Project 1: recommender system

Decision support systems

by group 9:

Daniel Christopher Biørrith, 201909298

Jeppe Stjernholm Schildt, 201905520

Aarhus University

Department of Electrical and Computer Engineering

August 21, 2024

Contents

1 Introduction

In a rapidly evolving online and connected world, individuals are constantly presented with an overwhelming number of choices. Which movie to watch? Which item to buy? Which book to read? The significance of filtering out irrelevant information and presenting recommendations tailored to each specific user has grown immensely. As a result, recommender systems have become an integral part of our daily online lives, enabling individuals to navigate the sea of possible choices. By harnessing the power of data and advanced algorithms, these systems help enhance user experiences and transform industries, such as the movie industry.

In this paper, we aim to explore, implement, and test a recommender system for recommending movies to users. We begin by introducing the general theory and methods behind recommender systems, followed by an introduction and analysis of the dataset used. Subsequently, we implement and test the recommender system. Lastly, we discuss the results and potential improvements and finally provide a conclusion for the project.

2 Methods

To achieve the task of suggesting relevant items to users, there are two main paradigms of methods; the *collaborative filtering* method and the *content-based* method.

2.1 Collaborative filtering

Collaborative filtering methods are based solely on past interactions between users and items, in order to produce new recommendations. This would be stored in a user-item matrix, such as the one shown in figure 1, in which there are five users and five items.

As evidenced by the matrix, not every entry is filled, since users generally don't rate all items. The primary goal of collaborative filtering is to predict the missing ratings based on the non-empty entries. The items with the highest predicted scores will then be recommended to the given user.

Collaborative filtering can be divided into two subcategories: model-based and memory-based methods.

	u_1	u_2	u_3	u_4	u_5
i_1	1		3	2	
i_2	4	2	5		3
i_3			2	4	4
i_4		5		2	1
i_5	5	4			4

Figure 1: An example of a user-item matrix with 5 users and 5 items

Generally, memory-based methods work directly with user-item interactions and primarily rely on nearest neighbour searches to find similarities between users or items (known as user-based and item-based approaches, respectively). A user-based approach involves identifying the K-nearest neighbours among users and recommending items based on the most popular items within each user's neighbours. Conversly, an item-based approach identifies neighboring items by determining which items users have interacted with in similar ways. User-based approaches tend to be more personalised, as they recommend items likely to align with users' specific preferences. However, they are sensitive to the limited number of recorded interactions, resulting in high variance but low bias. In contrast, item-based approaches exhibit low variance and high bias, as items typically have many interactions but from a diverse range of users, making the recommendations less personalised. A big flaw in memory-based models specifically is that their complexity is $O(ndk)$, where n is the number of users, d is the number of items and k is the number of neighbours. This means the system does not scale well with increasingly large datasets.

Model-based methods, on the other hand, utilize the existing ratings to estimate a model that generates predictions, based on mathematically extracting a latent representation of the users and items. This approach often involves techniques like matrix factorization, clustering, or deep learning algorithms. Model-based methods aim to generalize patterns in the data.

Collaborative filtering has the flaw that it suffers from a cold start. As it relies on past ratings, it cannot make personalised recommendations to a user with no prior ratings, nor can it recommend an item that has no ratings. To counteract this, new users can be shown popular items and new items can be randomly distributed across users until it gets enough ratings.

2.2 Content based methods

Where collaborative methods only look at the user-item interaction, content based methods utilize additional information about the user, such as their sex, age, or job, or item features such as release year or genre for movies. The goal is to build a model based on these features and recommend items by looking at similar users or items that are alike the ones a user rated highly.

Content based methods can be based on either a classification problem (will a user like or dislike a given item) or a regression problem (prediction of the user's rating). Similar to the collaborative method, content based methods can be user-centric or item-centric. User-centric methods are based on the item features and recommend items that are similar to those a user already likes. Item-centric methods are based on the user features and recommend items that users with similar features rated highly.

As opposed to collaborative methods, content based suffer far less from the cold start problem. Now, only users or items with features unknown to the model will be a problem.

The content based methods will not be presented in more detail in this paper.

2.3 Methods used in the project

We decided to develop the system based on collaborative methods, as it is a very effective method that is relatively simple to use. Furthermore, there exist many papers to compare our results against for the dataset we're working with.

For reasons explained later, we've chosen to continue with the SVD++ algorithm for this project [1]. SVD++ is an extension of the SVD algorithm[2] in which implicit ratings are taken into account. It provides a better result in exchange for performance, as it is quite a bit more costly than normal SVD.

SVD, short for singular value decomposition, is a very popular linear algebra technique which breaks down a matrix into a product of smaller matrices. In the case of recommender systems, it may be used to find relationships between items. SVD in the context of recommender systems differs from the linear algebra technique. It utilizes the fact that with a user-item matrix M, one can obtain information regarding users and items by finding the matrixes $M \cdot M^T$ and $M^T \cdot M$, respectively, as they would provide person-to-person entries and item-to-item entries. This is possible as the two matrices would have the same set of eigenvalues. To understand exactly how this is mathematically possible one would have to study the SVD technique, which is out of scope for this project [2] [3]. The SVD++ algorithm will not be explained in detail here; important to know is simply that it includes the implicit ratings.

3 Dataset and Exploratory Data Analysis (EDA)

Important things to write about

- Removed movies with under 20 ratings (link to the notebook where I found the idea is in the code). This leaves 94968 ratings and 939 movies. (cold start discussion).
- Minimum number of ratings per user is 20 (cold start discussion)
- Good distribution of the data
- Big chunk of the ratings is from 3-5 (82 % specifically)

In this project, we will be working with the MovieLens 100k (ML100k) Dataset [4]. It is a dataset that consists of 100.000 ratings from 1000 users and 1700 movies. It is a relatively small, yet very useful, dataset that has been widely used. The movies are rated on a discrete scale of 1-5, and each user has rated at least 20 movies. The dataset includes features about the movies and the users, but these will not be used as we'll be using collaborative methods.

Before working with the dataset, an Exploratory Data Analysis (EDA) was performed. The following will present the results. First, the distribution of the ratings was examined. The result is shown in figure 2.

From the figure, it's clear to see that most of the ratings are from 3-5, specifically 82 %. Though this is not as spread as one might wish, it is still very useable for this project. Next, we examined the number of ratings per item. The result is shown in figure 3.

The figure shows a histogram of the spread of the number of ratings, where each bin has a width of 20 movies. From the figure, we can see that most movies have under 100 ratings, and

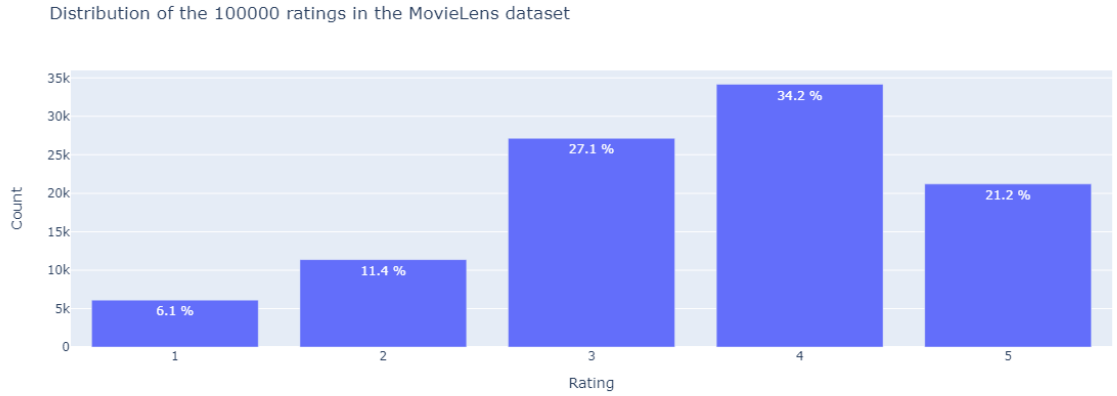

Figure 2: Distribution of the ratings.

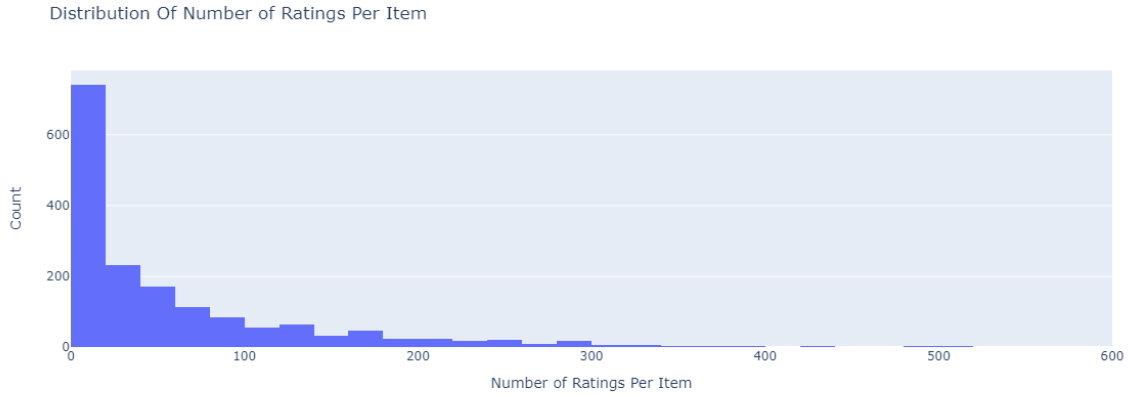

Figure 3: Distribution of the number of ratings per item

specifically most have under 20 ratings. This might become a problem, as movies with not a lot of ratings might suffer from the cold start problem as described earlier, and introduce a lot of unwanted randomness. For this reason, we chose to filter out all movies with under 10 ratings. This resulted in a new distribution, shown in figure 4.

This now leaves us with 1152 movies, which would still be enough to create a good model, as 97953 ratings are still present. The distribution is also acceptable to continue with. Next, we examined the distribution of the average rating of the items and the users, which is shown in figure 5 and 6, respectively.

From the figures, it's clear to see a nice spread, which is good to work with.
With the EDA performed, we were ready to implement the model.

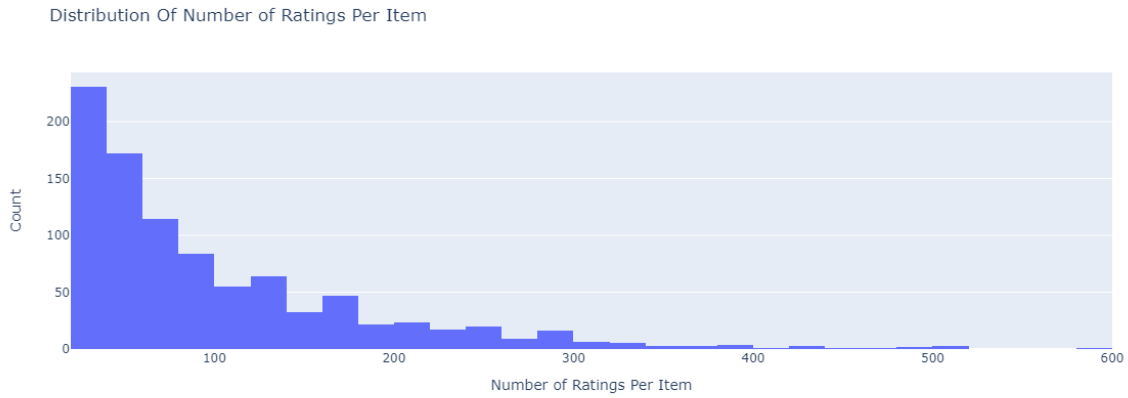

Figure 4: Distribution of the number of ratings per item after filtering out movies with under 10 ratings. Though it says it starts from 0, it starts from 10 (for some reason, couldn't figure out why).

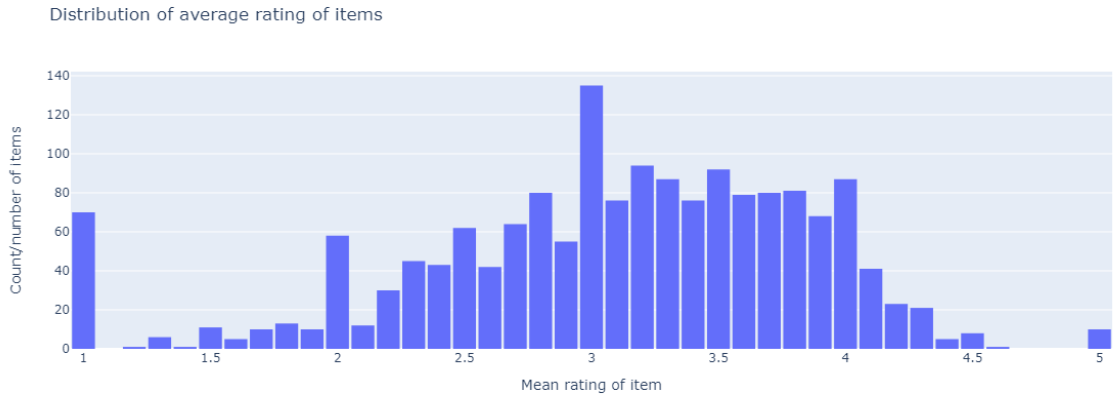

Figure 5: Distribution of average rating of the items in the dataset.

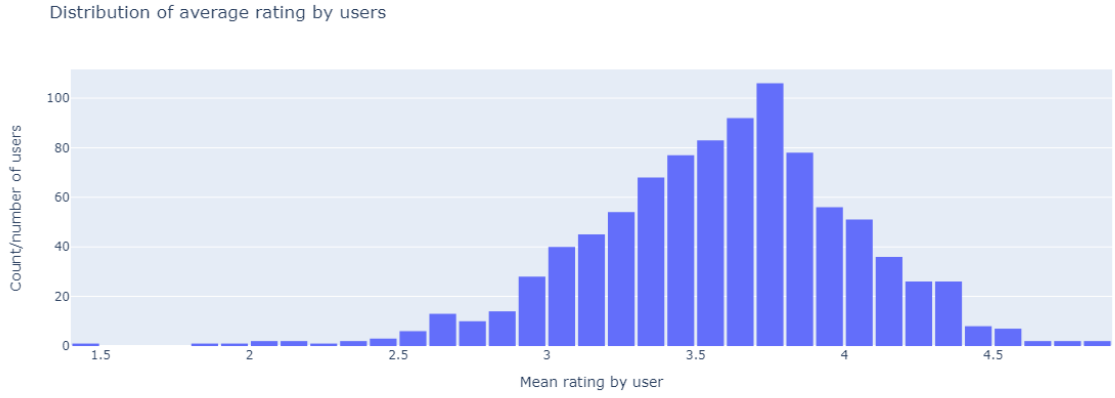

Figure 6: Distribution of average ratings by the users in the dataset.

4 Implementation

The implementation was carried out in Python, using the Surprise library [5]. Surprise is a kit developed specifically for building and analyzing recommender systems, and is very easy to use as it provides a plethora of tools and models. Along Surprise, Pandas for the handling of data. The code was developed with inspiration from [6][7][8][9][10].

4.1 Model selection

Prior to implementing a model, it must be chosen which one to implement. In order to find the optimal one, 10 different algorithms (both memory-based and model-based) were tested. These were tested against each other by cross-validation on each algorithm and comparing the root mean square error (RMSE) of each result to compare the algorithms' results. This was done using the surprise library's built-in `cross_validate` function, setting the number of folds to 3, and using the default training parameters. The result of the testing is shown in table 1.

Algorithm	RMSE score
SVD++	0.924218
KNNBaseline	0.931528
SVD	0.942135
SlopeOne	0.943485
BaselineOnly	0.944727
KNNWithZScore	0.950187
KNNWithMeans	0.950680
NMF	0.965079
KNNBasic	0.976228
NormalPredictor	1.508299

Table 1: Result of cross-validating the 10 different algorithms

As seen in the table, the algorithm with the best RMSE value was SVD++. Therefore, the SVD++ is what will be used for the implementation. For the training of the model, the standard Surprise parameters were used, which are 20 epochs, 20 factors, a learning rate of 0.007, and a regularization term of 0.02. The dataset was split into 80/20, where 20% is for testing and 80% is for training. With this, the `fit()` to train the model on the train set, from which we got an RMSE of 0.9140 on the test set.

5 Testing and results

As mentioned in the implementation section, the model achieved an RMSE of 0.9140. In addition to that, we decided to manually look at the model's predictions. First off, we looked at the model's 10 best predictions, where the model predicted exactly what the user rated the movie. One might question if these were lucky guesses. However, we would argue they are not, as all the movies have between 24-259 ratings. With that many ratings, it would imply that the model is accurately doing it based on previous ratings, as opposed to lucky shots.

Some of the worst guesses give room for more speculation, as the two worst predictions are on movies with 126 and 291 ratings, meaning the model should not be performing that poorly. However, after closer inspections of the average rating of the items, the user's ratings are clear outliers, which explains the poor performance of the model.

With this, we moved on to actually recommend movies to users. We created the function `generate_recommendation` which provides a list of 5 recommendations (by default) to the user. It does so by randomly sampling from the recommendations until it finds 5 different movies with a predicted score over a certain threshold, which by default is 4. A test for user 200 gives the 5 recommendations provided in table 2.

ID	**Title**	**Release Date**	**Genre(s)**
888	One Night Stand	January 1st 1997	Drama
524	The Great Dictator	January 1st 1940	Comedy
1319	The Neon Bible	March 1st 1995	Drama
1399	Stranger In The House	January 1st 1997	Thriller
1388	Gabbeh	June 27th 1997	Drama

Table 2: Five recommended movies by the model

Comparing the recommendations to the user's top 5 rated movies, it can be observed that the suggested drama, comedy, and thriller movies are well-aligned with the user's highest-rated movies in those genres.

5.1 Comparison to other models

As the ML100k dataset is widely used, many others have tried different methods of achieving the best possible model. These are all collected in PapersWithCode, a website used to see state-of-the-art performance on many datasets[11]. Some well-performing models on the ML100k dataset are summarized in table 3.

Model	**RMSE**
GHRS	0.887
MG-GAT	0.890
GraphRec + Feat	0.897
SRGCNN	0.929
SVD++	0.9140

Table 3: Comparison of RMS value for different models

As presented in the table, our model is not the best-performing model, as SOTA achieves an RMSE of under 0.9, yet it performs fairly well for the scope of this project.

6 Discussion

Though we obtain a model that performs well, evaluating how this performance might be improved is important. Firstly, the model was only trained with the standard hyperparameters in Surprise, which means there might exist some hyperparameters that yield better performance in training. Ideally, a grid search of different hyperparameters would be conducted, as to find the optimal parameters.

Furthermore, in this project, we also only tested collaborative methods, neglecting the potential benefits of content based method. It is possible that content-based methods could yield even better results, especially when used in conjunction with collaborative filtering methods in a hybrid model. By using a hybrid approach, it might be possible to reintroduce items with fewer than 10 ratings, as content-based methods are less affected by cold start problems.

Additionally, it is worth noting the tradeoff between performance and precision. Though SVD++ was chosen due to superior performance, the complexity of the model greatly increases both the fit and test time, both taking over 15 times longer compared to standard SVD. Despite the improvement of 0.02 in the RMSE in the cross-validation, it might not be worth the tradeoff if the model is to be applied to a large-scale product.

7 Conclusion

In this paper, we've introduced, implemented, and tested recommender systems using collaborative-based methods on the MovieLens 100K dataset. Among ten different algorithms evaluated through cross-validation, the SVD++ model was selected, yielding a final result of 0.9140 RMSE. Although this is not state-of-the-art performance, it is a highly acceptable outcome within the scope of this project.

As a result, we have developed a functioning model capable of recommending relevant movies to users. However, if the model is to be deployed in a large-scale system, it may be worth reconsidering the choice of SVD++ as the optimal model due to its increased complexity compared to other available options.

Bibliography

[1] S. Wang, G. Sun, and Y. Li, "Svd++ recommendation algorithm based on backtracking," *Information*, vol. 11, no. 7, 2020, ISSN: 2078-2489. DOI: 10.3390/info11070369. [Online]. Available: https://www.mdpi.com/2078-2489/11/7/369.

[2] D. Billsus and M. J. Pazzani, "Learning collaborative information filters," 1998. [Online]. Available: https://www.ics.uci.edu/~pazzani/Publications/MLC98.pdf.

[3] Y. Zhang, "An introduction to matrix factorization and factorization machines in recommendation system, and beyond," 2022. DOI: 10.48550/arXiv.2203.11026. [Online]. Available: arXiv:2203.11026.

[4] Grouplens. "Movielens 100k dataset." (), [Online]. Available: https://grouplens.org/datasets/movielens/100k/ (visited on 05/10/2023).

[5] N. Hug. "Surprise: A python library for recommender systems." (2020), [Online]. Available: https://doi.org/10.21105/joss.02174.

[6] divensambhwani. "Movie recommender system using surprise library." (), [Online]. Available: https://github.com/divensambhwani/MovieLens-100K_Recommender-System/blob/master/MovieLens-100K-Recommeder%20System-SVD.ipynb (visited on 05/10/2023).

[7] S. Li. "Building and testing recommender systems with surprise, step-by-step." (Dec. 26, 2018), [Online]. Available: https://towardsdatascience.com/building-and-testing-recommender-systems-with-surprise-step-by-step-d4ba702ef80b (visited on 05/10/2023).

[8] N. Hug. "Analysis of the knnbasic algorithm." (), [Online]. Available: https://towardsdatascience.com/how-you-can-build-simple-recommender-systems-with-surprise-b0d32a8e4802 (visited on 05/10/2023).

[9] A. Mavuduru. "How you can build simple recommender systems with surprise." (), [Online]. Available: https://nbviewer.org/github/NicolasHug/Surprise/blob/master/examples/notebooks/KNNBasic_analysis.ipynb (visited on 05/10/2023).

[10] N. Hug. “Surprise documentation.” (), [Online]. Available: https://surprise.readthedocs.io/en/stable/index.html (visited on 05/10/2023).

[11] Kaggle. “Recommendation systems on movielens 100k.” (), [Online]. Available: https://paperswithcode.com/sota/collaborative-filtering-on-movielens-100k (visited on 05/10/2023).