Comparison of Content Based-Filtering and Collaborative Filtering in Movie Recommendation Systems

Mack Stancil
Department of Computer Science
University of South Carolina Upstate
Spartanburg, South Carolina, USA
mstancil@email.uscupstate.edu

ABSTRACT

Today it is uncommon for a website not to have some kind of recommendation system. Typically, these recommendation systems are built using AI or machine learning. Experience with machine learning and AI have become some of the most sought-after skills in the STEM field. In this paper I will be discussing and comparing two popular machine learning models used in recommendation systems. This project is being used to determine which of these machine learning methods is more efficient. The first method is content based-filtering which uses attributes from movies such as the genre, director, and cast which is then put through the cosine similarity algorithm to find the best recommendations. The other method is collaborative filtering. Instead of using attributes from the movies, this method uses data from users. Specifically, the ratings they give to movies they've seen. Then this data is put into the matrix factorization algorithm to make predictions on what a user would rate a movie. These methods were implemented using the python programming language in a google colab notebook. Through testing, it was recorded that both methods worked well in their own ways, but each method had some flaws. When it comes to efficiency, it was determined that the content based-filtering system was more efficient. It was approximately 9.17 times faster than the collaborative filtering method when running tests. However, even though the content based-filtering method was faster, it had some poor recommendations. It was determined that the best recommendation system would be to combine both methods to make a hybrid system. With this, the cosine similarity algorithm could narrow down the movies to be input into the matrix factorization algorithm. This would save time, and give the most accurate results.

Keywords

ML, Machine Learning, AI, Content-Based Filtering, Collaborative Filtering, Recommendation System, Matrix Factorization, Data Science, Cosine Similarity, SVD, Single Value Decomposition, RMSE, Root Mean Square Error,

1. INTRODUCTION

Data science, machine learning, and AI have all become increasingly more popular and implemented into so many major websites and programs. I've chosen to do this project because I wanted to gain some experience and insight into the process of data science. This is a field that I'm especially interested in pursuing. Not only because it's an interesting process, but it's growing at an exponential rate, so it's a field in the STEM industry that will be around for many years.

Recommendation systems have steadily become favored in nearly every industry. You see these systems used in shopping on websites such as Amazon for example. This is a fantastic use of machine learning and recommendation systems to drive more sales. Which is why so many companies are using these methods. Next time you shop on Amazon take a look around the website and you'll notice that they recommend items that are frequently bought together, or they recommend items based on your previous purchases.

That was just an example of how machine learning and recommendation systems are used in industry. When deciding on this project I had to determine which type of content to use for my recommendation system to make and I chose movies. The entertainment industry is huge. There is a growing number of streaming services out there, nearly all of which use a recommendation system. These recommendation systems have been around for quite a long time. However, they weren't implemented the same way that they are now. Years ago, people had to read TV guides and go to a store to rent movies and get recommendations from the employees. Now we have streaming services that have massive databases full of movies and shows. It can be extremely difficult and frustrating to go through these databases and find movies that we might want to watch. So having a recommendation system can make the user experience vastly more enjoyable.

2. LITERATURE REVIEW

Given the overwhelming use of recommendation systems today. There is an abundance of articles and research papers discussing recommendation systems and the different types of filtration methods. You can find information on recommendation systems going all the way back to 1979 when the first iteration was introduced by the computer librarian Grundy [10]. A journal entry titled, "A Brief Introduction of Recommender Systems" gives a fantastic overview of recommendation systems and how they were introduced. For example, according to this article, the earliest versions of recommendation systems used collaborative based filtering methods. Specifically, the user-user based collaborative filtering [10]. This article also goes into detail about different types of models used in recommendation systems. These include, linear models, low rank models, neural models, and causality inspired methods. Like stated previously, there are many scholarly articles and research papers with fantastic information about recommendation systems.

3. METHODOLOGY

3.1 Content Based-Filtering

Content-based filtering makes recommendations based on similarities in varying attributes of the content being analyzed. Some attributes to consider when designing a movie recommendation system include genre, director, actors, rating, description, etc. The basic idea for this method is that if a user likes a particular movie, they would like another movie that has similar attributes. The algorithm used for this method is known as cosine similarity. This is the first algorithm I used for comparison. Cosine similarity is used to measure the difference between two non-zero vectors of an inner product space [5]. Basically, this just means that the cosine similarity will measure the similarity between these two vectors. In this case, cosine similarity will be comparing how similar a user's movie preference is compared to other movies in the database.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Equation 1 Cosine Similarity [9]

This is an equation used to calculate cosine similarity. In this case, A would be the movie that the user has already watched, and B would be the movie that it is being compared to find similarities.

3.2 Collaborative Filtering

The collaborative filtering method is different than the content-based filtering method, because with this method you can compare multiple users' data, in this case movie ratings, and make predictions on what rating a user might give a movie. Depending on the method of implementation, you might not even need to have the metadata that you would typically use with a content based-filtering system. For this method you need to implement a matrix factorization-based algorithm. Matrix factorization is used to multiply varying entities to find latent features [2]. In this case, the two entities being multiplied are the users, and the ratings they gave to movies. The algorithm is looking for users that both watched the same movie and gave it a similar rating. Below is an example of what one of these matrices would look like.

	Movie1	Movie2	Movie3	Movie4	Movie5
U1		5	4	2	1
U2	1			5	3
U3	1	4	4	1	
U4			2		2
U5	3	1	1		

Figure 1 User/Movie Rating Matrix [2]

For example, let's say we want to determine if user one(U1) would enjoy Movie1. The algorithm would look at other users who gave similar ratings to movies that user one has viewed then make a prediction based on what those other users rated Movie1. This method is also known as non-negative matrix factorization (NMF) because a zero would be placed in any empty space. There are no negative numbers implemented [7]. Another method implemented is single value decomposition (SVD). This is used to help with scalability and sparsity issues. Simply put, this improves the algorithm's ability to evaluate similarities between users and items which returns better latent features [7]. I will also be using the root mean square error metric (RMSE). This metric will allow us to determine the performance of the algorithm. The lower the RMSE, the better.

4. EXPERIMENTAL SETUP

4.1 Technology

I decided to implement my code using the python programming language. Everything I saw on making a recommendation system used some kind of notebook software. Based on recommendations I decided to use google colab. The only issues I ran into while using google colab was that I had to save my files directly to google drive or they wouldn't save outside of a specific runtime.

4.2 Datasets

I obtained my datasets from Kaggle. I started my implementation by building the content based-filtration system. So, the first dataset I implemented was the TMDB 5000 Movie Dataset. This dataset contained two files. The first file was the movies excel sheet which contained twenty data types, but the first ten were: budget, genre, homepage, id, keywords, original_language, original_title, overview, popularity, production_companies. The second file was the credits excel sheet. This file contained four data types: movie_id, title, cast, crew. I started implementing these datasets by cleaning the data and making a new dataset by merging them on the 'id' column which both datasets contained and allowed the data to merge smoothly. Below is a small example of what the part of the cleaned dataset looked like.

	budget	genres	homepage	id	keywords	original_language	original_title	overview
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...
4	260000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://movies.disney.com/john-carter	49529	[{"id": 818, "name": "based on novel"}, {"id":...	en	John Carter	John Carter is a war-weary, former military ca...

Figure 2 Cleaned Movie/Credits Dataset

I cleaned the data even further for use with the cosine similarity algorithm which will be demonstrated in the next section. The other dataset used was also from Kaggle, The Movies Dataset. This dataset is larger than the previous and contains seven separate files. However, I only implemented one of these files titled, 'ratings_small'. This file contained data which showed a large group of users, as well as ratings for movies that they've watched. Below is a small sample of this dataset.

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205

Figure 3 ratings_small dataset

This dataset was used for the collaborative filtering method to compare the different users ratings of movies.

4.3 Content Based-Filtration

For the content based-filtration system, I needed a few datatypes to compare between movies (cosine similarity). The first data I started with was the overview. To find relevant common words in movie overviews I had to implement tfidvectorizer which removed stop words such as 'the' or 'a'. This vectorizer was able to use cosine similarity and find similar movies based on these common words that were found in the overviews. I did an initial test with just the overview check, and these were the results when running 'The Dark Knight Rises'.

```
65                                  The Dark Knight
299                                  Batman Forever
428                                  Batman Returns
1359                                        Batman
3854    Batman: The Dark Knight Returns, Part 2
119                                  Batman Begins
2507                                     Slow Burn
9             Batman v Superman: Dawn of Justice
1181                                          JFK
210                                  Batman & Robin
```

Figure 4 Overview Cosine Similarity

When looking at these initial results, the movies that were suggested are similar. Eight of the ten suggestions are batman movies because the overview of each of these movies mentions batman multiple times, which the algorithm would have noticed. In order to broaden the results, I needed to add some different data types. I implemented the credits, genre, and keywords data. The credits included the cast, crew, and director. I created a function called 'soup' which combined each of these data types and put them into a matrix called 'count_matrix'. Then the count matrices were input and ran through the cosine similarity function to return results. Below is an example output using the same movie as before 'The Dark Knight Rises' while using the new metadata.

```
65                  The Dark Knight
119                   Batman Begins
4638    Amidst the Devil's Wings
1196                The Prestige
3073          Romeo Is Bleeding
3326             Black November
1503                      Takers
1986                      Faster
303                     Catwoman
747               Gangster Squad
```

Figure 5 Example Metadata Cosine Similarity

When comparing this version of the cosine similarity method, you can see that there is a much broader selection of movies suggested. Obviously, there are still a couple Batman movies included, but there are also other movies with a similar theme of action/crime. For example, 'Amidst the Devil's Wings' is about a wealthy man who makes it his life's purpose to get revenge on the men responsible for his fathers' death. This has some very similar plot points to 'The Dark Knight Rises'.

4.4 Collaborative Filtering

For the collaborative filtering method, I used the 'ratings_small' dataset which contained different user's ratings on movies they've watched. I started by building the trainset which contained the svd and data. As well as the RMSE and MAE measurements. Using this new trainset, the algorithm is able to make predictions on what a user would rate a specific movie based on that user's previous movie ratings. For my initial testing, I simply used user 1's data, some of which can be seen below.

	userId	movieId	rating	timestamp
0	1	31	2.5	1260759144
1	1	1029	3.0	1260759179
2	1	1061	3.0	1260759182
3	1	1129	2.0	1260759185
4	1	1172	4.0	1260759205
5	1	1263	2.0	1260759151
6	1	1287	2.0	1260759187
7	1	1293	2.0	1260759148
8	1	1339	3.5	1260759125
9	1	1343	2.0	1260759131
10	1	1371	2.5	1260759135

Figure 6 User 1's Data

Now when I was just doing the initial testing, I only checked the predicted rating of an individual movie to ensure that the algorithm was working. Below is an example.

```
svd.predict(1, 302, 3)
```

```
Prediction(uid=1, iid=302, r_ui=3, est=2.6384894810906263
```

Figure 7 User 1's Predicted Rating of Movie 302

As you can see in the figure, I ran the algorithm to find User 1's predicted rating of movie 302. The prediction can be seen from the 'est' attribute. So in this case it is predicted that User 1 would give movie 302 a rating of 2.638. Now, because of how the collaborative filtering method works, we don't see what the actual movie titles are. With this method, we don't have to take any other attributes other than the Users and the ratings for the algorithm to make its predictions. After determining that the algorithm worked, I decided to add code that would check all of the movies and find the top ten highest rating predictions for a User. Below is a figure of the initial test of this new code.

```
Prediction(uid=1, iid=1172, r_ui=3, est=3.692365542827942,
Prediction(uid=1, iid=296, r_ui=3, est=3.660436757445102,
Prediction(uid=1, iid=318, r_ui=3, est=3.6158280721883225,
Prediction(uid=1, iid=926, r_ui=3, est=3.571640051891815,
Prediction(uid=1, iid=922, r_ui=3, est=3.5651667818744435,
Prediction(uid=1, iid=904, r_ui=3, est=3.560279824018852,
Prediction(uid=1, iid=3462, r_ui=3, est=3.5404572576927476,
Prediction(uid=1, iid=1196, r_ui=3, est=3.526298946117874,
Prediction(uid=1, iid=969, r_ui=3, est=3.514890602388939,
Prediction(uid=1, iid=1217, r_ui=3, est=3.514415619372767,
```

Figure 8 Initial Collaborative Filtering Test

The code went through all of the movies, found the prediction and input that prediction into an array, then the array was sorted by the predicted rating. As you can see, the top ten highest predicted ratings are shown.

5. RESULTS

5.1 Content Based-Filtration

For final testing of the content based-filtration method, I simply tested ten different movie titles to see what the recommended movies would be. In general, I would say that this method worked well. However, I did notice some major flaws. One flaw I noticed was that for each time I tested the system, it would give one or two recommendations that weren't related to the original movie whatsoever. For example, in test 5, seen in the figure below.

```
get_recommendations('Iron Man')
```

```
79                    Iron Man 2
31                    Iron Man 3
1868          Cradle 2 the Grave
7          Avengers: Age of Ultron
538                      Hostage
119                Batman Begins
1290     Baahubali: The Beginning
4574                     Roadside
2044           The Little Vampire
2633    The Clan of the Cave Bear
```

Figure 9 Content Based-Filtering Test 5

In this example, the algorithm was finding recommendations for the original 'Iron Man' movie. As you can see, the results include other Iron Man movies as well as other super hero movies. However, it also includes 'The Little Vampire' and 'Baahubali: The Beginning'. When looking at the plots of these movies you can see that they are simlar in some ways which explains why these movies were suggested. Even so, I don't believe these movies have enough similarities for them to be a good recommendation for 'Iron Man'. Another flaw I noticed was that the system doesn't take movie ratings into account such as PG, PG-13, and R. This is a significant issue because this could recommend rated R movies for a child who just watched a cartoon. For example, in test 6, shown below.

```
get_recommendations('Jurassic Park')
```

```
28                    Jurassic World
2527    National Lampoon's Vacation
1580                     The Nut Job
479          Walking With Dinosaurs
2805           The Land Before Time
1983                Meet the Deedles
3577              The Way Way Back
54                 The Good Dinosaur
3139                Adventureland
1536                      Vacation
```

Figure 10 Content Based-Filtering Test 6

AS you can see in the figure, the algorithm is giving recommendations for the first 'Jurassic Park' movie. In the results you can see multiple movies related to dinosaurs, as well as movies related to theme parks. The main issue here is that

'Jurassic Park' is a film meant for adults, but one of the recommendations is 'The Land Before Time' which is a children's cartoon. I would say that most people that are watching 'Jurassic Park' probably wouldn't want to watch a movie like 'The Land Before Time'.

5.2 Collaborative Filtering

Unlike the content based-filtering system, the collaborative filtering system works by comparing a user's ratings of movies they've watched. So, instead of finding recommendations for ten different movies, this system found recommendations for ten different users. This method also worked fairly well, but similarly to the content based-filtering method, it had some flaws. The first flaw, like previously stated, is that this method doesn't take attributes of the movies into account. Because of this, the algorithm has to run through every movie in the database, make the predictions where possible, then find the movies with the highest predicted rating. This is a problem because it decreases the efficiency of the system. Especially if you consider scalability issues. As the database increases the efficiency will decrease. The other flaw with this system is that there is a chance that there aren't enough similarities between user ratings for some users to get strong recommendations. This can be seen in test 10 in the figure below.

```
Prediction(uid=12, iid=2804, r_ui=3, est=3.9001649002450747,
Prediction(uid=12, iid=904, r_ui=3, est=3.8931270546340695,
Prediction(uid=12, iid=1172, r_ui=3, est=3.8389198996989973,
Prediction(uid=12, iid=915, r_ui=3, est=3.8087874179148336,
Prediction(uid=12, iid=1204, r_ui=3, est=3.80837442275055, d
Prediction(uid=12, iid=2917, r_ui=3, est=3.807564349671726,
Prediction(uid=12, iid=2318, r_ui=3, est=3.8060715202052053,
Prediction(uid=12, iid=6016, r_ui=3, est=3.773082393039723,
Prediction(uid=12, iid=1217, r_ui=3, est=3.7501108764216453,
Prediction(uid=12, iid=1221, r_ui=3, est=3.7442750086904932,
```

Figure 11 Collaborative Filtering Test 10

As you can see in this figure, the highest rating prediction is only 3.900. This isn't a terrible rating, but considering the amount of users to compare to, there should be some similarities that would give a better rating.

5.3 Comparison

The best way to compare the two methods was execution time. For testing, I used ten different movies. Each of which, I found ten recommendations for. I did this fore both methods. Then I recorded the execution time each time I executed the code. With this information I was able to make the figure below.

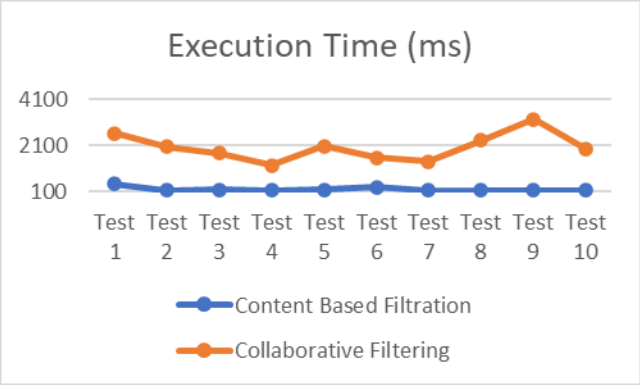

Figure 12 Execution Time Comparison

As you can see in the figure, the content based-filtration method was significantly faster than the collaborative filtering method. The average execution time for content based-filtering was 199ms while it was 2024ms for the collaborative filtering method. This means the content based-filtering method was approximately 9.17 times faster than the collaborative filtering method.

6. CONCLUSION

In conclusion, this project aimed to explore and compare the efficiency of two prominent machine learning models, namely content-based filtering and collaborative filtering, in the context of recommendation systems for movies. The significance of recommendation systems, driven by artificial intelligence and machine learning, cannot be overstated in today's digital landscape, where they are ubiquitous on various platforms. The project was conducted using the Python programming language in a Google Colab notebook, and two datasets, TMDB 5000 Movie Dataset and The Movies Dataset, were utilized for implementation.

Content-based filtering relies on attributes such as genre, director, and cast to make movie recommendations. The cosine similarity algorithm was employed to measure the similarity between movies based on these attributes. The methodology involved cleaning and preprocessing the data, implementing TFIDVectorizer for analyzing movie overviews, and incorporating additional data types like credits, genre, and keywords. While the content-based filtering method exhibited reasonable performance, some notable flaws were identified. Recommendations occasionally included unrelated movies, and the system did not account for movie ratings, potentially leading to inappropriate recommendations.

In contrast, collaborative filtering leverages user data, specifically movie ratings, to predict and recommend movies. Matrix factorization algorithms like Singular Value Decomposition (SVD) and Non-Negative Matrix Factorization (NMF) were employed for this method. The Root Mean Square Error (RMSE) metric was used to evaluate the performance of the collaborative filtering algorithm. Like content-based filtering, collaborative filtering demonstrated effectiveness but had its own set of limitations. It overlooked movie attributes, requiring the algorithm to iterate through the entire movie database for predictions, resulting in longer execution times.

Efficiency was a key criterion for comparison between the two methods. Through rigorous testing, it was determined that content-based filtering outperformed collaborative filtering in terms of execution time. On average, content-based filtering was approximately 9.17 times faster than collaborative filtering. However, the speed advantage of content-based filtering came at the cost of some suboptimal recommendations, highlighting a trade-off between efficiency and recommendation quality.

Recognizing the strengths and weaknesses of both methods, a hybrid recommendation system was proposed to capitalize on their respective advantages. The content-based filtering system could efficiently narrow down movie selections, which would then be fed into the collaborative filtering algorithm for more accurate predictions. This hybrid approach aimed to strike a balance between speed and recommendation accuracy, providing an optimal solution for movie recommendation systems.

In conclusion, the project shed light on the intricate dynamics of recommendation systems, emphasizing the need for a nuanced

approach that considers both efficiency and recommendation quality. As machine learning continues to evolve, the hybridization of diverse models may hold the key to unlocking more robust and effective recommendation systems in the future.

7. REFERENCES

[1] aishwarya.2. (2023, January 11). *Python: Implementation of movie recommender system*. GeeksforGeeks. https://www.geeksforgeeks.org/python-implementation-of-movie-recommender-system/#

[2] Chen, D. (2020, July 9). Recommendation system - matrix factorization. Medium. https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b

[3] Kaul, K. (2021, March 3). *Content based movie recommendation system*. Medium. https://medium.com/web-mining-is688-spring-2021/content-based-movie-recommendation-system-72f122641eab

[4] Kniazieva, Y. (2022, April 14). *Guide to movie recommendation systems using machine learning*. High quality data annotation for Machine Learning. https://labelyourdata.com/articles/movie-recommendation-with-machine-learning

[5] Kurka, B. (2019, June 30). Building movie recommender systems using cosine similarity in python. Medium. https://medium.com/@bkexcel2014/building-movie-recommender-systems-using-cosine-similarity-in-python-eff2d4e60d24

[6] Parashar, M. (2023, June 8). *Movie recommendation system*. Kaggle. https://www.kaggle.com/datasets/parasharmanas/movie-recommendation-system?select=movies.csv

[7] Roy, A. (2020, July 31). *Introduction to Recommender systems- 1: Content-based filtering and collaborative filtering*. Medium. https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421

[8] Vaddy, S. (2020, May 10). *Collaborative filtering movie recommendation*. Medium. https://medium.com/@srinidhi14vaddy/collaborative-filtering-movie-recommendation-60461c7ef897

[9] Varun. (2020, September 27). Cosine similarity: How does it measure the similarity, maths behind and usage in Python. Medium. https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db

[10] Dong, Z., Wang, Z., Xu, J., Tang, R., & Wen, J. (n.d.). A brief history of Recommender Systems - arxiv.org. arxiv.org. https://arxiv.org/pdf/2209.01860.pdf