 12*, 2825–2830.

The Upwork Team. (2021). *What Content-Based Filtering is & Why You Should Use it*. https://www.upwork.com/resources/what-is-content-based-filtering