# Recommendation System for MovieLens20M

**(Final Report Group 9)**
**Web Mining Project FSS 2022**

Kathrin Lara Schmidt (`kathrins@mail.uni-mannheim.de`),
Johanna Reuschel (`jreusche@mail.uni-mannheim.de`),
Subash Poudel (`spoudel@mail.uni-mannheim.de`), and
Jorida Jolla (`jjolla@mail.uni-mannheim.de`)

University of Mannheim, 68131 Mannheim, Germany

**Keywords:** Movie Recommendation System · Collaborative Filtering · Content-based Recommender · Hybrid Recommender · MovieLens20M

## 1 Introduction

In times of online ordering, streaming services for music and movies, and social media platforms, recommendations for new products, similar movies, and interesting profiles are essential to save users the time and effort of sifting through the mass of options. Who hasn't had the case of wanting to watch a movie on the couch while eating, and by the time the huge Netflix offer has been clicked through, the food is already cold? A possible alternative to make a decision more quickly would be to trust that Netflix knows the tastes of its users and to select a film from the "Recommended for you" section. Good and functioning recommendation systems have become part of everyday life and may be particularly crucial in the case of streaming services. Unlike ordering a needed product on Amazon or following a well-known person on Instagram, the end user usually does not yet have a concrete movie wish in mind and is more dependent on a good recommendation. Therefore, this project will focus on the problem of a recommendation system for movies as part of the category of web usage mining. The research question is to find out if it is possible to achieve better results than existing movie recommendation systems by implementing a hybrid approach of collaborative and content-based filtering. The measurement of the results is related to the accuracy of the predictions of ratings from 0 to 5, which reflect how much a certain movie will be liked by a certain user.

## 2 Experimental setting

### 2.1 Data Analysis

The recommender system is fundamentally based on the MovieLens20M dataset [2]. This consists, as the name suggests, of 20 million ratings performed by 38,000 users on 27,000 movies. In addition, a total of 465,000 tags were placed on the content of the individual movies [1]. The GroupLens Research Project has developed and made available various versions of the MovieLens dataset, but the largest (20 million ratings) seems to make the most sense for this recommendation system, since the larger the datasets and thus ultimately the larger the test set, the more precise and accurate the algorithms can be trained. In general, MovieLens datasets are one of the most used datasets to implement movie recommender systems. This means that various results and their evaluations are already available, which makes it easier to compare the results of this hybrid approach at the end. The advantage of this dataset over other movie ratings datasets is that it has the highest density of ratings. For example, if you have a user with only one rating for a movie, it is difficult for the recommender system to make a suitable prediction (due to the cold start problem). The movieLens dataset will only include users if the user has rated at least 20 different movies. It consists of the following seven files:

- *movies.csv*: This file contains information related to one movie in the following format: movieId, title, genres.
- *ratings.csv*: The rating file shows the ratings of one user according to one movie. The format is: userId, movieId, rating, timestamp.
- *tags.csv*: Represents the single tags that an user has given to one movie.
- *links.csv*: This data can be used to link the dataset with the so called IMDb dataset or the Movie Db dataset to add additional features to the MovieLens data.
- *genome-scores.csv*: The file shows how relevant the given movie-tags are.
- *genome-tags.csv*: Provides the tag descriptions for the tag IDs.

One disadvantage of the MovieLens dataset is that it does not contain very much information beyond the actual ratings. Therefore, only few features in content-based filtering can be implemented for this dataset. For this reason we supplemented the dataset with the IMDb [3] dataset using the *link.csv* file. The IMDb dataset is a collection of different subsets in TSV format. Following subsets are contained: *title.akas.tvs, title.basics.tsv, title.crew.tsv, title.episode.tsv, title.principals.tsv, title.ratings.tsv, name.basics.tsv.* In this project, only the files *title.crew.tsv* and *title.principals.tsv* were used to implement additional content-based features. The file *title.crew.tsv* contains information about the directors and writers of a movie with its identifier tconst. The *title.principals.tsv* file, on the other hand, provides details about the crew and the cast of the single movies.

To better understand how the individual data points of the movieLens dataset are distributed, it is useful to first look at the visual representation of the data. The first figure shows the distribution of the individual ratings from 0 to 5. The left-hand side shows the entire dataset with 20 million entries. On the right-hand side is a much smaller dataset with around 7.5 million entries. How this dataset was created and why it is relevant to the project is explained in the preprocessing chapter. As can be seen from the graph, apart from the total amount, the distribution of ratings is very similar across the entire datasets. Most movies were rated with 3 to 4 points while only relatively few movies were rated with points between 0 and 1.

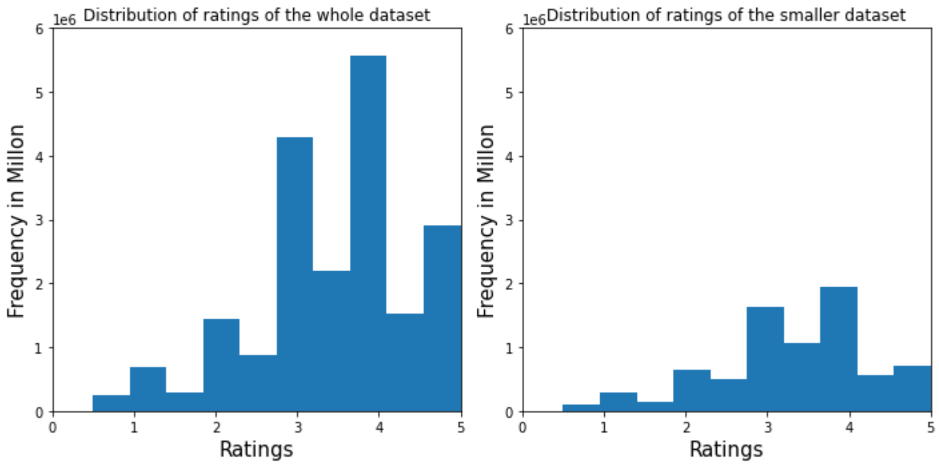

**Fig. 1.** Distribution of ratings on the scale of 0 to 5

The second figure shows the distribution across all users, how many movies they have rated in total and thus how many entries per user occur in the dataset. Again, the distribution over the entire 20 million dataset is shown on the left, and the smaller dataset itself is shown on the right. This time, it should be noted that, in contrast to figure 1, the y-axis has a different interval, as the data could not otherwise be presented clearly. Now, the two graphs differ more clearly from each other, as the large dataset represents that over 120,000 users and thus the majority have rated 0 to 500 movies, while in the smaller dataset there is no user who has not rated at least 250 movies. Otherwise, apart from the fact that the total number of users significantly differs, the distribution of the graphs is similar.

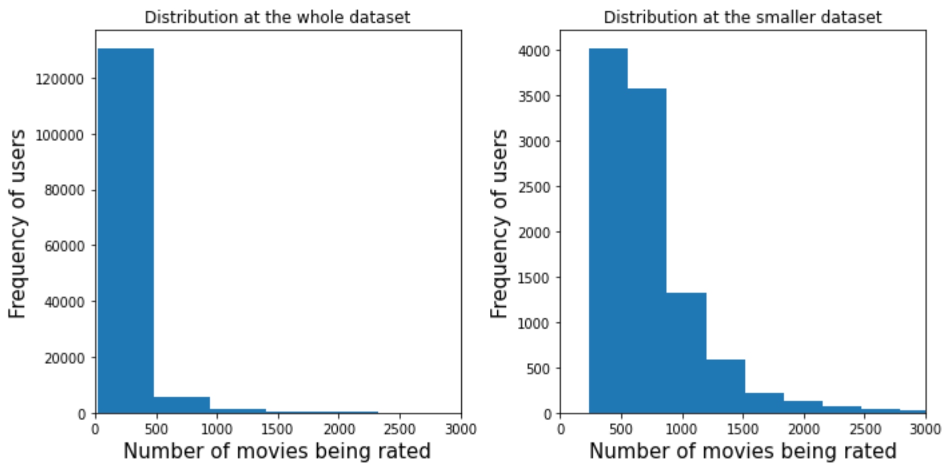

**Fig. 2.** Distribution of how many users have how often rated

### 2.2 Preprocessing

Before performing any calculations regarding a recommendation system we started with some preprocessing steps. Since the MovieLens dataset was already pretty complete not many classical preprocessing steps had to be applied, e.g. there weren't any rows with missing values or duplicated that had to be removed.

In order to build a Content-based Recommendation System that incorporates the release year and the genre(s) of a movie the data contained in the *movies.csv* file needed some reformatting. First, the release year was split from the movie title and got moved to its own column. Then all movies were assigned to one of 5 different year categories based on their year of release, which was again added as a new column:

- Category 0: before 1960
- Category 1: 1960 - 1979
- Category 2: 1980 - 1999
- Category 3: 2000 - 2019
- Category 4: after 2010

Since the majority of all movies has more then just one genre listed in their respective column, we split up the genre list and introduced a new column for every of the 19 existing genres. If a movie corresponds to a genre the respective column was filled with a 1, and with a 0 if not.

| movieId | title | year | year_category | Adventure | Animation | Children | ... | FilmNoir |
|---|---|---|---|---|---|---|---|---|
| 1 | Toy Story | 1995 | 2 | 1 | 1 | 1 | ... | 0 |
| 2 | Jumanji | 1995 | 2 | 1 | 0 | 1 | ... | 0 |

As mentioned earlier, the original data set within this project was limited from over 20 million rating data points to 7581837 data points. The reason for this is that primarily the collaborative filtering with the normal RAM capacity limits of 13 GB cannot be calculated with the amount of users and movies. Since we didn't want to use one of the smaller movieLens datasets, which have fewer users and movies, but also fewer ratings per user and movie, we reduced the 20M movieLens dataset as follows. First, we sorted the users by their rating count and then kept only the 10000 that gave the most ratings. The same process was then repeated for the movies, leaving 10000 movies in the end. The resulting dataset was used throughout the rest of the project.

As a last preprocessing step we split our ratings dataset into a trainset and a testset. Therefore we used the "Leave One Last Basket" method, where we randomly chose 25% of the ratings from each user to put them into the testset. We chose this method so that every user would be present in the trainset as well as in the testset, because content-based recommendation would not work for users that only exist in the testset but not in the trainset. That is because it predicts a rating based on the similarity to already seen and rated movies of a certain user. If a user would not be in the trainset that would mean that he had not watched and rated any movies yet, so no similarities can be computed and therefore also no predicted rating. After executing the split the resulting train- and testset we exported to csv-files so they can be easily reloaded without having to run the very time-consuming splitting algorithm again.

## 3 Methodology

The concept of the recommendation system developed in this project is based on a hybrid approach. First, collaborative filtering was implemented (both memory- and model-based), then a content-based model was developed (with three different features). Finally the results of these two methods were combined. The reason why a hybrid approach was chosen is that collaborative filtering is usually selected for recommendation systems for movies, but content-based filtering is also possible considering the selectable features. Since we wanted to deviate from the classic concept of a movie recommendation system and compare our results with existing solutions, we tried to combine the two approaches. In the following, the three individual approaches are explained in more detail.

### 3.1 Baseline

As a baseline we used the average rating of a movie by all users, which resulted in the following performance values:

**Table 1.** Performance of Baseline Recommender on Testset

| MAE | RMSE | NDCG@10 | NDCG@100 |
|---|---|---|---|
| 0.7075 | 0.9082 | 0.8758 | 0.8758 |

## 3.2 Collaborative Filtering

**Memory-based Collaborative Filtering**
In order to be able to compare which delivers the better results, both approaches of memory-based collaborative filtering (user- and item-based) were implemented. After creating a user-movie matrix, a similarity matrix was created for users as well as movies. Then, using the given formulas, a prediction of the ratings was calculated based on the previous ratings of either the corresponding user or the corresponding movie.

**Model-based Collaborative Filtering**
For the implementation of model-based collaborative filtering, different algorithms were tested using a 5 fold cross validation to find out which one performs best under the given circumstances. The results are shown in table 2, where the different algorithms are listed together with their RMS and MAE values. The hyperparameters of the different algorithms were selected in such a way that default values were chosen first. Then, based on the results, it was decided whether the hyperparameters should be corrected upwards or downwards, or whether the best possible result for the algorithm had already been achieved. As you can see, the SVD n_factors equal 50 provides the best results as the corresponding values are the lowest. Therefore, this model was subsequently used to train on the whole training set and evaluate the final result on the actual test set. At the end, a table with the predictions was created.

**Table 2.** Results of the performing of different algorithms for model-based collaborative filtering.

| Algorithm | RMS | MAE | Fit Time | Test Time |
|---|---|---|---|---|
| KNNBasic(k=10, sim=cosine, item-based) | 0.943522 | 0.716971 | 363.1378 | 788.221979 |
| KNNBasic(k=20, sim=cosine, item-based) | 0.919404 | 0.697669 | 327.656371 | 758.252104 |
| KNNBasic(k=50, sim=cosine, item-based) | 0.906579 | 0.686921 | 556.153844 | 747.974485 |
| KNNBasic(k=20, sim=pearson, user-based) | 0.881193 | 0.684907 | 1455.36913 | 2731.981014 |
| KNNWithMeans(k=20, sim=cosine, item-based) | 0.813355 | 0.618325 | 556.444174 | 663.35883 |
| SVD(n_factors=30) | 0.746386 | 0.568832 | 183.165293 | 28.160773 |
| SVD(n_factors=50) | 0.745352 | 0.567891 | 218.118131 | 26.317425 |
| SVD(n_factors=100) | 0.745401 | 0.567783 | 333.150664 | 29.760682 |
| SVD(n_factors=150) | 0.74629 | 0.568548 | 449.504294 | 29.142676 |
| NMF(n_factors=15) | 0.825956 | 0.634066 | 411.954342 | 15.592871 |
| NMF(n_factors=30) | 0.868169 | 0.656528 | 529.678849 | 14.722431 |

## 3.3 Content-based Filtering

For building our Content-based Recommender, we wanted to include five different features to compute the similarity between the movies. Therefore the movie genre, the release year and the tag relevance scores, that were provided by the

MovieLens dataset, and the director and the main actors of a movie, that were provided by the Imdb dataset, we used.

**Movie Genre**
As mentioned above, the genre column was split up so that there is now a separate column for each type of genre, filled with 1 if the film corresponds to this genre and 0 if it does not. Based on the resulting "vectors" a similarity matrix was computed, that stores the pairwise cosine similarity between every two movies. For predicting a rating for an unknown user-movie-pair(consisting of userId and movieId), first the one row matching the given movieId was filtered out of the matrix, to get the similarity scores between the specific movie and all other movies. Then all movies were filtered out that the specified user has not seen and rated until now. Finally the predicted rating was determined by calculating the weighted sum of the ratings of the top n most similar movies.

In order to determine the optimal number of nearest neighbours to use for calculating the final prediction, we run our recommender on a smaller sample of the trainset with different settings. It turned out the best results were archived with *top_n = 7* .

**Table 3.** MAE and RMSE for CB Recommender with Movie Genres

| *top_n* | MAE | RMSE |
|---|---|---|
| 2 | 0.8460 | 1.0920 |
| 3 | 0.8071 | 1.0344 |
| 4 | 0.7867 | 1.0045 |
| 5 | 0.7742 | 0.9862 |
| 6 | 0.6806 | 0.8704 |
| 7 | 0.6788 | 0.8685 |

**Release Year**
To enable content-based recommending based on the release year we introduced five different year categories, each covering a specific period of time. The exact classification has already been discussed in the preprocessing section. As a first step all already seen and rated movies of a specified user were selected from the trainset and the column containing the information about the year category was added from the movie dataset. The predicted rating for a specific movie was calculated as the average of all ratings of movies already seen with the same year category as the given film.

**Tag Relevance Scores (Content Summary)**
For calculating the movie similarity based on the tag relevance scores we took a pretty similar approach as with the movie genre. By turning the original table from the *genome_scores.csv* into a pivot table with the movieId's as indices, we ended up with "relevance vectors" for every movie. Based on those again a

cosine similarity matrix was computed just like with the movie genre vectors. The following steps and the final prediction of the rating are identical to the procedure described in the movie genre section.

We performed the same calculations for determining the optimal number of nearest neighbours for the final prediction, as we already did with the movie genres. It turned out the best results were archived with *top_ n = 2* .

**Table 4.** MAE and RMSE for CB Recommender with Tag Relevance Scores

| *top_ n* | MAE | RMSE |
|---|---|---|
| 2 | 0.3777 | 0.5167 |
| 3 | 0.4660 | 0.6119 |
| 4 | 0.5030 | 0.6549 |
| 5 | 0.5261 | 0.6821 |
| 6 | 0.5419 | 0.7012 |
| 7 | 0.5513 | 0.7123 |

**Director and Actors**

In order to get a more diversified Content-based Recommender we wanted to incorporate even more features, which were gathered from the Imdb dataset. The MovieLens dataset contains a file named *links.csv*, that matches the MovieLens-ID (listed as “movieId”) of a movie to the corresponding Imdb-ID (listed as “imdbId”). First some quick reformatting of the Movie-ID (listed as “tconst”) from the Imdb dataset had to be performed, so it would match the naming scheme from the *links.csv* file. Based on the Imdb-ID we were then able to perform a left-join to merge the MovieLens-IDs with the directors- and actors columns from their respective dataframes and removed all entries with missing or empty values that resulted from the join. Then all already seen and rated movies of a specified user were selected and compared the a specified movie to check if there are any matches in the list of directors or actors. The exact number of matches was stored in a new column called “similarity”. It was planed to compute the predicted rating as a weighted sum of the ratings of movies where matches were counted, but after some initial testing rounds and a more detailed data analysis, we realized that there are hardly any matches to find at all. This is because there are only a handful of movies that have the same director or overlapping actors, and the chance is even lower that a user chosen from the dataset has rated two or more movies that have those kind of matches, since every user has only rated of small fraction of the overall amount of movies in the dataset.

This is why we finally decided to not incorporate Content-based Recommendation based on directors and/or actors in the final version of our Hybrid Recommender, even though the main functionalities were already implemented.

### 3.4 Hybrid Recommendation

Our initial idea of our Hybrid Recommendation was to combines the following six different recommenders:

- Item-based Memory-based Collaborative Filtering
- User-based Memory-based Collaborative Filtering
- Model-based Collaborative Filtering with SVD(n=50)
- Contend-based Recommendation with Movie Genres ($top\_n = 7$)
- Contend-based Recommendation with Release Years
- Contend-based Recommendation with Tag Relevance Scores ($top\_n = 2$)

The final rating of a movie from a certain user results from calculating the average value from the predictions of the six recommenders. We started with incorporating all six recommenders into our hybrid version and then tested some versions that incorporated a smaller selection of recommenders.

## 4 Evaluation and Discussion of the Results

### 4.1 Evaluation Setup

The MovieLens dataset includes only numerical ratings that range on a scale between 0 and 5. Therefore we used classical numerical evaluation measures such as Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) to compare the performance of our different recommenders, to optimize certain parameters and to get an impression of the overall performance of our hybrid solution. The lower both values are, the better the solution.
In order to benchmark our solution we also used another typical evaluation measures for ranked results: The Normalized Discounted Cumulative Gain (NDCG@K). This metric is very commonly used when comparing the top ranking papers about recommendation systems for the MovieLens20M dataset on PapersWithCode [4]. NDCG@K ranges between 0 and 1, but in contrast to MAE and RMSE here the latter one is the optimal solution.

### 4.2 Results & Discussion

Short Disclaimer/Explanation:
Since the Contend-based Recommendation had a very high run time because we weren't able to parallelise the prediction of the ratings in google colab (even tho the calculation of the different ratings is independent from one another). So we were only able to test the performance of our Hybrid Recommender on a small sample of the testset, which contained only 40.000 ratings. Furthermore, colab often has fluctuating speeds or execution times as far as the runtime of the code is concerned. This has made the execution of RAM and time-intensive code even more difficult.

Table 5. Performance of Different Recommender Systems on the entire Testset

| Combination | MAE | RMSE | NDCG@10 | NDCG@100 |
|---|---|---|---|---|
| Only user-based collaborative filtering | 2.5325 | 2.7447 | 0.9096 | 0.9096 |
| Only item-based collaborative filtering | 2.5175 | 2.7271 | 0.9511 | 0.9337 |
| Only model-based collaborative filtering | 0.5614 | 0.7377 | 0.9779 | 0.9779 |

Table 6. Performance of Different Recommender Systems on the first 40,000 Entries of the Testset

| Combination | MAE | RMSE | NDCG@10 | NDCG@100 |
|---|---|---|---|---|
| Only user-based collaborative filtering | 2.4509 | 2.6675 | 0.8866 | 0.8714 |
| Only item-based collaborative filtering | 2.4373 | 2.6469 | 0.8929 | 0.8815 |
| Only model-based collaborative filtering | 1.037 | 1.3095 | 0.63 | 0.6576 |
| Only feature "Genre" of content-based filtering | 0.757 | 0.9695 | 0.8474 | 0.8304 |
| Only feature "Year" of content-based filtering | 0.757 | 0.952 | 0.9435 | 0.8677 |
| Only feature "Tags" of content-based filtering | 0.7077 | 0.9387 | 0.8984 | 0.8759 |
| **All three features of content-based filtering and all three approaches of collaborative filtering** | 1.0188 | 1.1988 | 0.9810 | 0.9207 |
| **All three features of content-based filtering and only model-based collaborative filtering** | 0.7176 | 0.9042 | 0.8988 | 0.8698 |
| **All three features of content-based filtering and no other** | 0.6783 | 0.8659 | 0.8701 | 0.8823 |

In order to finally analyze and evaluate the results, they have been summarized in tables 5 and 6. As can be seen in Table 6, the four evaluation metrics were applied to various compositions of collaborative and content-based filtering and thus provide the possibility of comparison between the different approaches. In order to first evaluate the individual filtering methods and their results, the methods are listed individually at the beginning of the table. As can be seen from the values, the two memory-based approaches (user- and item-based) of collaborative filtering have by far the highest MAE and RMSE values (each greater than 2) and thus represent the worst approaches, with the ratings predictions deviating the most from the actual ratings. The user-based approach performs slightly worse than the item-based approach. Immediately after memory-based collaborative filtering, model-based collaborative filtering follows with the next lowest value, with ratings metrics still just above 1. This is not a particularly good result either. If we then compare the three content-based features with each other, we can see that with an MAE value of around 0.75 and an RMSE value of around 0.95, they are now below 1 for the first time and thus perform significantly better than the collaborative filtering methods. The accuracy of the predictions of the three features is close to each other, whereas the feature that considers the "tags" stands out positively from the other two. Since the results considered so far are based only on the first 40,000 entries of the test set, it makes sense to look at the results for the whole test set, at least for the collaborative filtering methods. Since collaborative filtering is the most commonly used approach for movie recommendation systems, one would expect better results than with content-based filtering. Looking at the results in table 5, it can be seen that the memory-based methods still perform very poorly and even slightly worse, while the model-based method is much better. With an MAE of just 0.5614 and an RMSE of 0.7377, it even outperforms the content-based methods, thus confirming the aforementioned assumption. By comparing the results of table 5 and table 6 (especially with respect to the model-based collaborative filtering) the assumption arises that the models for the first part of the test set perform worse, which has been confirmed for us in the course of the development of the implementations. This comes from the fact that the content-based feature have already achieved better results on samples of the test set than on the first 40,000 entries.

The initial setup of our hybrid recommender showed rather mediocre results, due to the bad performance values of both memory-based collaborative filtering methods. Therefore we tested the performance of different recommender combinations, which improved the overall performance, when only considering MAE and RMSE. By removing both memory-based collaborative filtering methods the results improved a lot (-0.3012 (MAE) & -0.2946 (RMSE)). Having a hybrid recommendation system based only on the three contend-based recommendation methods achieved even better results (-0.3405 (MAE) & -0.3329 (RMSE)), which is kind of logical when looking at the performance values of every recommendation method individually. Those three methods also have the best individual performance.

When looking at the NDCG@10 and NDCG@100 values it can be seen that for all test round the results were pretty close no matter what recommender or combination nor dataset was used. They all range between 0.8304 and 0.9810, except from one outlier, which was the model-based collaborative filtering method on the entire testset, which simultaneous has the best performance for MAE and RMSE. These two observations and the fact that our results are suspiciously close to the optimal value lead us to the conclusion that there must have been an error in the calculation of this measured value or some kind of bias.

This suspicion becomes even stronger when you compare our results with the top-performing papers on paperswithcode. There the paper that scored best in the NDCG@100 category achieved a score of 0.448 and the best paper in the NDCG@10 category achieved a score of 0.6404, which are booth significantly lower then all our results. Since the quality of the papers listed on paperswithcode is quite high and they all use very advanced methods, it is rather unlikely that we actually achieved that much better results with our rather simple setup. So a possible next step for further improvement would be to exactly analyse and verify those results and do a recalculation if the suspicion should be confirmed, in order to do a more extensive and meaningful benchmarking.

On paperswithcode there was also one paper in the top 15 that listed an RMSE value, that we can use for comparison. The method proposed by this paper achived an RMSE of 0.7887, which is definitely better then all our hybrid approaches, but actually not to far away of our best method. With just the model-based collaborative filtering method on the entire testset we were actually able to score better then this paper.

## 5 Conclusion and Future Direction

Finally, to summarize the new insights gained from this project, it can be said that a hybrid recommender for movies is a pretty promising field for future research and improvement. That is because we could generate quite good results even with a rather simple approach. The next useful steps would be those of hyperparameter optimization regarding the hybrid recommender as such. This would primarily involve the weights of the individual content-based and collaborative parts to find an optimal distribution of them. In addition, it would be useful to train the individual methods with the entire 20 million entry data set, which would require resources beyond the scope of this project. Also in the area of content-based filtering it is probably worthwhile to further develop the implementations, as this has yielded surprisingly good results. Furthermore, it involves considers the end-user and his preferences in more detail than just by a single value (the actual rating). The reason why it would make sense to further expand hybrid and/or content-based recommenders specifically for movies is that streaming services like Netflix will continue to play a major role in the future. A good recommendation system could set them apart from competing streaming services and therefore get them more profit.

## References

1. https://grouplens.org/datasets/movielens/
2. https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset
3. https://www.imdb.com/interfaces/
4. https://paperswithcode.com/sota/collaborative-filtering-on-movielens-20m