Tim Benjamin Hoffmann

MŰEGYETEM 1782

Budapest, May 28, 2025

Movie Recommender System Using MovieLens

Contents

1 Introduction

1.1 Motivation

In an age where streaming platforms and digital libraries offer thousands of films at the click of a button, viewers are often overwhelmed by choice. Traditional browsing or keyword search can leave users sifting through endless lists, only to abandon the effort or settle for familiar titles. A thoughtfully designed recommender system can transform this experience by learning individual preferences and guiding each user toward films they are most likely to enjoy. Beyond improving satisfaction and engagement, personalized recommendations foster deeper exploration of content, helping audiences discover hidden gems and niche genres. This project embraces that potential by building an interpretable yet effective movie recommender, showcasing how even straightforward methods can deliver meaningful value to both end users and content providers.

1.2 Problem Description

At its core, this project seeks to construct a system that suggests unseen films likely to resonate with a given viewer. Instead of relying on an exhaustive manual search or generic popularity rankings, the recommender infers personal taste from past interactions such as ratings and tags and subsequently ranks candidate movies according to predicted interest. The challenge lies in translating sparse feedback into reliable signals, balancing the trade-off between recommending popular blockbusters and uncovering lesser-known titles, and packaging the logic in a transparent, understandable framework. By focusing on user-centric prediction and clear evaluation of top-N suggestions, our aim is to demonstrate how different simple personalized recommender systems can achieve relevant results.

1.3 Literature Review

1.3.1 Collaborative Filtering Approaches

A: Memory-Based Methods

Early research in recommender systems often relied on memory-based CF, typically *user-based* or *item-based* approaches. User-based methods compute similarities between users to predict ratings [1]. In contrast, item-based CF analyzes item-to-item similarities, which have been shown to be effective and scalable on datasets such as MovieLens [2].

Despite their interpretability, memory-based methods can suffer from data sparsity and scalability issues as the number of users and items increases. However, for a relatively small dataset like MovieLens 100K, user-based or item-based CF generally achieves acceptable accuracy in predicting ratings (often with root mean square error (RMSE) around 0.93–0.95) [1, 2].

B: Model-Based Methods

Model-based CF addresses certain limitations of memory-based approaches. Matrix factorization (MF) techniques such as singular value decomposition (SVD) gained prominence after demonstrating significantly improved accuracy [3]. These methods learn latent features for both users and items, producing more accurate predictions.

More advanced variants include SVD++, factorization machines, and neural collaborative filtering (NCF) [4]. While these methods can offer further improvements, the benefits are most

pronounced on larger datasets. For the relatively small MovieLens 100K dataset, simpler methods (e.g., vanilla SVD or even item-based CF) suffice.

1.3.2 Content-Based Filtering

Content-based filtering (CBF) uses item attributes (genre, cast, director) to profile user preferences [5]. In a movie context, it learns which features (e.g., genre tags) a user likes, then recommends items sharing those attributes. CBF eliminates the need for user ratings to overlap, making it effective for *new items*. However, purely content-based methods can yield less diverse recommendations, as they tend to stay within a narrow range of items similar to what the user already prefers. In MovieLens 100K, the item metadata is fairly limited (primarily genre flags), which can restrict pure CBF accuracy unless used alongside collaborative signals [6].

1.3.3 Performance Metrics and Reported Results

Performance in rating prediction is commonly measured by mean absolute error (MAE) or RMSE [1]. In top-N recommendation scenarios, metrics such as precision, recall, and normalized discounted cumulative gain (NDCG) help evaluate the quality of ranked recommendations [7]. Typical CF-based approaches achieve an RMSE around 0.93–0.95 on MovieLens 100K, whereas matrix factorization can reduce that to about 0.90 [3]. Even marginal improvements in RMSE can significantly improve user satisfaction.

In addition to accuracy, other considerations include coverage (the fraction of items that can be recommended), novelty, diversity, and user satisfaction. While these broader metrics are often discussed, rating prediction remains a straightforward quantitative benchmark.

1.3.4 Recommended Models and Evaluation

For a small dataset like MovieLens 100K, an *item-based* or *user-based* CF approach is simple to implement and interpret. Matrix factorization would likely provide slightly better accuracy, but may require more advanced tooling. Content-based methods are an option if one wishes to leverage genre or textual information, though, for MovieLens 100K, CF typically outperforms a pure content-based approach.

Given that tools like RapidMiner and Tableau offer user-friendly GUI-based workflows, a memory-based CF or a simplified model-based approach is a good choice. The results of the developed models can be evaluated through RMSE/MAE on a held-out test set, aiming for an RMSE in the 0.90–0.95 range [1–3].

1.3.5 Approach to follow

Extensive research shows that collaborative filtering, especially item-based CF or matrix factorization, consistently performs well on relatively small datasets such as MovieLens. Content-based filtering can improve recommendations when combined with collaborative filtering, particularly in the handling of new or niche items. However, for a concise project focused on the MovieLens 100K dataset, a memory-based CF solution or even a content-based filtering evaluated using standard metrics (e.g., RMSE) offers both conceptual clarity and practical feasibility, aligned with proven methods in the literature. Table 1 concisely compares the reviewed approaches.

Approach	Complexity	Performance	Best for Dataset Size
Memory-Based CF	X	XX	small
Model-Based CF	XXX	XXX	medium–large
Content-Based Filtering	XX	XX	small–medium

Table 1: Concise Comparison of Recommender System Approaches, comparing the complexity, expected performance and best dataset sizes for which the approaches are applicable

2 Data and modeling

2.1 Data Description

In this project, the `ml-latest-small` MovieLens dataset maintained by GroupLens Research at the University of Minnesota is used [8]. This dataset is designed as a development resource for exploring movie recommendation systems in both research and educational contexts.

Overview and Statistics

- **Ratings and Tags:** The dataset includes 100,836 ratings and 3,683 tag applications, reflecting the 5-star rating scale (in half-star increments) and free-text tagging behavior of users.
- **Movies:** A total of 9,742 distinct movies are represented, covering a wide range of genres.
- **Users:** Data comes from 610 users, each having rated at least 20 movies. No demographic information is provided.
- **Time Span:** The rating and tagging activity spans from March 29, 1996 to September 24, 2018. The snapshot of the dataset used in this project was generated on September 26, 2018.

Dataset Structure All data are stored in CSV files with a single header row. Each file provides a distinct facet of the MovieLens data:

- `ratings.csv`: Contains all 100,836 ratings in the format:

  $$\textit{userId}, \textit{movieId}, \textit{rating}, \textit{timestamp}$$

  Each line corresponds to a single rating event. Ratings range from 0.5 to 5.0 (inclusive) in increments of 0.5, and timestamps are in Unix time.

- `tags.csv`: Contains 3,683 tags in the format:

  $$\textit{userId}, \textit{movieId}, \textit{tag}, \textit{timestamp}$$

  Tags are user-generated keywords or short phrases. Like `ratings.csv`, this file is ordered first by *userId* and then by *movieId*.

- `movies.csv`: Describes each of the 9,742 movies with:

  $$\textit{movieId}, \textit{title}, \textit{genres}$$

  Genres are listed in a pipe-separated format (e.g., "Action—Comedy"), chosen from 18 distinct categories such as `Drama`, `Thriller`, and `Sci-Fi`.

- `links.csv`: Maps each *movieId* to external databases using:

  $$\textit{movieId}, \textit{imdbId}, \textit{tmdbId}$$

  These identifiers enable cross-referencing with IMDb and The Movie Database (TMDb).

In summary, the `ml-latest-small` dataset furnishes a well-balanced platform to investigate recommendation strategies. Its moderate size (fewer than one million ratings) allows for quick model iterations, while the inclusion of free-text tags and diverse genre labels makes it suitable for both collaborative filtering and content-based experimentation.

2.2 Data Preprocessing

The raw MovieLens data is consolidated into two new files: a comprehensive **user data file** and a **movie data file**. These transformations are performed in RapidMiner using a series of operators to join, generate, and aggregate data. Figures 1, 2 and 3 show the RapidMiner processes used to create these files.

User Data File The goal is to summarize the preferences of each user in a single row. Specifically, movies that a user *liked* or *did not like* based on their ratings provided are identified by defining the rating value of three as a threshold value. This value is chosen because no information gain is expected from movies with a rating of three, since the user did not particularly *like* nor *dislike* the movie they rated. Subsequently, the tags and genres associated with these liked and disliked movies are collected and aggregated to result in these six columns:

- **User ID:** The unique identifier for each user.
- **Tags Liked:** A concatenation of all the tags applied to movies the user rated better than 3.
- **Tags Disliked:** A similar concatenation of all tags from movies rated lower than 3.
- **Genres Liked:** A list of genres associated with movies with positive user ratings.
- **Genres Disliked:** A list of genres associated with movies with negative user ratings.
- **Movies Watched:** A compiled list of all *movieId* values a user has rated and therefore has already watched.

This arrangement provides a clear snapshot of each user's preferences. The two columns containing tags (liked and disliked) and the two columns containing genres (liked and disliked) serve as immediate indicators of interest for content-based filtering. Meanwhile, the `Movies Watched` column ensures the system avoids recommending movies a user has already seen.

Movie Data File The second generated file focuses on movie-level attributes. For each movie in the dataset, the following attributes are stored:

- **Movie ID:** The unique identifier for each movie.
- **Title:** The full title of the movie.
- **Associated Tags:** An aggregate of all tags users applied to this movie.
- **Genres:** The genre labels (e.g., "Action," "Drama,") initially provided by the dataset.
- **Rating:** The average aggregated rating for each movie from 0 to 5 in 0.5 steps.

By gathering user-generated tags alongside the original genre data, this file becomes a convenient reference for content-based recommendations. For instance, a system can compare a user's preferred tags or genres with those in this file to produce relevant suggestions.

RapidMiner Process Setup Within RapidMiner, a series of operators are used to:

1. *Join* and merge the `ratings`, `movies`, and `tags` tables.
2. *Filter* based the user rating threshold to distinguish between liked and disliked movies.
3. *Aggregate* tags, genres, and movie identifiers into single row entries per user and movie.
4. *Export* the final results to two separate CSV files: one for user data and one for movie data.

Figure 1 provides a high-level overview of this workflow, while Figures 2 and 3 show the steps performed for each file in more detail. After generating these two CSV files, the rest of our recommendation pipeline can easily refer to the preferences of each user and the available pool of movies.

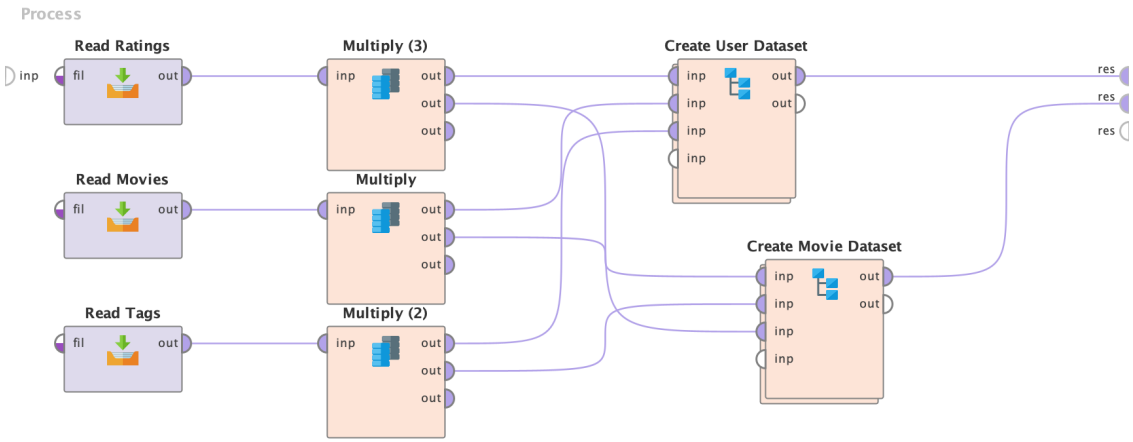

Figure 1: High-level RapidMiner process flow for data preprocessing.

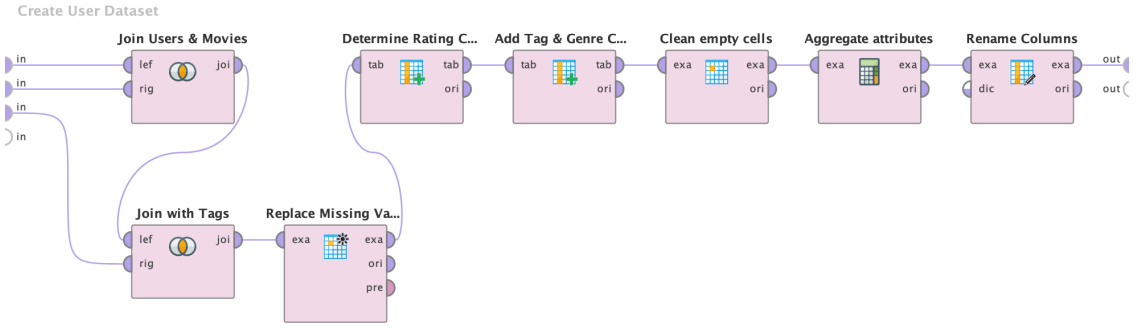

Figure 2: Detailed view of operators used to create the user file.

With these two consolidated files, collaborative filtering and content-based methods can be applied more intuitively.

2.3 Modeling

In this project, two types of recommendation systems are created. As explored in Section 1, both the memory-based CF, in particular user-based CF, and the content-based filtering are worth

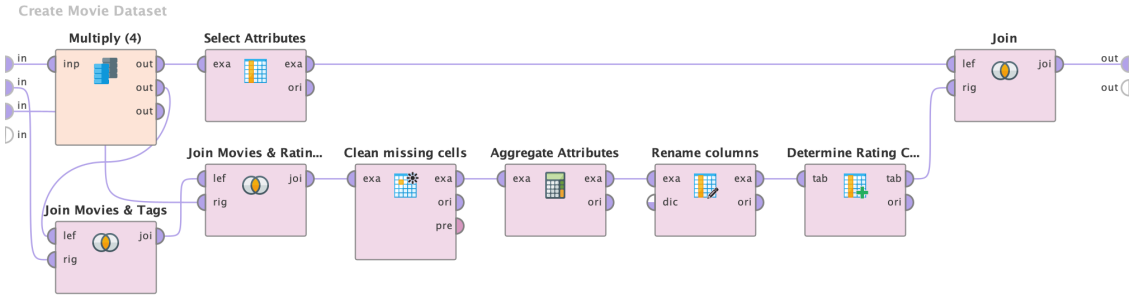

Figure 3: Detailed view of the operations used to create the movie file.

implementing. The memory-based CF aims to find the best user-user matches to recommend movies, while the content-based filtering aims to find the best user-movie matches.

Fig. 4 shows the implementation of a user-based collaborative filtering approach, while Fig. 5 shows the implementation of a content-based filtering approach for movie recommendations.

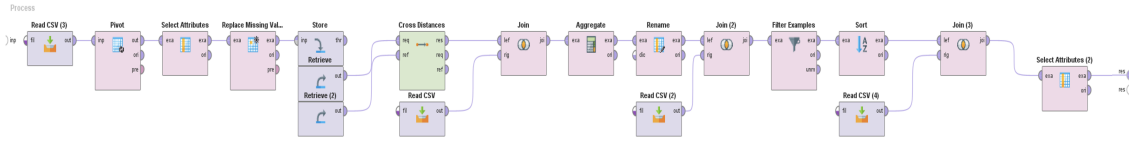

Figure 4: User-based filtering implementation in Rapidminer to recommend best matching movies to a user based on the interests of similar users.

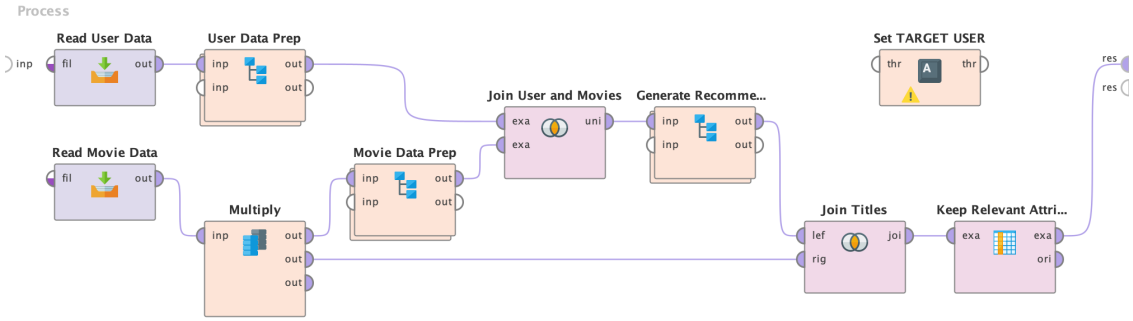

Figure 5: Content-based filtering implementation in Rapidminer to recommend best matching movies to a user based on previously liked movies considering both tags and genres.

2.3.1 User-based Collaborative Filtering

The user-based collaborative filtering model as seen in Fig. 4 starts by reading a custom csv file containing *userId*, *movieId*, and *rating* triplets. Subsequently, the *pivot* operator is used to convert the data such that each row is a user, each column a movie, and the cells store the rating a user gave this particular movie. Next, *userId* is discarded temporarily to sort through the data, and the missing values are replaced with zero to ensure that similarity is not biased. Before computing the similarity using Cosine Similarity, the full matrix is saved and copied for later comparison. Now, the similarity is calculated and a KNN algorithm is employed to find the ten best matches for each user. Following this, the similarity results are joined with the rating table to retrieve the ratings of

each neighbor. For each user-movie combination, the *aggregate* operator then averages the ratings from the most similar users to predict each the rating a user would give a particular movies if they were to rate it.

To filter out already-watched movies, the next step is to join the original ratings of the target user and remove movies the user has already watched by checking whether the original rating is greater than zero. Now, the recommendations are sorted by their predicted rating in descending order and joined with relevant movie metadata such as title and genre. The final *Select Attributes* operator keeps only the final relevant recommendation columns: *userId*, *movieId*, *title*, *predicted rating*

2.3.2 Content-based Filtering

The content-based filtering model as seen in Fig. 5 consists of several steps. It starts by reading the custom user and movie datasets described in Sec. 2.2 and preparing them for processing within the model. The Operator *Set TARGET USER* allows the target user to be specified by their unique user ID. For this user, the model generates recommendations.

In the *User Data Prep* subprocess, as shown in Fig. 6 the row corresponding to the target user is extracted, followed by the restructuring of columns to use a space instead of a vertical bar as separator. This is relevant for vectorization later in the model. Additionally, a macro is defined to save all movie IDs of movies the user has already watched, so these can be filtered out before returning the final recommendation. Subsequently, a new column is created, joining positive tags and genres together in one single cell, the movieId -1 is given to the user as a unique identifier to distinguish the user from the movies and the Profile column is transformed to text form.

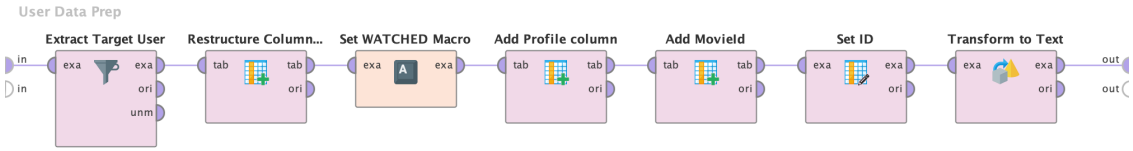

Figure 6: *User Data Prep* subprocess of the content-based filtering model implemented in Rapidminer.

In the *Movie Data Prep* subprocess, as shown in Fig. 7, similar steps are taken to preprocess the data for the model as to the *User Data Prep* subprocess. The genre and tag columns are restructured to use a space instead of a vertical bar as separator, a profile column is created combining genres and tags associated with each movie, and this column is transformed to text form.

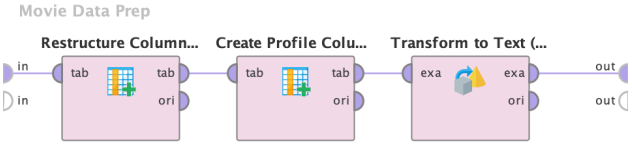

Figure 7: *Movie Data Prep* subprocess of the content-based filtering model implemented in Rapidminer.

Following the two subprocesses, the user row and movie rows are joined. Within the *Generate Recommendations* subprocess seen in Fig. 8, the *Profile* column of the movies and the user is vectorized and the other columns are dropped. Subsequently, employing Cosine Similarity, the similarity of the user preferences with the attributes of each movie is calculated. This is the core

of the model. Subsequently, the movies that the user has already watched are filtered out and the recommendations are sorted by similarity to put the best recommendations at the top.

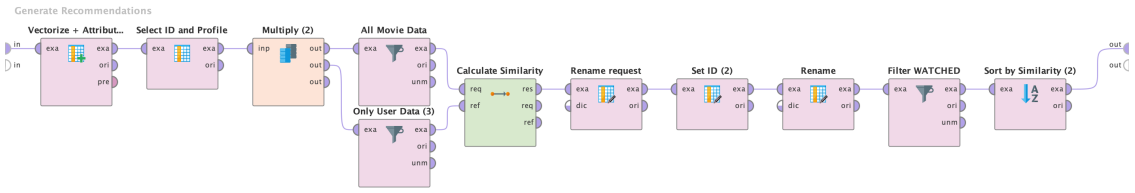

Figure 8: *Generate Recommendations* subprocess of the content-based filtering model implemented in Rapidminer.

Finally, the title of each movie is merged with the recommendations, so they can be easily interpreted by the *Join Titles* operator in combination with the *Keep Relevant Attributes* operator, and the sorted recommendations can be returned by the model.

3 Results

In this chapter, the results of this project are presented. These include exploratory data visualizations as well as evaluation results of the implemented models.

3.1 Exploratory Visualizations

A: Overall Rating Distribution

In Fig. 9 the distribution of ratings per user is shown. This diagram shows how many users have rated and therefore watched a particular number of movies. This is especially relevant information when it comes to extracting a test dataset from the entire dataset. Because enough test data must be used to retrieve representative metrics of the model whilst keeping the data the model has access to as close to the original distribution as possible, to minimize the impact of the split itself on the model performance.

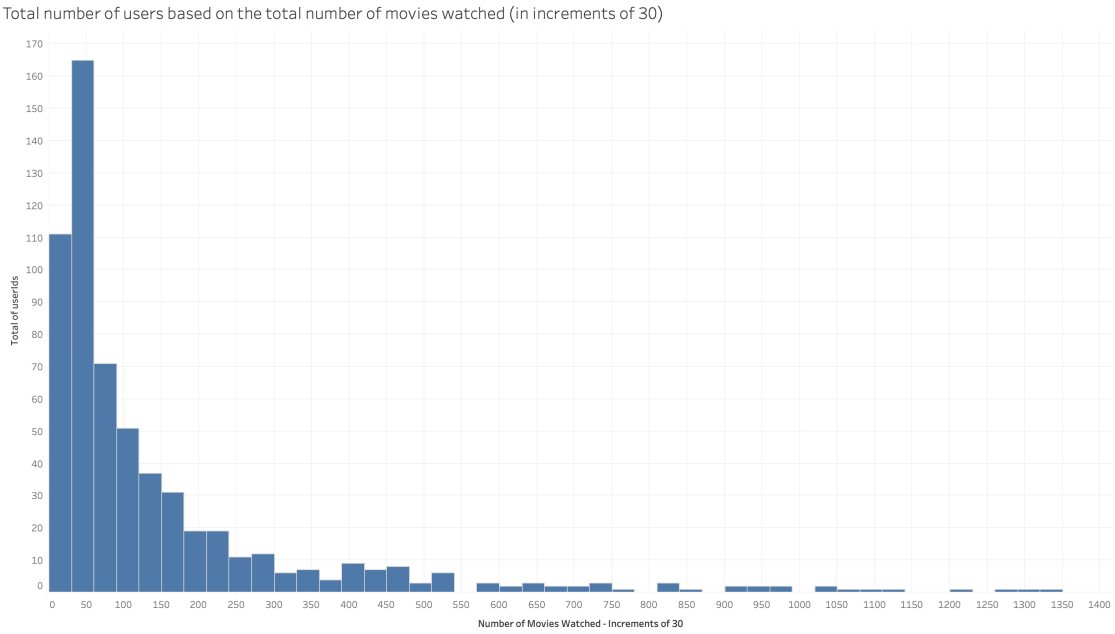

Figure 9: Distribution of users per movies rated, i.e., how many users watched and rated a particular amount of movies cut into sections of 30 movies.

B: Genre Analysis

This subsection gives insights into the genres featured in the dataset and their key characteristics. Therefore, Fig. 10 shows the relative relevance of each genre included in the dataset. It uses the sizing of bubbles to indicate the number of ratings a particular genre has, and colors to indicate the average rating a particular genre has received. In this way, popularity is put in the context of both rating and viewership of genres. To obtain this visualization, the hot encoding method is used to create columns by genre and then permuted them to create a bubble by genre.

Fig. 10 shows a tendency for the average rating score to decrease with the amount of people who have rated a particular genre of movies, e.g., the largest genre *comedy* is significantly less popular

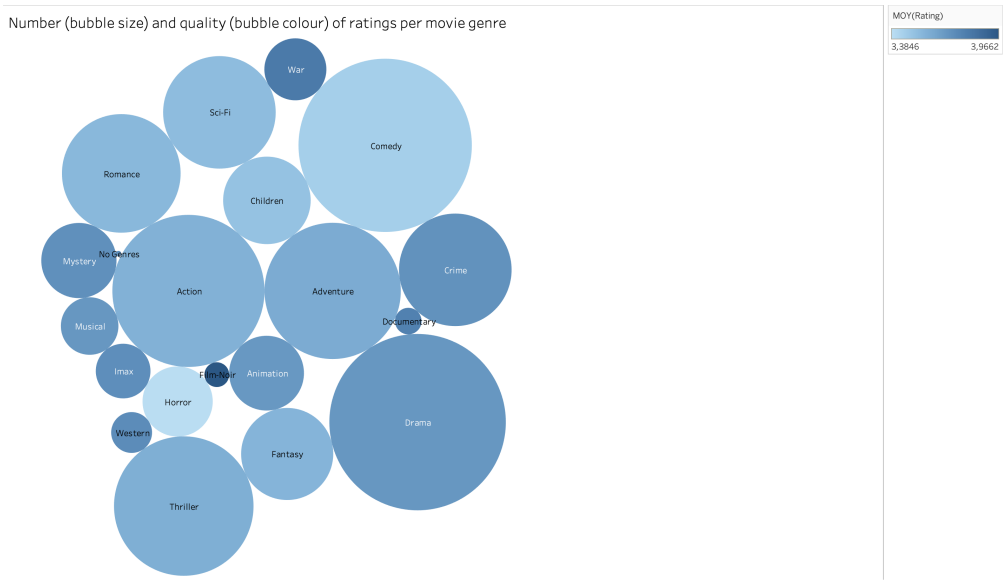

Figure 10: Overview of movie ratings. Bubble size indicates number of ratings a genre has received (i.e., prevalence), color indicates the average rating (i.e., popularity)

Figure 11: User activity levels, e.g., number of ratings or tags per user.

than a less often rated one like *animation*. However, this is just a tendency, as there are notable outliers such as the *drama* genre which has almost as many ratings as the *comedy* genre but a significantly better average rating. On the other side, the *horror* genre is rather poorly rated for the relative amount of ratings it received.

Another relevant visualization is shown in Fig. 11, which is also created using the hot encoding method to create the genre columns and then permute them to create a histogram per genre. It shows the number of *best rated* movies per genre, i.e., ratings greater than or equal to 4. Similarly to Fig. 10 it shows the relative dominance of the *comedy* and *drama* genres. This fact is important to keep in mind when creating the recommendation model, as the relatively large number

of movies of particular genres could lead to favoring these over other less frequently rated movie genres. This is especially problematic, as a tendency towards better ratings for less prevalent genres has been observed. Therefore, favoring the most prevalent movies could lead to lower-quality recommendations

C: User Tags

Figure 12: Scatter plot showing how many good tags are are given by each user depending on the amount of movies they have rated in total. Each point represents one user.

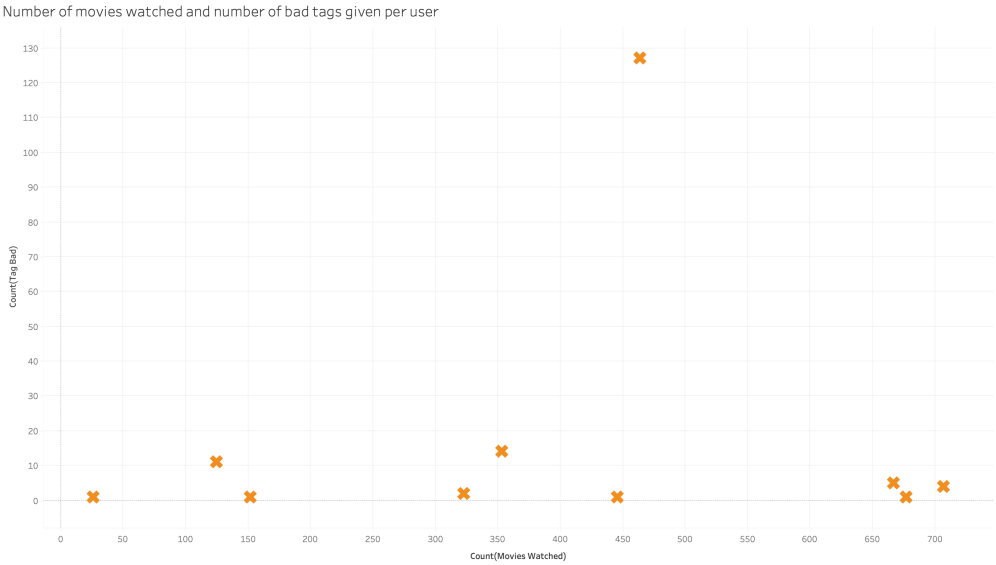

Figure 13: Scatter plot showing how many bad tags are are given by each user depending on the amount of movies they have rated in total. Each point represents one user.

To visualize the number of tags in the dataset that can be utilized for the recommendation

system, the number of good and bad tags a user has given for a certain number of movies are shown in Figures 12 and 13 respectively. Users who have not given any tags are not included in the diagrams. A good tag is a tag associated with a good rating for a movie ($>= 3$) and a bad tag is a tag associated with a bad rating of a movie (< 3).

Several key observations can be made by analyzing Figures 12 and 13. Firstly, the dataset consists of significantly more good than bad tags. A reason for this could be that there are in general more good than bad ratings in the dataset. The fact that users rarely provide bad tags and more often good tags could also jeopardize the quality of suggestions based on bad tags, as these are only provided by a relatively small set of users. Therefore, their opinions are overly represented within the bad tags. Consequently, bad tags are not considered in either recommendation model.

3.2 Model Evaluation

To accurately evaluate the performance of both models and compare them, the precision metric is used. In the context of movie recommendation systems, we define the precision of a model as follows: The number of movies that a given user has positively rated, that is, a rating above or equal to 3, of the top ten recommended movies summed by all users. Therefore, positive-rated movies are the True Positives, and the total number of recommendations is the sum of True Positives and False Positives.

$$P = \sum_{u=0}^{n} \frac{TP}{TP + FP} \tag{1}$$

We pick the top ten recommendations, so $TP + FP = 10$. In order to know whether a user liked or disliked a movie for all top ten recommendations, all of these need to be within the list of rated movies for any given user. Therefore, the ratings of each user are split 80/20, with 80% of the ratings remaining in the training set, that is, data available to the model to make predictions, and 20% of the ratings placed in a test set. To divide the data, stratified sampling is used, so both sets contain a random number of positive and negative ratings.

Fig. 14 shows the total number of recommendations and the number of movies with positive and negative rating thereof. The resulting precision metrics of both models are similar, with 0.8572 and 0.8366 for the user-based and the content-based model, respectively.

For the user-based recommendation model, the root-mean-square error (RMSE) value is also calculated. This is possible because the model predicts a rating that a user would give a movie based on the ratings of similar users. Therefore, the difference between predicted and actual rating for a movie in the test set can be calculated and summed for all users. Ultimately, the RMSE is 1.072, which means that the average error for a predicted rating is roughly a rating of ± 1. Consequently, movies with a predicted rating of 2.0 to 4.0 are more at risk of being falsely recommended as a good fit for a user. However, as long as there are enough movies with a predicted rating greater than 4.0, most recommendations should be relevant to a given user, as indicated by the precision score of 85.72%.

Finally, we evaluate both models for ourselves, providing 30 movies with individual ratings for each of us, and evaluating the recommended movies. Both models have generated 10 recommendations of movies that we might like based on our specified ratings. Tables 2 and 3 show the result of this test. They roughly align with the results of the analysis with the test sets, further strengthening our confidence in the precision scores calculated.

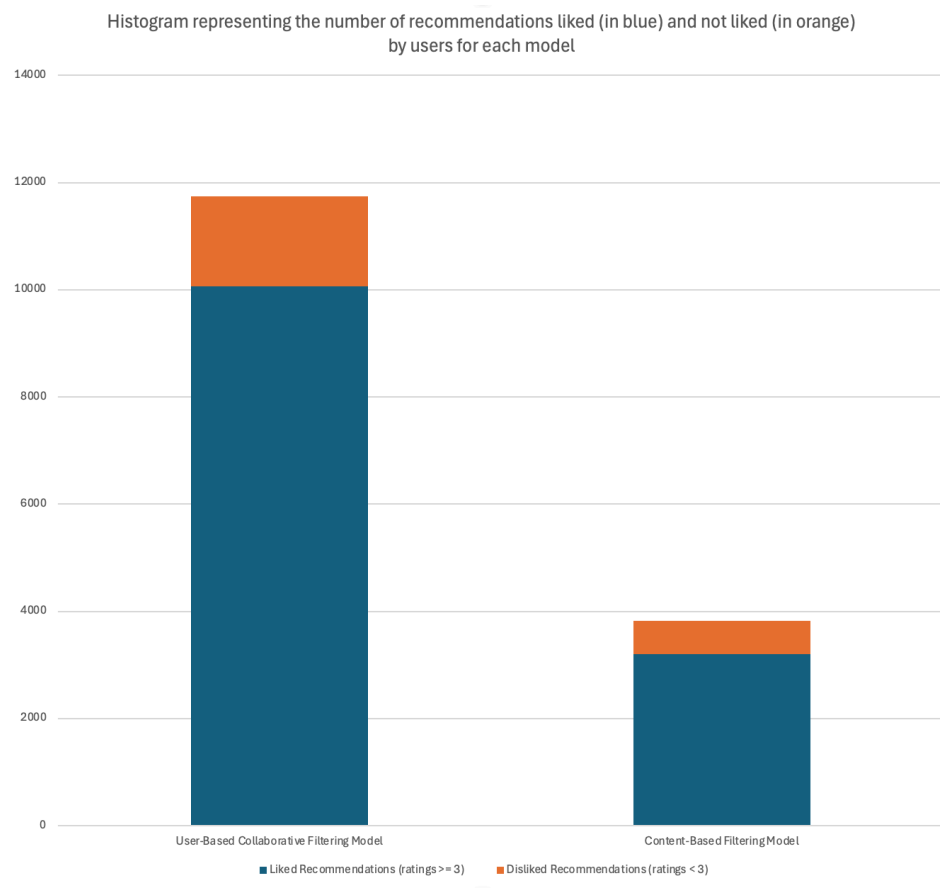

Figure 14: Histogram showing the number of True Positives and False Positives within the total number of recommended movies.

	Tim	Ruhan	Quentin	Total
Liked	8	7	10	25
Disliked	2	3	0	5

Table 2: Personal evaluation results of user-based collaborative filtering model.

	Tim	Ruhan	Quentin	Total
Liked	7	8	9	24
Disliked	3	2	1	6

Table 3: Personal evaluation results of content-based filtering model.

4 Conclusion

In this project, we developed and evaluated two recommendation systems, user-based collaborative filtering (CF) and content-based filtering (CBF) leveraging the MovieLens ml-latest-small dataset. The CF model predicts the rating of a user for unseen movies by aggregating the ratings of their most similar peers, while the CBF model compares the genre and tag profile of each movie directly against the preferences profile of a user. Through a stratified 80/20 train–test split, we measured precision on the top-10 recommendations and, for CF, also calculated the RMSE of predicted ratings. The user-based CF approach achieved a precision of 85.7% with an RMSE of 1.07, indicating a reliable identification of favored movies. The CBF approach achieved a precision of 83.7%, indicating that both models deliver recommendations of similar quality. Our exploratory analysis revealed important dataset characteristics: genre popularity does not always correlate with higher ratings (e.g., Comedy is highly prevalent but less well rated, while Drama commands high average scores), and users tend to apply far more 'good' tags than 'bad', which leads us to focus on positively tagged content. These insights informed model design choices, such as excluding sparsely populated negative-tag signals, resulting in robust recommendation performance. Furthermore, the evaluation of the recommendations on our own preference profiles showed the effectiveness of both models to recommend mainly relevant movies. Therefore, we conclude that two recommendation systems for movies based on user ratings have been successfully developed and evaluated.

References

[1] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998, pp. 43–52.

[2] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th International Conference on World Wide Web (WWW)*, 2001, pp. 285–295.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web (WWW)*, 2017, pp. 173–182.

[5] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The Adaptive Web: Methods and Strategies of Web Personalization*, ser. Lecture Notes in Computer Science. Springer, 2007, vol. 4321, pp. 325–341.

[6] P. Melville, R. J. Mooney, and R. Nagarajan, "Content-boosted collaborative filtering for improved recommendations," in *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 2002, pp. 187–192.

[7] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 5–53, 2004.

[8] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.