Bryce Doperalski

COMPUTER SCIENCE 464: Project Report

Introduction

This project is an implementation of a Movie Recommendation System. The system uses multiple methods of recommending movies based on the title of the movie, then the user receives a list of movies that match the input. These lists are based off of ratings, genres, movie descriptions, or the cast and crew of a movie, depending on which method is being used.

Related Work

The dataset being used is “The Movies Dataset” from Rounak Banik on Kaggle. The dataset includes metadata from over 45,000 different movies from the Full MovieLens dataset. It includes movies released on or before July 2017 and it includes cast, crew, keywords for the plots, titles, vote averages, vote counts, among a bunch of other useful data. There is another file with ratings from over 270,000 users for the movies in the dataset, all of which are sourced from the GroupLens website.

Related works that use this type of system include IMDb’s recommendation system. Movies are recommended to users based off of their ratings of movies, similar to the implementation in this project. Netflix and other streaming services have systems that will recommend new movies or shows to the user, so they stay on the platform longer.

Recommendation systems are not only for movies. They can be used for recommending new products at an online retailer for users. Using “Collaborative Filtering”, users and their ratings are compared with others directly, providing a score on how much that user may like the product being recommended to them.

Proposed Method & Experiments

The first movie recommendation method is an extremely simple, bare bones, and basic recommender. It simply orders the highest rated movies out of the dataset of 45,000+ movies using a formula by IMDb, weighted rating:

$$\text{Weighted Rating (WR)} = \left(\frac{v}{v+m} . R\right) + \left(\frac{m}{v+m} . C\right)$$

where ‘v’ is the number of votes for a movie, ‘m’ is the minimum required votes to be listed in a chart, ‘R’ is the average rating for a movie, and ‘C’ is the average of the votes across the whole report. The way this can be calculated in code is as follows:

```
def weighted_rating(x, m = m, C = C):
    v = x['vote_count']
    R = x['vote_average']
    # Compute the weighted rating of each movie based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

This function allows us to get a list of movies that are popular and rated highly based on the average ratings and amount of them. The list can bye thought of a “popular movies” list or a trending list of movies.

	title	overview	imdb_id	genres	vote_count	vote_average	score
314	The Shawshank Redemption	Framed in the 1940s for the double murder of h...	tt0111161	[Drama, Crime]	8358.0	8.5	8.445869
834	The Godfather	Spanning the years 1945 to 1955, a chronicle o...	tt0068646	[Drama, Crime]	6024.0	8.5	8.425439
10309	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	tt0112870	[Comedy, Drama, Romance]	661.0	9.1	8.421453
12481	The Dark Knight	Batman raises the stakes in his war on crime. ...	tt0468569	[Drama, Action, Crime, Thriller]	12269.0	8.3	8.265477
2843	Fight Club	A ticking-time-bomb insomniac and a slippery s...	tt0137523	[Drama]	9678.0	8.3	8.256385
292	Pulp Fiction	A burger-loving hit man, his	tt0110912	[Thriller, Crime]	8670.0	8.3	8.251406

This weighted rating function can be reused to then search for best rated movies based on an input genre, instead of a title. Below is a search for “Romance” movies:

	title	overview	vote_count	vote_average	popularity	score
10309	Dilwale Dulhania Le Jayenge	Raj is a rich, carefree, happy-go-lucky second...	661	9	34.457024	8.565285
351	Forrest Gump	A man with a low IQ has accomplished great thi...	8147	8	48.307194	7.971357
876	Vertigo	A retired San Francisco detective suffering fr...	1162	8	18.20822	7.811667
40251	Your Name.	High schoolers Mitsuha and Taki are complete s...	1030	8	34.461252	7.789489
883	Some Like It Hot	Two musicians witness a mob hit and struggle t...	835	8	11.845107	7.745154
1132	Cinema Paradiso	A filmmaker recalls his childhood, when he fel...	834	8	14.177005	7.744878
19901	Paperman	An urban office worker finds that paper airpla	734	8	7.198633	7.713951

The next method approaches the problem of recommending movies based on their descriptions. This is accomplished by using “Term Frequency-Inverse Document Frequency” or TF-IDF. This method calculates the amount of times a term is used in a singular instance or description compared to the amount of times that term appears in the dataset as a whole. This is then multiplied by the log of the number of descriptions divided by the amount of descriptions with that term in it. This then gives a matrix where each column is a word or term in the description vocabulary, where each row is a movie.

The next step is to calculate a similarity score. There isn't one right way to calculate this, but the implementation in the project is using cosine similarity:

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

An example of a function that uses the cosine similarity is a method that is used to recommend movies based on the title of the movie, using the terms in the movie's description to recommend movies with similar terms:

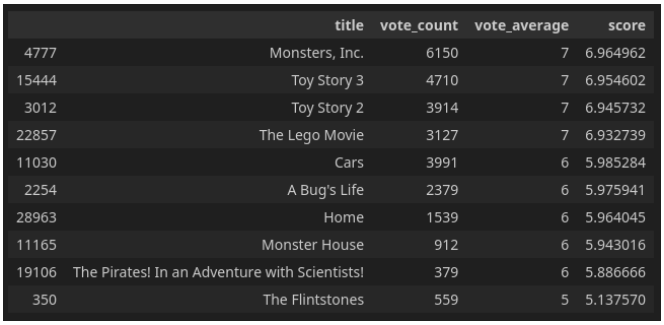
```
def get_recommendations_description(title):
    index = new_movies_data[new_movies_data['title'] == title].index[0]
    similar_scores = list(enumerate(cosine_sim[index]))
    similar_movies = sorted(similar_scores, key=lambda x: x[1], reverse=True)
    similar_movies = similar_movies[1:31]
    movie_indices = [i[0] for i in similar_movies]
    return titles.iloc[movie_indices]
```

```
get_recommendations_description('Toy Story').head(15)
✓ 0.0s

15348                                  Toy Story 3
2997                                   Toy Story 2
24522                                    Small Fry
10301                        The 40 Year Old Virgin
23842                  Andy Hardy's Blonde Trouble
3057                                  Man on the Moon
29201                                    Hot Splash
43424               Andy Kaufman Plays Carnegie Hall
38473    Superstar: The Life and Times of Andy Warhol
6435                          What's Up, Tiger Lily?
42718    Andy Peters: Exclamation Mark Question Point
```

Going further with this method, we can alter the function that is used to recommend movies to use the cast, crew, and genres to give us another set of recommendations. To accomplish this, all of the metadata is compiled into a single column, a "soup" of a sort. The director of the movie is alienated and then tripled when put into the column to be more prominent when comparing it to other movies, as the director is typically a crew member that people search for in their movies. Instead of using TF-IDF for this method a "Count Vectorizer" is used instead as to not weigh down the director of the movies. A new cosine similarity is used and then the above function is reused to make a recommendation:

```
get_recommendations_description('Toy Story').head(15)
✓ 0.0s

10710                    Luxo Jr.
19227                     Tin Toy
19281                Red's Dream
3012                 Toy Story 2
19331                Knick Knack
17477                     Cars 2
22992    Mater and the Ghostlight
11030                        Cars
2254                A Bug's Life
22064        Toy Story of Terror!
15444                Toy Story 3
```

This method can still be improved further. The function can be changed to include the weighted rating from the Simple Recommender. This allows the function to use the scores from the weighted rating and then the recommender that uses metadata.



	title	vote_count	vote_average	score
4777	Monsters, Inc.	6150	7	6.964962
15444	Toy Story 3	4710	7	6.954602
3012	Toy Story 2	3914	7	6.945732
22857	The Lego Movie	3127	7	6.932739
11030	Cars	3991	6	5.985284
2254	A Bug's Life	2379	6	5.975941
28963	Home	1539	6	5.964045
11165	Monster House	912	6	5.943016
19106	The Pirates! In an Adventure with Scientists!	379	6	5.886666
350	The Flintstones	559	5	5.137570

The third and fourth methods for recommending movies are a bit more personal to the user, rather than being more general. With “Collaborative Filtering”, a score is predicted based on the dataset of movies. There are two types of Collaborative Filtering, user-based and item-based filtering. This project takes advantage of the item-based Collaborative Filtering. Item-based Collaborative Filtering recommends items based on their similarity with the items that the target user rated. It uses Cosine Similarity to accomplish this. This can be accomplished using a few algorithms, including “Singular Value Decomposition” (SVD) or a K-Nearest Neighbors algorithm. The method used in this project is the SVD method.

Singular Value Decomposition is a latent factor model that helps leverage scalability and sparsity issues that may arise in a project like this. It turns a recommendations problem into an optimization problem instead. It uses “Root Mean Square Error” as a metric for accuracy. Essentially, SVD maps users and items into a latent space, making users and items directly comparable to each other. The implementation used was about 80% accurate from cross validation. There is some tuning that can be done to get that percentage higher, but it didn’t happen for this project.

The “Hybrid” Recommender is a continuation of the Collaborative Filtering method, but with some added flavor. This recommender will give a list of movies based on the title of the movie for a specific user ID. This method is personal to the user ID that is input into the algorithm. Every

recommendation for every user ID will be different. This method displays the recommendations in order from their scores on how accurate the model is predicting the user to be interested in the recommended movies.

```
hybrid_recommender(1, 'Avatar')
```

	title	genres	vote_average	vote_count	id	score
45137	T2 3-D: Battle Across Time	[]	7.0	29.0	65595	4.073572
7480	Babylon 5: A Call to Arms	[Action, Drama, Science Fiction, Adventure]	6.9	36.0	10916	4.057682
18996	Icarus XB 1	[Science Fiction]	6.7	16.0	19757	4.057216
26720	Star Wars: The Force Awakens	[Action, Adventure, Science Fiction, Fantasy]	7.5	7993.0	140607	3.923079
1167	Aliens	[Horror, Action, Thriller, Science Fiction]	7.7	3282.0	679	3.919049
9701	Aliens of the Deep	[Action, Documentary, Science Fiction]	6.8	19.0	22559	3.916247
1101	The Abyss	[Adventure, Action, Thriller, Science Fiction]	7.1	822.0	2756	3.842323

For user ID 1, with the movie ‘Avatar’ being the movie we are searching for recommendations for, we see that ‘T2 3-D: Battle Across Time’ is the highest predicted movie to recommend to our user with subsequent recommendations following it in order of score.

```
hybrid_recommender(500, 'Toy Story 2')
```

	title	genres	vote_average	vote_count	id	score
15444	Toy Story 3	[Animation, Family, Comedy]	7.6	4710.0	10193	5.000000
19331	Knick Knack	[Animation]	7.1	135.0	13928	4.975998
0	Toy Story	[Animation, Comedy, Family]	7.7	5415.0	862	4.963822
25492	The Kingdom of Dreams and Madness	[Animation, Documentary]	7.7	49.0	252511	4.879204
2254	A Bug's Life	[Adventure, Animation, Comedy, Family]	6.8	2379.0	9487	4.860139
13808	Up	[Animation, Comedy, Family, Adventure]	7.8	7048.0	14160	4.777556
3979	The Brave Little Toaster	[Fantasy, Adventure, Animation, Comedy, Family...	6.8	94.0	19933	4.760014
18339	Arthur Christmas	[Drama, Animation, Family, Comedy]	6.7	340.0	51052	4.739131

For user ID 500, with the movie ‘Toy Story 2’ being the movie we are searching for recommendations for, we see that ‘Toy Story 3’ is the highest predicted movie to recommend to our user with more recommendations following.

If we were to input the same movie title but different user IDs, we would receive different recommendations because this method is taking in that user’s scores for the movies they rated. Everyone rates movies differently and has different tastes in movies.

Results and Discussion

The results of the project are satisfactory for me, although I acknowledge there may need to be more wok done on this project and model to make them

more accurate. 80% accuracy isn't the most accurate for recommending new things to users, but then again, movies are very subjective and even if the algorithm gives a perfectly scored recommendation, the user may not like the movie anyway.

Improvements could include using another model to grant recommendations. I mentioned that a K-Nearest Neighbors model could have been used, I just hadn't had the time to properly test or implement it to compare it to the Singular Value Decomposition Model.

Conclusions

In conclusion, the project was a fine learning experience, and could be improved if that were so desired. Movie Recommendation is a fairly simple concept, mostly using math and machine learning to come up with a solution.

References

The following links are the sources that I followed and the dataset that I used for the project.

Ahmed, Ibtesam. "Getting Started with a Movie Recommendation System." *Kaggle*, Kaggle, 2 Feb. 2024, www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system.

Banik, Rounak. "Movie Recommender Systems." *Kaggle*, Kaggle, 6 Nov. 2017, www.kaggle.com/code/rounakbanik/movie-recommender-systems.

Banik, Rounak. "The Movies Dataset." *Kaggle*, 10 Nov. 2017, www.kaggle.com/datasets/rounakbanik/the-movies-dataset/data.