Movie Recommender Web application

By Marshall Lee

Springboard

March 21th, 2023

Recommender systems have become very popular over the last couple decades. For example, Spotify's user base has more than doubled since 2019, which is likely due to "Discover Weekly" playlists. Many of my friends prefer to use Spotify over other applications because of the playlist's quality recommendations. Netflix also has a refined recommendation system, which has given them the competitive edge over other streaming services. In 2009, they even hosted a competition to see who could deliver the best collaborative filtering algorithm - and gave away a prize of $1,000,000[1]!

What is the secret to a quality recommender? What constitutes a "good" recommendation versus a "bad" one? My goal for this project is to build a movie recommendation website from scratch in order to try and understand the foundations of a good recommender. I will be using the public MovieLens 20M dataset[2], as well as multiple filtering techniques to try and come up with the appropriate algorithm.

Data Wrangling

The original dataset consists of four tables: a table with all users and their ratings, a tags table with all user "tags" of a movie, a movies table with movie-Ids and titles, and a metadata table containing the metadata for each movie. After dealing with duplicate values, removing null values, and aggregating the data together, we get a clean dataframe with the following features:

- *userId* - unique for every user
- *MovieId* - unique for every movie
- *Rating* - on scale from 0.5-5
- *Date* - date timestamp of rate and tag
- *Rating_average* - average rating of each movieId
- *Rating_count* - rating count of each movieId
- *Genres* - list of genres for each movieId
- *imdbId* - unique for every movie
- *tmdbId* - unique for every movie. This will be used to retrieve the poster urls
- *Title* - the movie title
- *Release_year* - release year of movie
- *Original* language - the original language of the movie
- *Popularity* - tmdb popularity rating
- *Runtime* - runtime in minutes

---

[1] Source: https://www.nytimes.com/2008/11/23/magazine/23Netflix-t.html

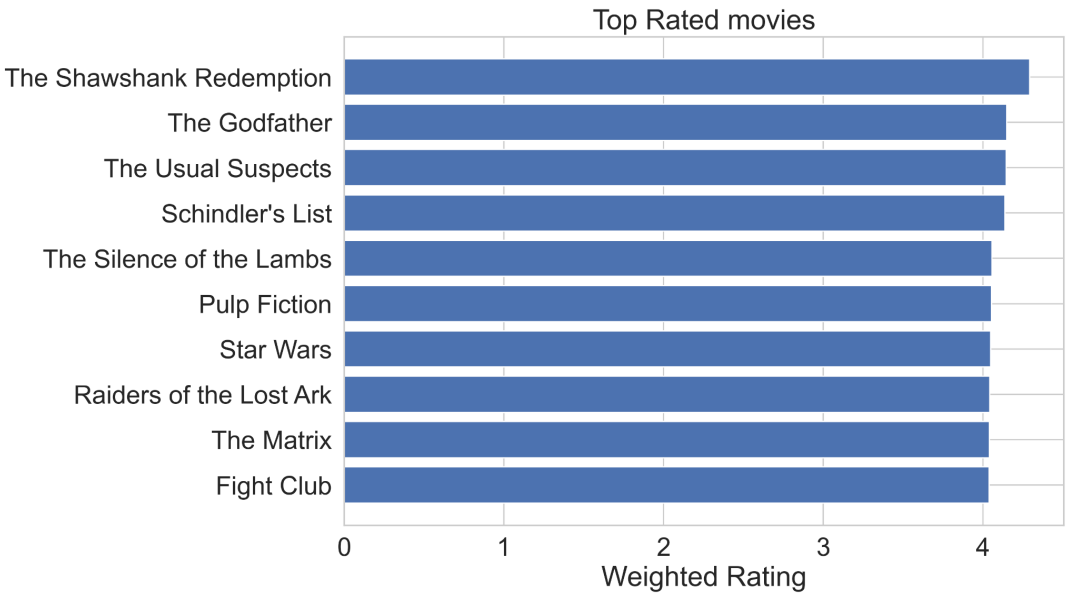
[2] Source: F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

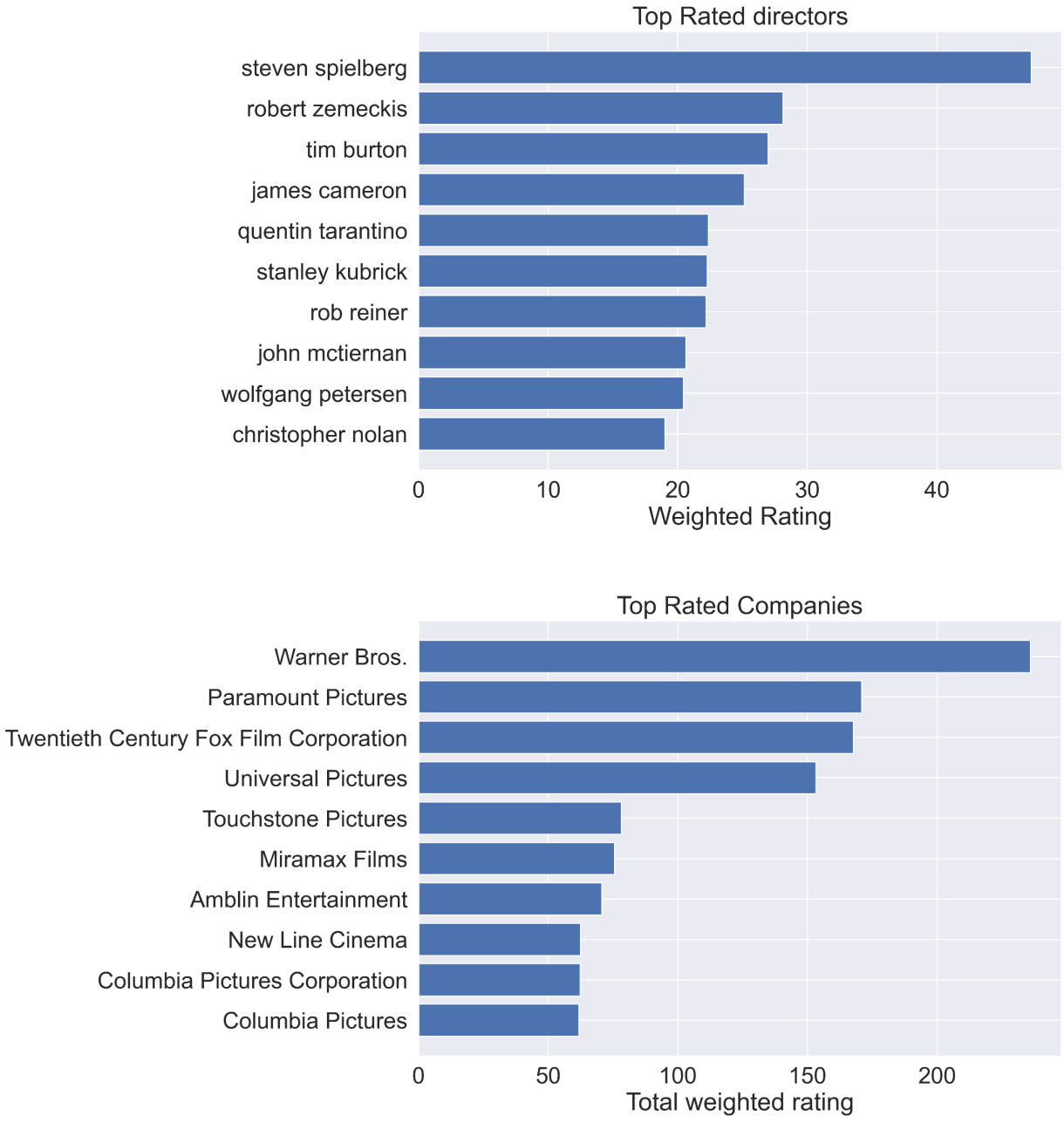
- *Production_companies*/id - list of production companies and their respective ids
- *Cast* - list of top actors/actresses
- *Cast*/*crew_size* - full cast and crew sizes
- *Director* - name of director
- *Net_profit* - Total revenue ($) minus budget ($)
- *Wr* - the weighted rating calculated by (v/(v+m) * R) + (m/(m+v) * C), where v is the number of ratings, m is the minimum number of ratings required, R is the average rating of the movie, and C is the average rating of the entire dataset
- *Soup* - a collection of strings that describe the movie. This includes all of the string features mentioned above, and is only used in the content-based filter.

We have a total of 4558 unique movies, 83000 users, and 15 million ratings in our dataset. I will be using this for Exploratory Data Analysis, while the larger ratings table (over 9000 movies), which contains less metadata, will be used for the recommender system.

Exploratory Data Analysis

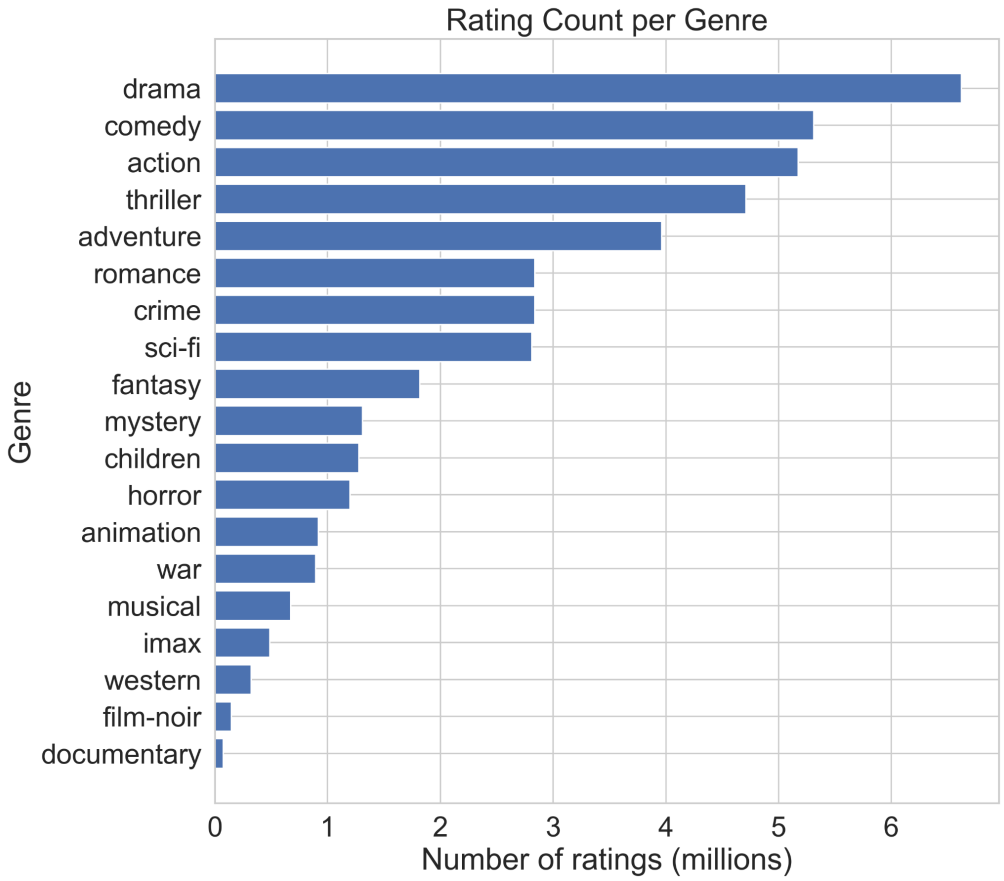
Because we are dealing with ratings for our recommender, I wanted to focus more on rating count and rating averages for the initial analysis. Rather than use the raw average rating or tmdb's popularity rating, I used an IMDB weighted rating formula to find the most popular movies in the dataset. It essentially normalizes the popularity so that a movie with only one rating of 5 isn't "better" than another movie with 30 ratings of 4.5. Let's take a look at the best movies and directors of our dataset.

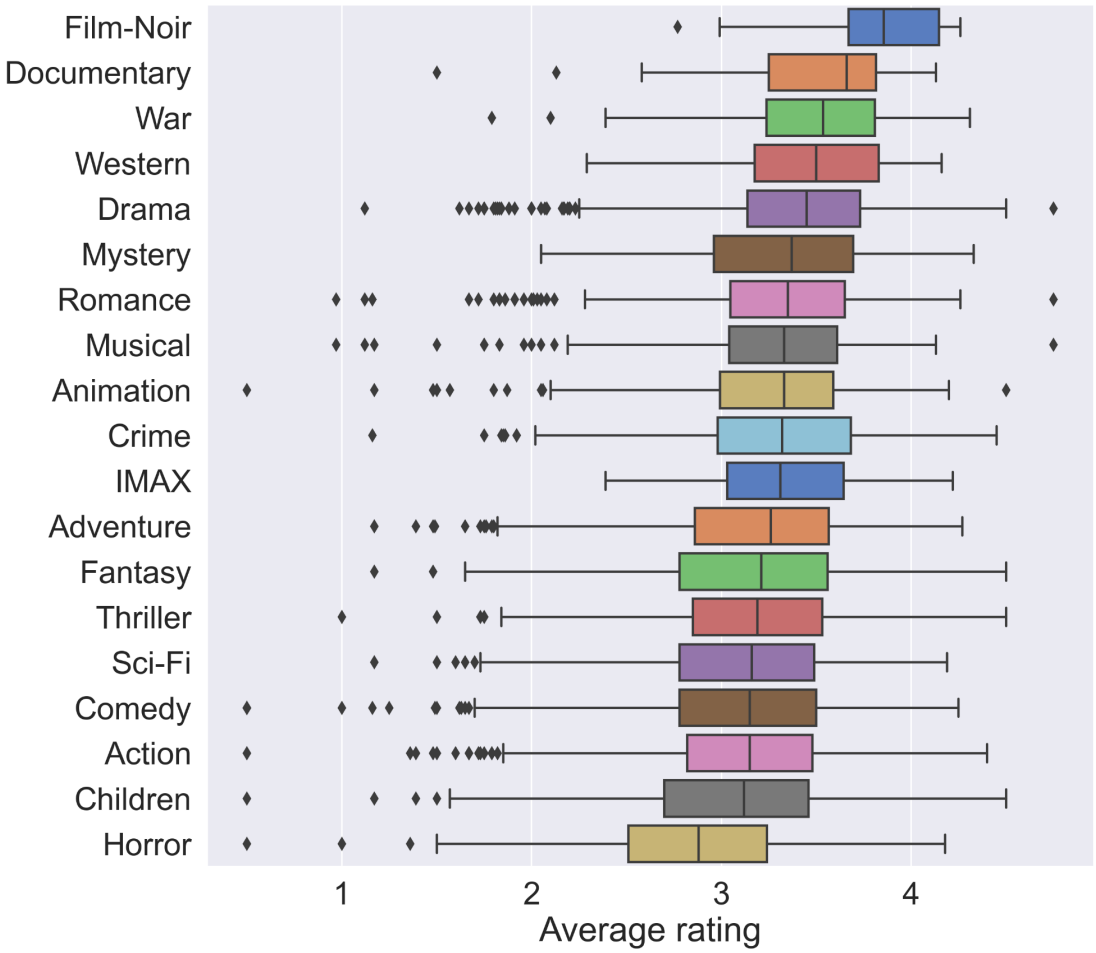




***Figures 1, 2, and 3**: The Shawshank Redemption, Steven Spielberg, and Warner Brothers top the ratings charts.*

Next, we will break down the movies by genre to see if we can uncover any interesting trends. We won't be using the weighted rating feature mentioned above since we are only concerned with the raw data.


Rating Count per Genre
drama
comedy
action
thriller
adventure
romance
crime
sci-fi
fantasy
mystery
children
horror
animation
war
musical
imax
western
film-noir
documentary
Genre
0
1
2
3
4
5
6
Number of ratings (millions)



***Figures 4 and 5**: Rating counts (figure 4) and rating averages boxplot (figure 5) broken down by genre.*

We see that although drama, action, and comedy are the most popular genres, they do not necessarily get the highest ratings. In fact, the average ratings for the most popular genres are much lower (0.5 less on average) than two highest rated genres, Film-Noir and documentaries. Film-Noir and documentaries are also the least popular genres in terms of rating count.

Looking into the distributions for the rating averages and counts yields the histograms below:

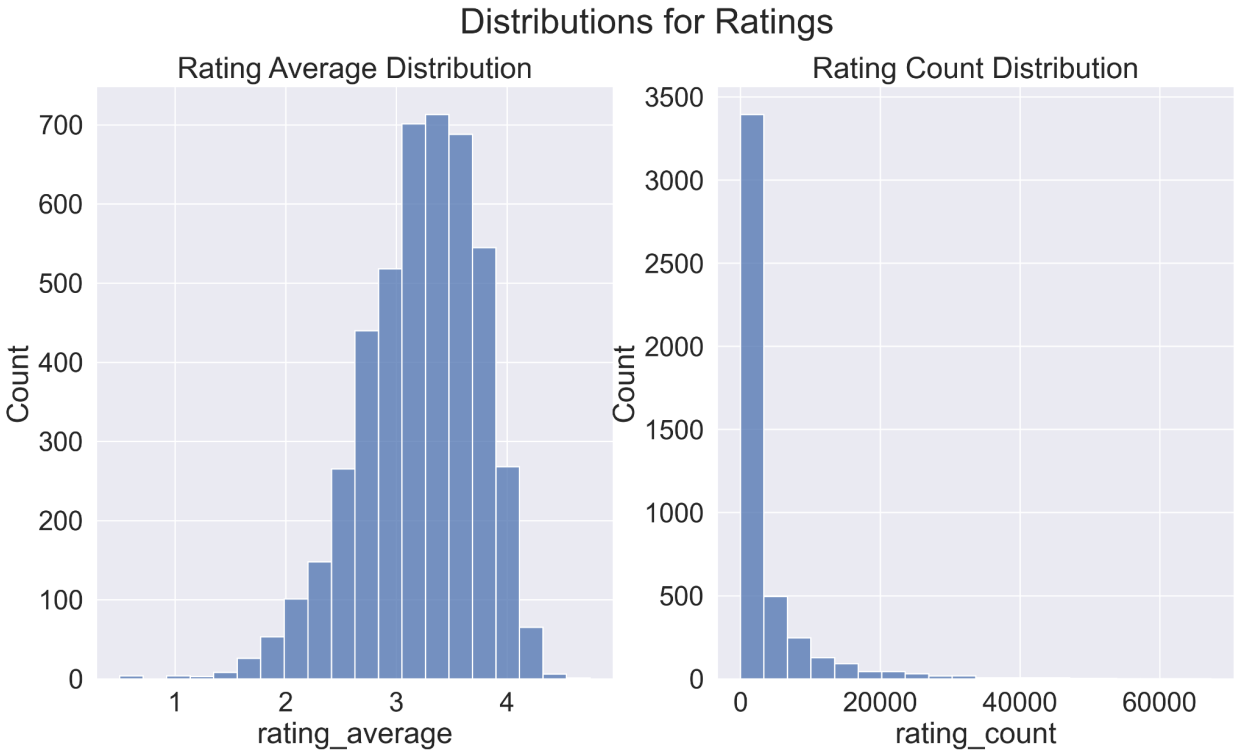


***Figure 6**: Rating average and rating count histograms.*

We see that the average rating for the movies in our dataset is around 3.5, and have been rated by 3300 unique users. A very small percentage of movies (approximately 0.1%) are rated higher than 4.5 or less than 1. In addition, a large percentage of movies (approximately 75%) have been rated 3300 times or fewer.

Since we have the release years of the movies, we can also see if we find any interesting trends that have happened over the years.

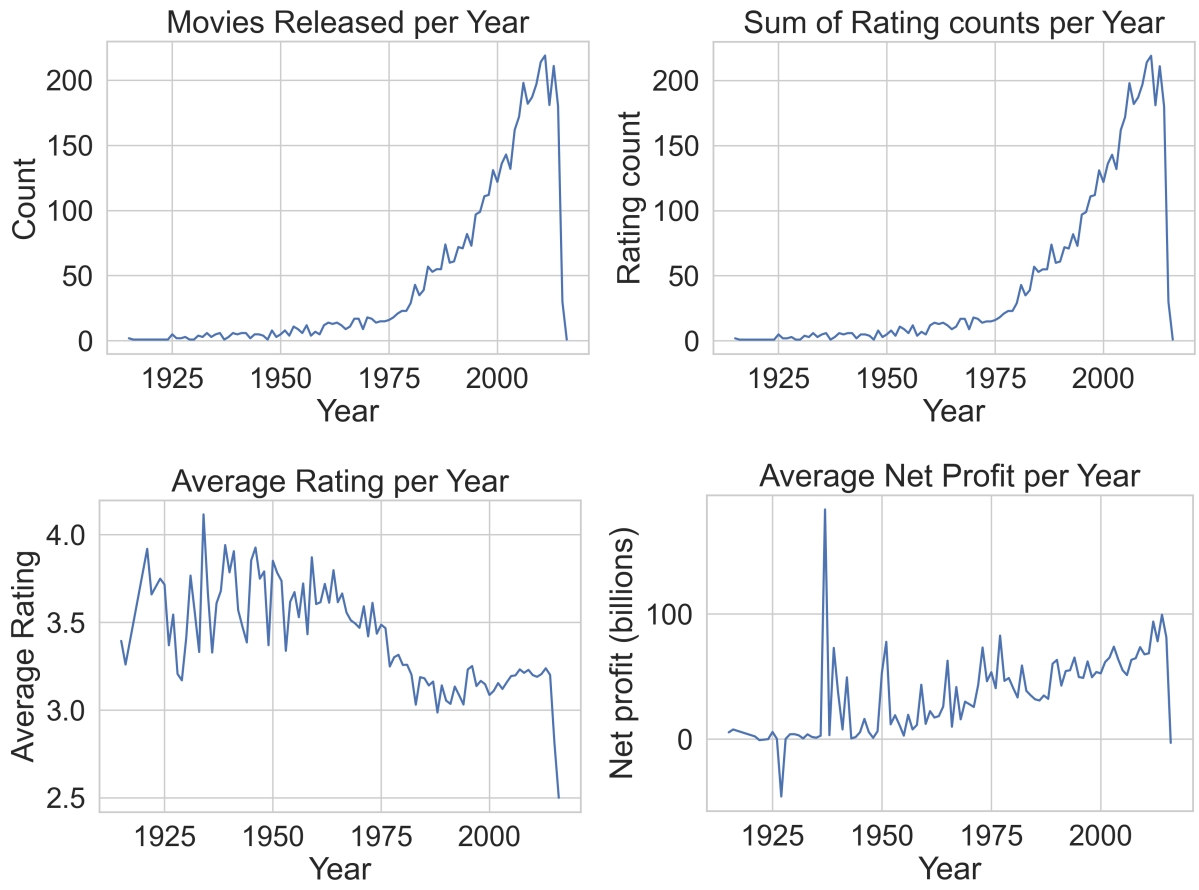


***Figure 7**: Feature analysis by year since 1915.*

The sum of average ratings per year and movies released per year are almost identical, which makes sense considering that the more movies there are, the more ratings there will be. But surprisingly, the average rating has actually gone down since the 1950s. This could be due to a couple reasons: users may have gotten more critical of movies over time, or the quality of movies may have suffered in recent years. The massive drops at the end are due to the low number of recent movies in the dataset (the most recent being released in 2016).

Content-Based vs Collaborative-Based Filtering

Content-based filtering is a simple, yet effective type of recommendation system that recommends items that are similar to other items the user has liked in the past. The similarity between items is determined based on their descriptions, and does not require other user ratings to be effective. I experimented with this method first as it was relatively simple to code. First, I combined the tags, cast, director, and genre strings for each movie into a new ‘soup’ feature that basically acts as the movie description. I fit a

CountVectorizer on this new feature and calculated the similarity scores using cosine similarity. Lastly, I used a function[3] to retrieve the top 10 most similar movies based on the inputted title. Here are the recommendations for one of my favorite movies of all time, *The Count of Monte Cristo (2002):*

Journey 2: The Mysterious Island (2012)
Patriot Games (1992)
The China Syndrome (1979)
Death Defying Acts (2007)
Free Willy (1993)
War of the Worlds (2005)
The Guns of Navarone (1961)
Metro (1997)
Starter for 10 (2006)
Major Dundee (1965)

Though these recommendations are great, there are some obvious limitations to the method. Perhaps the biggest drawback is its limited scope - the algorithm only considers item description, which might lead to a set of narrow (and possibly low quality) recommendations. Another problem is that the collection could be biased towards a specific genre, known as the "gray-sheep" problem[4].

I tried a collaborative-based filter that would take in more user-input next to see if it would generate better recommendations. First, I created a user-item matrix with the original raw ratings, and scaled the ratings by user using a Standard Scaler. This standardizes the ratings so an average rater is on the same scale as a harsh critic, who tends to give movies much lower ratings. After calculating the cosine similarities in the item-to-item matrix, I was able to get a list of movies based on my favorite movies, which includes *The Count of Monte Cristo*. The main problem with this approach was the slow computational speed, especially when taking in new inputs. The item-to-item matrix was extremely large in size, and when adding in a new user's ratings, another similarity matrix had to be calculated. Luckily, there are factorization techniques that help deal with matrices like this one.

The dataset's high dimensionality and high sparsity (97.5%) made it perfect for Singular Value Decomposition, or SVD. SVD is a matrix factorization technique that breaks down the original matrix into three matrices to reduce its dimensionality while

[3] Get recommendation function credited to Ibtesam Ahmed and her Kaggle post on https://www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system

[4] Source: Herrada, Oscar Celma. "The Recommendation Problem." *Music Recommendation and Discovery in the Long Tail,* 2008, pp.37

preserving important information. I chose to use Truncated SVD (another form of SVD that outputs specific values) for the recommender, and after using GridSearch to find the optimal number of latent factors, *k,* I set k equal to 10. The latent factors are the number of features associated with the rating that are not explicitly stated in the ratings data, such as directors, actors, etc. I fed the model my three favorite movies, with the following results:

Lord of the Rings: The Fellowship of the Ring (2001)
Lord of the Rings: The Two Towers, The (2002)
Matrix, The (1999)
Pirates of the Caribbean: The Curse of the Black Pearl (2003)
Gladiator (2000)
Dark Knight, The (2008)
Shrek (2001)
Finding Nemo (2003)
Batman Begins (2005)
Incredibles, The (2004)

RMSE: 1.029

The movies generated were more in line with my favorite genres, which are action and fantasy. Though the algorithm recommended great movies and does not suffer from the same drawbacks as the content-based filter, the inherent problem here is that all of its recommendations are basically blockbuster movies - otherwise known as “popularity bias.” To address this, I tested several standardization techniques that would help the website to recommend more novel movies.

It is important to note that in addition to popularity bias, content and collaborative filters both suffer from the “cold-start” problem (the system has no information for new users) and the gray-sheep problem mentioned above. The website reduces both problems by giving users a quick questionnaire in order to ‘learn’ about their preferences.

Standardization

In addition to scaling ratings by user, I tried two other methods which penalized popularity by dividing each movie by its number of ratings. It generated these results using my same three movies:

Gentlemen of Fortune (Dzhentlmeny udachi) (1972)
Here Comes the Boom (2012)
Mouchette (1967)

Great Buck Howard, The (2008)
Swing Vote (2008)
Relax... It's Just Sex (1998)
American Me (1992)
We Bought a Zoo (2011)
Morvern Callar (2002)
Bob le Flambeur (1955)

RMSE: 0.012

Despite the lower Root-Mean-Squared-Error and ability to recommend lesser-known movies, this method did not recommend any that I found appealing. Method #3 takes a similar approach, but divides each movie by the square root of rating count rather than just the raw rating count.

Gentlemen of Fortune (Dzhentlmeny udachi) (1972)
24 Hour Party People (2002)
Great Buck Howard, The (2008)
My Favorite Wife (1940)
Amarcord (1973)
Songs From the Second Floor (Sånger från andra våningen) (2000)
American Me (1992)
Swing Vote (2008)
Bob le Flambeur (1955)
Dark Blue (2003)

RMSE: 0.074

Not surprisingly, this movie set is very similar to the one generated using division by rating count - namely, half of the recommended movies were the same as those listed in method#2. Although this definitely recommended more novel movies, I still preferred the movies generated from the filter that scaled ratings by user.

Deployment and Conclusion

After experimenting with different filtering methods and techniques, I decided to deploy the collaborative filter recommender because it generated what I believed to be the best recommendations. The website can be summarized in three main steps:

1. The user is shown a list of ten movies and is prompted to select one of them, or none of them. The movies that are chosen are given a rating of 5, and the movies that are not chosen are given a rating of 0.5.
2. This will repeat until the user has chosen three movies.
3. The website combines the new user input with the original dataset and generates the top 10 movies using TruncatedSVD

By prompting users with a questionnaire, the website learns user preferences and applies it to the algorithm to reduce the cold-start problem.

Still, the website does have its drawbacks and can be improved in several ways. First, the formula that automatically assigns ratings based on what the user chooses will, in most cases, not be accurate. The user may have actually rated the other non-chosen movies higher, and vice versa. A possible solution is implement a Netflix or tmdb login so that the website will already have a user's existing ratings.

There are also many other filtering methods, including a combination of content and collaborative filters. Other methods may be more efficient and provide even better recommendations. Another way of improving on the website would be to survey users for feedback on the quality of recommendations.

Lastly, the popularity bias wasn't solved in this project. In the future, I would like to do more research on it, and apply it to the website so that it recommends novel movies that users would likely enjoy.

***Figure 8**: Full website can be found on marshalllee.pythonanywhere.com*

References

1. **Dataset -** F. Maxwell Harper and Joseph A. Konstan. *The MovieLens Dataset: History and context.* ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

2. Thompson, Clive. *If You LIked This, You're Sure to Love That*. November 21, 2008. NY Times

3. Herrada, Oscar Celma. "The Recommendation Problem." *Music Recommendation and Discovery in the Long Tail,* 2008, pp.37

4. Get recommendation function for content filter credited to Ibtesam Ahmed and her post on Kaggle