Movies Recommendation System

Mobina Kargar
dymamsijhidjj@gmail.com

Abstract

This capstone project integrates the major topics covered in the course exploratory data analysis (EDA), feature engineering, modeling, evaluation, and lightweight MLOps for deployment through the design of a movie recommender system using The Movies Dataset.

1 Introduction

This This project implements a full recommendation pipeline with the following stages, executed in order: data cleaning, baselines (using IMDb's Weighted Rating for popularity), content-based filtering with TF-IDF, collaborative filtering with k-NN (cosine similarity), and a hybrid model that blends CB and CF scores. For evaluation, I report metrics at K∈{10,20}—Precision@K, Recall@K, and Hit Rate@K—along with 95% confidence intervals computed via bootstrap resampling.

2 Features

The merge file that I make (I mean Movies.csv file) is composed of 17 columns and 9010 entries (The data has 9010 rows and 17 columns). We can see all 17 dimensions of our dataset by printing out the first 3 entries:

Table 1: train dataset (3 rows x 17 columns)

	id	rating	title	....	revenue	runtime	poster_path
0	2	4.5	Ariel		0.0	69.0	/gZCJZOn4l0Zj5hAxsMbxoS6CL0u.jpg
1	5	2.8125	Four Rooms		4300000.0	98.0	/eQs5hh9rxrk1m4xHsIz1w11Ngqb.jpg
3	6	3	Judgment Night		12136938.0	110.0	/lNXmgUrP6h1nD53gkFh4WDzT6RZ.jpg

We can inspect the types of feature columns:

In this part I merge all the file in once to handle everything easily

Table 2: Data columns (total 17 columns):

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9010 entries, 0 to 9009
Data columns (total 17 columns):
#   Column               Non-Null Count  Dtype
---  ------              --------------  -----
0   id                  9010 non-null   int64
1   rating              9010 non-null   float64
2   title               9010 non-null   object
3   genres              8979 non-null   object
4   overview            9010 non-null   object
5   spoken_languages    8872 non-null   object
6   cast                8921 non-null   object
7   crew                8986 non-null   object
8   vote_average        9010 non-null   float64
9   vote_count          9010 non-null   float64
10  popularity          9010 non-null   float64
11  budget              9010 non-null   int64
12  keywords            8263 non-null   object
13  production_companies  8357 non-null   object
14  revenue             9010 non-null   float64
15  runtime             9010 non-null   float64
16  poster_path         9010 non-null   object
dtypes: float64(6), int64(2), object(9)
memory usage: 1.2+ MB
```

3 EDA and Data Preparation

Keep important Column that I used in most of the task:

```
Index(['id', 'title', 'genres', 'overview', 'rating', 'spoken_languages',
       'cast', 'crew', 'vote_average', 'vote_count', 'popularity', 'budget',
       'keywords', 'production_companies', 'revenue', 'runtime',
       'poster_path'],
      dtype='object')
```

Handle missing values:

	# 0
id	0
title	0
genres	0
overview	13
rating	0
spoken_langu	0
cast	0
crew	0
vote_average	0
vote_count	0
popularity	0
budget	0
keywords	0
production_cc	0
revenue	0
runtime	0
poster_path	6

17 rows x 1 cols 50 per page

3.1 Distribution of Ratings

Total ratings: 9010

count	9010.000000
mean	3.202842
std	0.863387
min	0.500000
25%	2.750000
50%	3.250000
75%	3.750000
max	5.000000

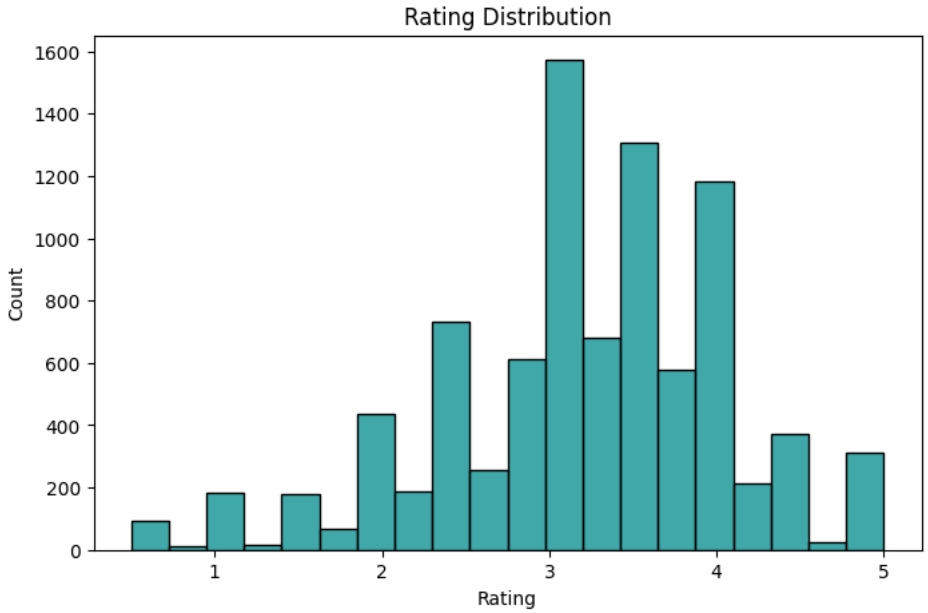


Figure1: The plot indicates that the majority of ratings are concentrated around the higher end of the scale, with the peak occurring around a rating of 4.

3.2 Long-Tail of Users and Movies

Users: 270896, mean ratings per user: 96.07

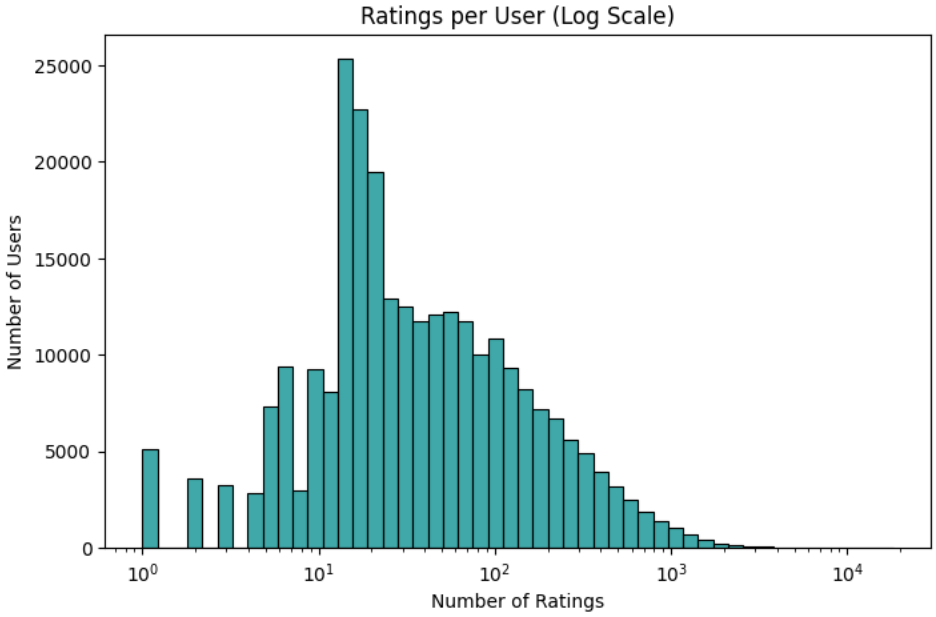


Figure2: The plot exhibits a long-tailed distribution, indicating that there are a large number of users who have provided relatively few ratings, while a smaller number of users have provided a very large number of ratings.

Movies: 45115, mean ratings per movie: 576.84

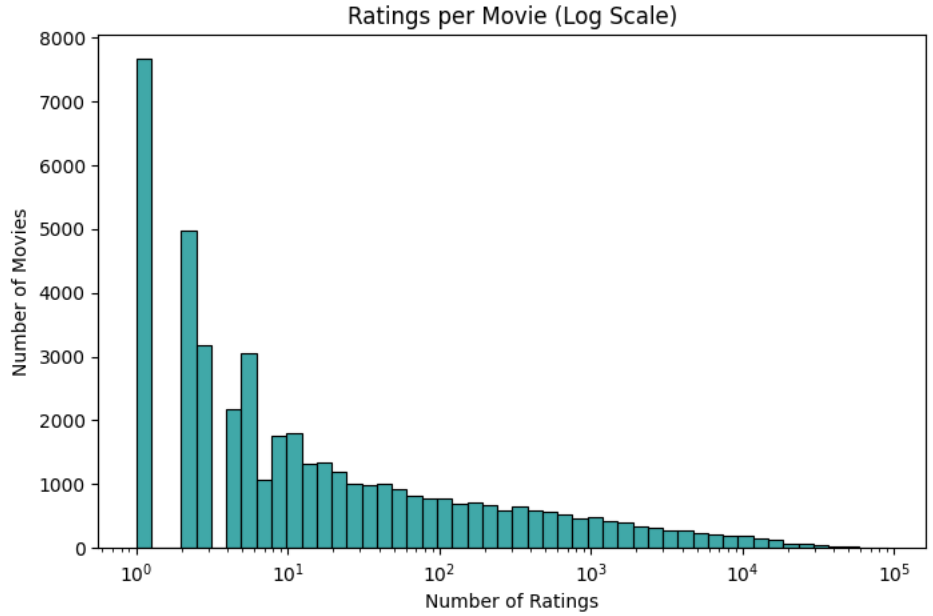


Figure3: This plot also exhibits a long-tailed distribution, indicating that there are a large number of movies that have received relatively few ratings, while a smaller number of movies have received a very large number of ratings.

3.3 Sparsity

Sparsity of user-movie ratings matrix: 0.9979

The sparsity of the user-movie ratings matrix is 0.9979, which means that the matrix is very sparse. This indicates that the majority of the possible user-movie rating combinations are missing or unobserved.

The high sparsity of the user-movie ratings matrix suggests that the dataset is characterized by a long-tail distribution, where a small number of users have rated a large number of movies, while the majority of users have rated only a few movies. This sparsity pattern is common in many real-world recommender system datasets and can pose challenges for building effective recommendation models.

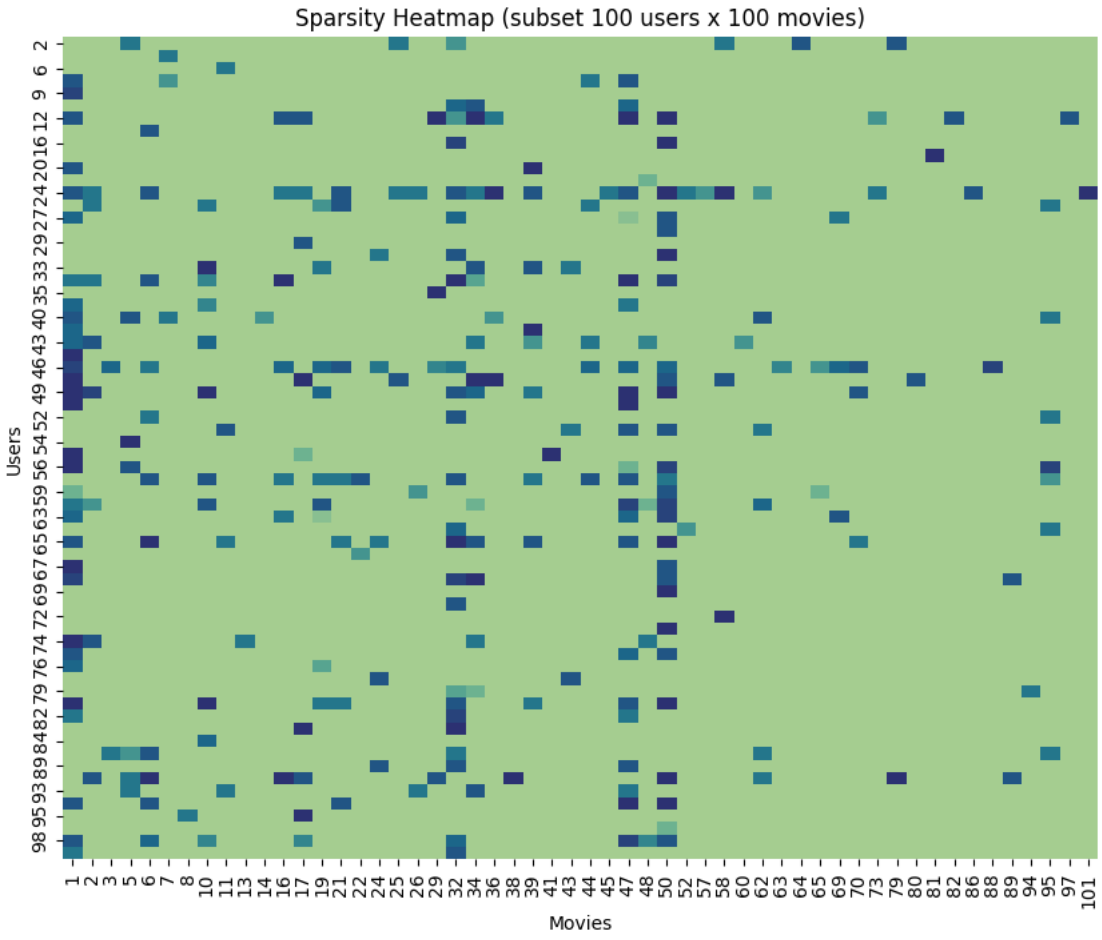


Figure4: The green cells represent the presence of a rating, while the blue cells represent the absence of a rating. The heatmap exhibits a highly scattered and sparse pattern, with only a few concentrated areas of green cells, indicating that most user-movie rating combinations are missing.

3.4 Temporal Dynamics

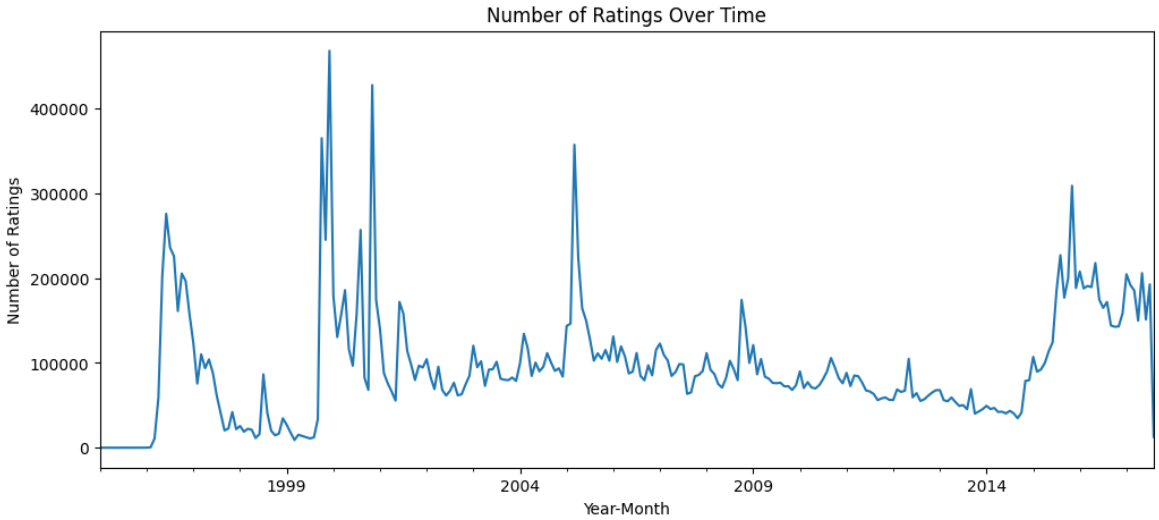


Figure5: The fluctuations in the number of ratings suggest that there are temporal variations in the user engagement and movie rating activities within the dataset.

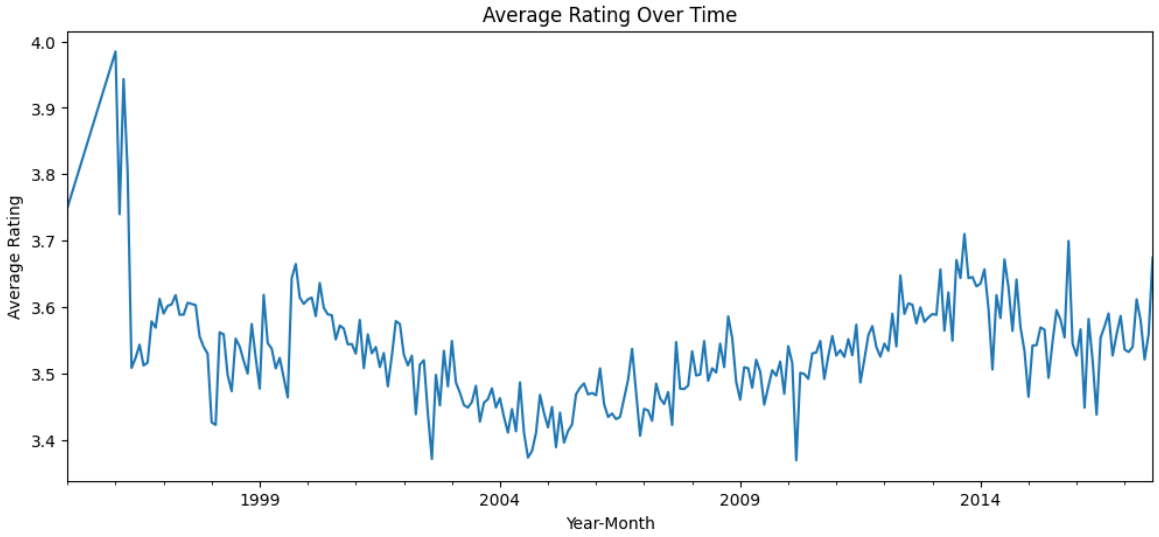


Figure6: The plot says that the overall quality perception of the movies in the dataset, as reflected by the average rating, has remained relatively consistent over the years, but the minor variations in the average rating may indicate changes in user preferences, movie quality, or other factors that influence the rating behavior.

4 Baselines

$$\mathrm{WR}(i) = \frac{v_i}{v_i + m} R_i + \frac{m}{v_i + m} C,$$

So in this part I find out the weighted rating formula to find popularity recommender by using IMDb's.
Like before I delete useless column that there is no need in my project:

	title	# vote_count	# vote_average	# score	# popularity
166	The Godfather	6024.0	8.5	9044.032626931985	41.109264
200	The Shawshank Redemption	8358.0	8.5	9043.852582287676	51.645403
227	Once Upon a Time in America	1104.0	8.3	8925.505905620497	32.182851
351	Psycho	2405.0	8.3	8923.602813426365	36.826309
337	One Flew Over the Cuckoo's Nest	3001.0	8.3	8923.282089124046	35.529554
168	The Godfather: Part II	3418.0	8.3	8923.12419546595	36.629307
436	Life Is Beautiful	3643.0	8.3	8923.05401472098	39.39497
80	Spirited Away	3968.0	8.3	8922.966693618211	41.048867
8732	Whiplash	4376.0	8.3	8922.875434163023	64.29999
288	Schindler's List	4436.0	8.3	8922.863429524563	41.725123

10 rows x 5 cols 10 per page « ‹ Page 1 of 1 › »

Let me explain Popularity and Weighted rating in my project:

Popularity : This usually ranks movies simply by the number of interactions or views (number of votes, views and …).
It doesn't account for rating quality a movie with many low ratings can still be "popular."

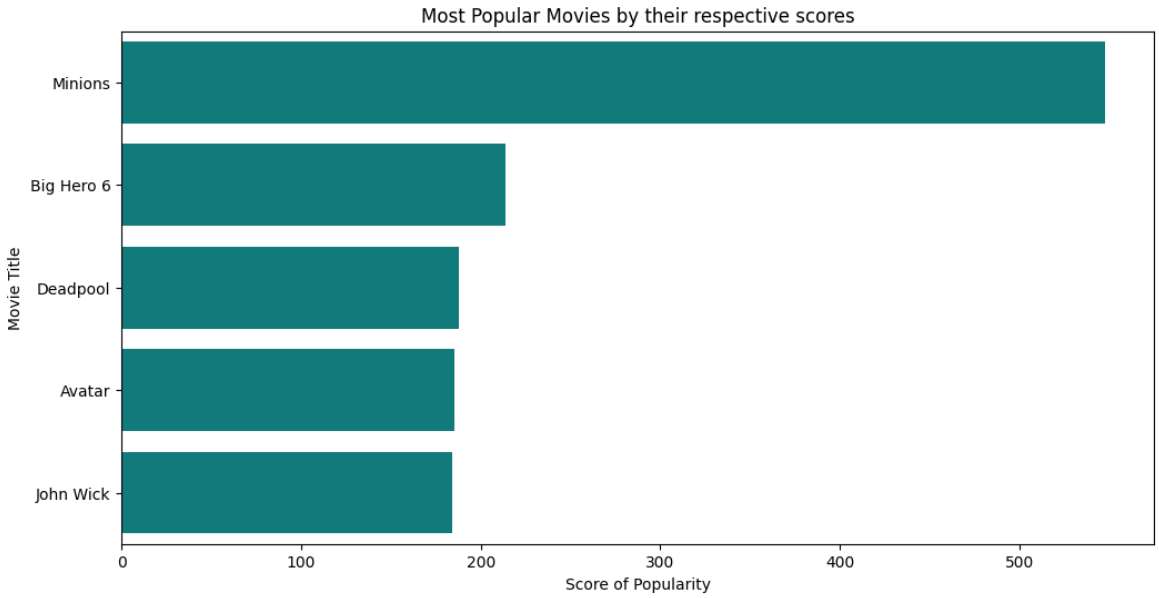


Figure7: top 5 movies by raw popularity.

Weighted Rating : This penalizes movies with very few votes and rewards movies with both high ratings and sufficient votes.

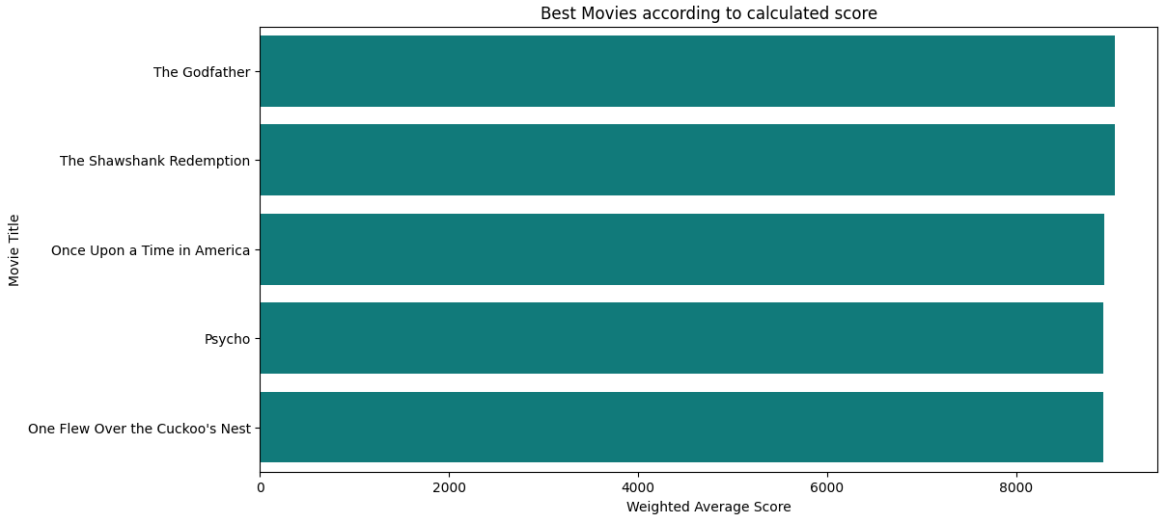
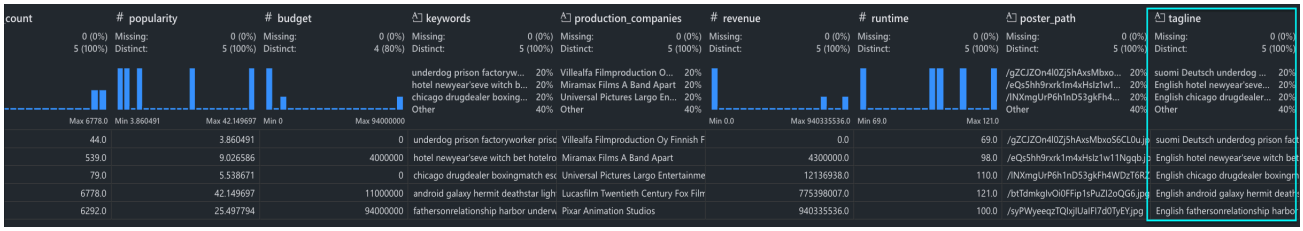


Figure8:The top 5 movies according to this balanced score, which is generally more reliable than raw popularity.

5 Content-Based Recommender

As document said I add tagline at the end of my Movies dataset:



Then drop unnecessary Column:

	id	title	genres	overview	poster_path	tagline
0	2	Ariel	Drama Crime	Taisto Kasurinen is a Finnish coal mir	/gZCJZOn4l0Zj5hAxsMbxoS6CL0u.jp	suomi Deutsch underdog prison fact
1	5	Four Rooms	Crime Comedy	It's Ted the Bellhop's first night on th	/eQs5hh9rxrk1m4xHslz1w11Ngqb.jp	English hotel newyear'seve witch bet
2	6	Judgment Night	Action Thriller Crime	While racing to a boxing match, Fran	/lNXmgUrP6h1nD53gkFh4WDzT6RZ	English chicago drugdealer boxingm
3	11	Star Wars	Adventure Action ScienceFiction	Princess Leia is captured and held hc	/btTdmkglvOi0FFip1sPuZI2oQG6.jpg	English android galaxy hermit deaths
4	12	Finding Nemo	Animation Family	Nemo, an adventurous young clown	/syPWyeeqzTQlxjlUaIFI7d0TyEYjpg	English fathersonrelationship harbor

Then add TF_IDF as document clarified(I just fitting the TF-IDF on the 'overview' text) :

<Compressed Sparse Row sparse matrix of dtype 'float64'
with 447829 stored elements and shape (7796, 31769)>

Coords	Values
(0, 27751)	0.12659794680007583
(0, 29687)	0.11557625298257457
(0, 22467)	0.1619843278690649
(0, 10402)	0.1364583950464264

(0, 22480)	0.10585196737007226
(0, 13609)	0.14739397193026327
(0, 14540)	0.06720670133790606
(0, 10456)	0.11186238069160359
(0, 11512)	0.29478794386052654
:	:
(7795, 31696)	0.21091768192789942
(7795, 3202)	0.2512050364572724
(7795, 31699)	0.21091768192789942
(7795, 765)	0.22529412320249748
(7795, 13823)	0.21573952029246238
(7795, 7983)	0.2226357153248469
(7795, 24638)	0.2512050364572724
(7795, 18164)	0.2617816390245619
(7795, 22977)	0.2848026983906117
(7795, 16399)	0.2617816390245619

Then I'm using a similarity matrix computed with the sigmoid kernel from scikit-learn:
For a summary : tfv matrix is a TF–IDF feature matrix built from movie overviews, taglines, genres, and other metadata. sigmoid_kernel is computes a similarity score between all pairs of movies, producing a matrix where each entry (i, j) represents how similar movie i is to movie j.

This approach tailors recommendations based on content similarity, making it ideal for users with some watched titles (by getting recommendations). And cold start scenarios where only genre or title preferences are known Unlike collaborative filtering, it does not require user–item interaction data and can work well for new users or new movies.

Forexample:

Genres = Action, Dram

['’Round Midnight', '*Æ*on Flux', 'xXx: State of the Union', 'xXx', 'eXistenZ', 'Zulu', 'Zorro, The Gay Blade', 'Zorba the Greek', 'Zoot Suit', 'Zombeavers']

Title = City Hall

['My Giant', 'Othello', 'Extreme Measures', 'Dracula: Dead and Loving It', 'Did You Hear About the Morgans?', 'Little Big League', 'Envy', 'Striptease', 'Misery', 'Absolute Power']

If you have no idea, don't worry cause :

['’Round Midnight', '*Æ*on Flux', '¡Three Amigos!', 'xXx: State of the Union', 'xXx', 'loudQUIETloud: A Film About the Pixies', 'eXistenZ', '[REC]', 'Zulu', 'Zorro, The Gay Blade']

Natural-language explanations:

For `Avatar` Movie:

Recommended: Dragonball Evolution
Reason: Why Recommended ? shared genres: Fantasy, Action, Adventure, ScienceFiction.

Recommended: Eragon
Reason: Why Recommended ? shared genres: Fantasy, Action, Adventure.

Recommended: Live Free or Die Hard
Reason: Why Recommended ? shared genres: Action.

Recommended: Australia
Reason: These movies are recommended based on similarity but have no direct genre or cast overlap.

Recommended: The A-Team
Reason: Why Recommended ? shared genres: Action, Adventure.

6 Collaborative Filtering (requires a GPU so I ran this in Google Colab)

In this section I merge movies_metadata and links and of course ratings to get what I need in it:

	id	imdb_id	title	movieId	imdbId	userId	rating	timestamp
0	862	tt0114709	Toy Story	1	114709	7	3.0	851866703
1	862	tt0114709	Toy Story	1	114709	9	4.0	938629179
2	862	tt0114709	Toy Story	1	114709	13	5.0	1331380058
3	862	tt0114709	Toy Story	1	114709	15	2.0	997938310
4	862	tt0114709	Toy Story	1	114709	19	3.0	855190091

Then after remove duplications , I add the total rating count :

	userId	movieId	rating	title	totalRatingCount
0	7	1	3.000	Toy Story	247
1	9	1	4.000	Toy Story	247
2	13	1	5.000	Toy Story	247
3	15	1	2.000	Toy Story	247
4	19	1	3.000	Toy Story	247

Information Like:

count	6414.000
mean	13.942
std	27.773
min	1.000
25%	1.000
50%	4.000
75%	13.000
max	341.000

Name: totalRatingCount, dtype: float64

This part is a movie–user ratings matrix where:

userId	1	2	3	4	5	6	7	8	9	10	...	662	663	664	665	666	667	668	669	670	671
title																					
10 Things I Hate About You	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	3.000	0.000	0.000	0.000	0.000	0.000	0.000
12 Angry Men	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
2001: A Space Odyssey	0.000	0.000	0.000	0.000	0.000	0.000	4.000	0.000	0.000	0.000	...	0.000	0.000	4.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
A Beautiful Mind	0.000	0.000	0.000	0.000	4.500	0.000	0.000	3.500	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	4.000
A Bug's Life	0.000	0.000	0.000	0.000	3.500	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	4.000
A Christmas Story	0.000	0.000	0.000	5.000	0.000	0.000	0.000	3.500	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	5.000
A Clockwork Orange	0.000	0.000	0.000	5.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	3.500	3.000	0.000	0.000	0.000	0.000	0.000	3.000
A Close Shave	0.000	0.000	0.000	0.000	0.000	0.000	5.000	0.000	0.000	0.000	...	0.000	0.000	0.000	0.000	0.000	5.000	0.000	0.000	0.000	4.000
A Few Good Men	0.000	0.000	0.000	5.000	0.000	0.000	0.000	0.000	3.000	0.000	...	0.000	0.000	0.000	4.000	0.000	0.000	0.000	0.000	0.000	0.000
A Fish Called Wanda	0.000	0.000	0.000	5.000	0.000	0.000	4.000	0.000	0.000	0.000	...	0.000	0.000	2.500	0.000	0.000	0.000	0.000	0.000	0.000	0.000

10 rows × 669 columns

Then I create a K-Nearest Neighbors model by using Brute Force Search(compares each item with all others no fancy tree structures) and cosine similarity metric (measures the angle between vectors good for recommend):

NearestNeighbors(algorithm='brute', metric='cosine')

For the Major Part:

This takes movie titles and ratings, converts the titles to numbers, and trains a KNN classifier to predict ratings for movies. It then searches for the best parameters (neighbors, distance type, etc.) to make the most accurate predictions.

Best accuracy :: 0.3716618075801749

This Movies Matrix looks like :

<Compressed Sparse Row sparse matrix of dtype 'float64'
with 42723 stored elements and shape (449, 670)>

Coords	Values
(0, 12)	3.5
(0, 14)	5.0
(0, 44)	2.5
(0, 55)	2.0
(0, 60)	4.0
(0, 72)	3.5
(0, 73)	4.0
(0, 94)	5.0
(0, 287)	5.0
:	:
(448, 559)	4.5
(448, 562)	5.0
(448, 567)	1.0
(448, 573)	4.0
(448, 583)	5.0
(448, 596)	3.5
(448, 601)	5.0
(448, 622)	3.0
(448, 652)	4.5

Recommendation movie for this part is Like:

Recommendations for A Close Shave:

1: The Wrong Trousers, with distance of 0.3426323067106505:

2: This Is Spinal Tap, with distance of 0.6055093141588085:

3: Monty Python and the Holy Grail, with distance of 0.6101802075797369:

4: A Fish Called Wanda, with distance of 0.6147928467232779:

5: Life of Brian, with distance of 0.6189618572544828:

7 Hybrid Model

In this section all I do is to find CB and CF then observe their interaction.(NOTE:: I got user_id equal to 1)

$$s_{\text{hyb}}(u,i) = \alpha\, s_{\text{CF}}(u,i) + (1-\alpha)\, s_{\text{CB}}(u,i),$$

CB: I build a simple content-based similarity model by combining genres, keywords, cast, and crew . Computing cosine similarity between all movies, and then averaging similarity scores for the movies a given user has rated to generate their CB recommendation profile.

CF: For this I compute the collaborative filtering score vector for a user by taking their ratings, filling in unrated movies with their average rating, mapping movie IDs to row indices in the dataset, and producing a complete score array aligned with all movies in df.

After that, I normalize both the CF and CB scores, then combine them to observe how they influence each other.

For alpha I used 0.5 because CF works better when you have enough user–item interaction data, so you give it more influence.(I select randomly from [0.1, 0.9] for better performing)

$\alpha = 0.5 \rightarrow$ 50% weight on collaborative filtering
$1 - \alpha = 0.5 \rightarrow$ 50% weight on content-based filtering

What it become:

The user ID is necessary because you need to know which movies this user has already rated in order to Compute the CB similarity for the movies they liked (sCB). And fill missing ratings for CF with the user mean (sCF).

Without the user ID, the model wouldn't know whose history to consider. It can't just guess your preferences—it needs that specific user's past ratings.

Top-10 hybrid recommendations:

['To Die For', 'Blackmail', 'Monsoon Wedding', 'Volver', 'The Godfather: Part III', 'Italian for Beginners', 'Harold and Maude', 'Forrest Gump', 'Fargo', 'Crouching Tiger, Hidden Dragon']

CB VS CF:

The CB scores are consistently higher than the CF scores across all the recommendations, indicating that the content-based approach is performing better than the collaborative filtering approach for this dataset.

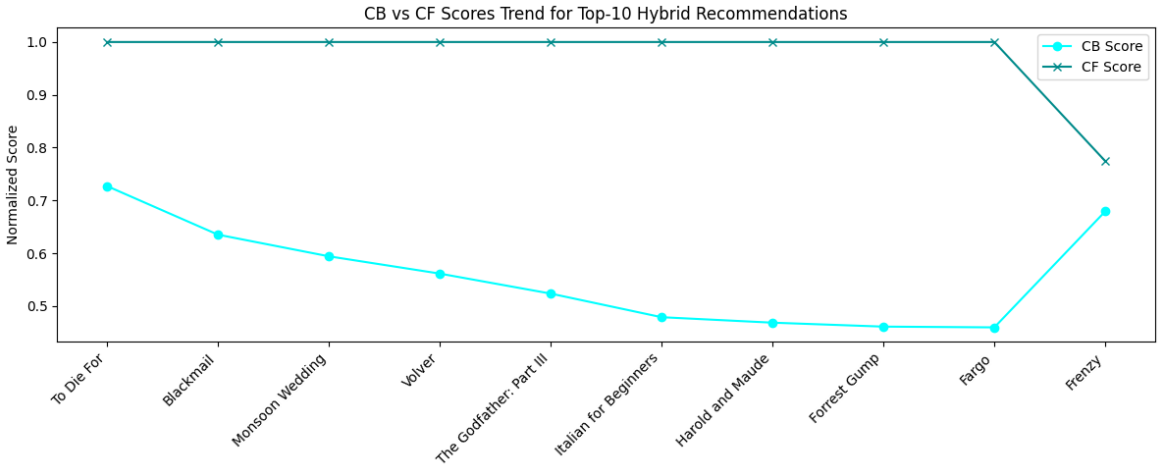


Figure9: There are decreasing trend in both CB and CF scores implies that the quality of the recommendations decreases as we move from the top-ranked to the lower-ranked recommendations.

8 Evaluation and Experiment Design

In this section, I keep only needed column for CB in ratings file and using Time Ware Protocol and it is leave-one-out per user.

1) Uses Count Vectorizer to turn movie metadata (genres, keywords, cast, crew) into a bag-of-words matrix.and computes cosine similarity between all movies to measure CB similarity.

2) Creates a dictionary mapping each movie's ID to its index in `df` for quick lookup.

3) Use Recommendation Function(Retrieves the user's rating vector from the training matrix/ For every movie the user rated, looks up similar movies in cb sim and weights them by the user's rating.)

Metric Definitions (Aggregates the metrics across all users to compute the mean Precision@K, Recall@K, and HitRate@K.)

Precision@K: Fraction of recommended movies in the top-K that are relevant to the user.

Recall@K: Fraction of relevant movies that appear in the top-K recommendations.

Hit Rate@K: Binary indicator showing whether at least one relevant movie appears in the top-K recommendations.

	K	Precision@K	Recall@K	HitRate@K
0	10	0.000674	0.002434	0.006742
1	20	0.000899	0.005468	0.017978

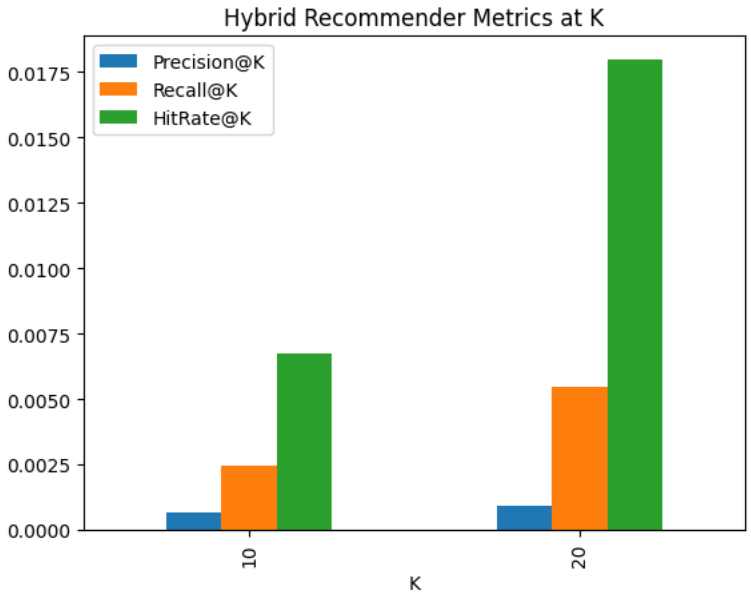


Figure10: This says , how these three metrics change as the value of K (the number of top recommendations) increases from 10 to 20. We can see that as K increases, the Precision@K and Recall@K metrics generally increase, while the HitRate@K metric approaches 1, indicating that the recommendation system is more likely to surface at least one relevant item in the top-K recommendations.

This section computes 95% bootstrapped confidence intervals for ranking metrics (Precision@K, Recall@K, HitRate@K) by resampling the metric values 1,000 times with replacement and calculating the mean for each sample. The lower and upper percentiles (2.5% and 97.5%) define the confidence interval, giving an estimate of the variability of the metrics. Finally, the metrics across different K values are visualized to compare their trends.

95% CI: {'Precision@K': [np.float64(0.0006741573033707866), np.float64(0.0008988764044943821)], 'Recall@K': [np.float64(0.0024344569288389513), np.float64(0.0054816479400749)], 'HitRate@K': [np.float64(0.006741573033707865), np.float64(0.017977528089887642)]}

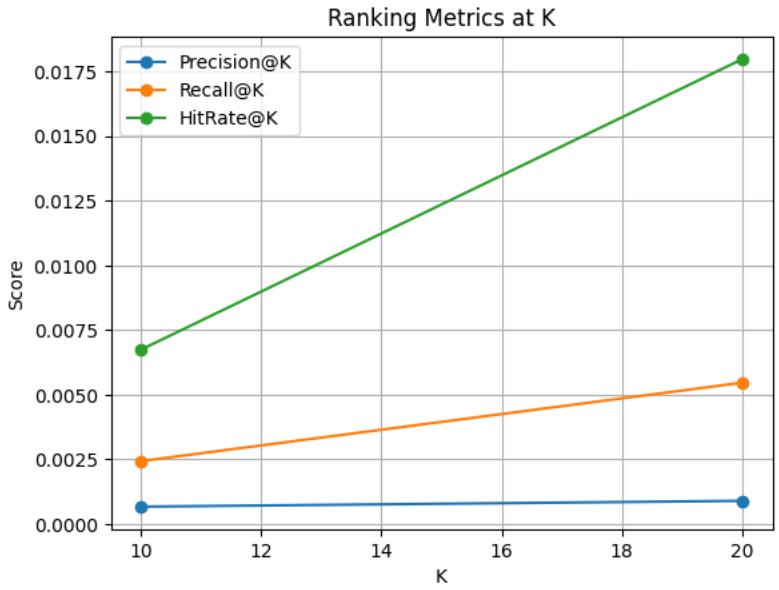
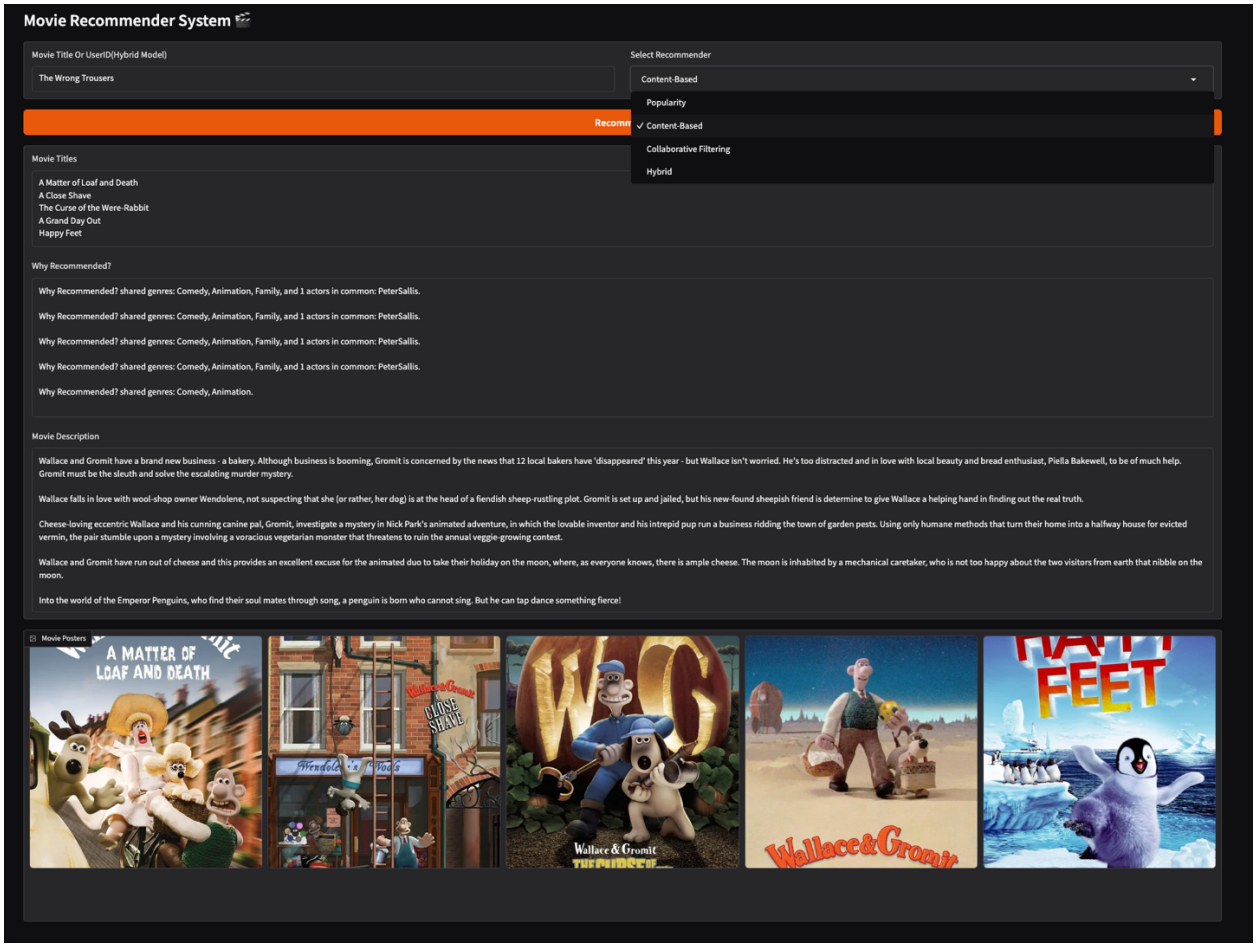


Figure11: The plot shows how the different metrics change as more recommendations are considered.

9 Conclusion

So, now we have a recommendation system with 3 difference recommendation such as Content-based , Collaborative Filtering and Hybrid Model , that recommend every movie from which genre or title or userId that we have like this mini app:



References

[1] Rounak Banik “The Movies Dataset” 2017.
https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset/