Comparative Analysis of Recommendation Engines for Movies

Harshit Pandey, Omkar Ubale, Archit Bhonsle
Khoury College of Computer Sciences
Northeastern University
Boston, MA, 02115
pandey.har@northeastern.edu,
ubale.o@northeastern.edu,
bhonsle.a@northeastern.edu

Abstract

Using the movies and ratings dataset, we demonstrate building a recommendation engine using various methods and use Normalized Discounted Cumulative Gain (NDCG) to compare the top 5 recommendations. We explore Collaborative Filtering, embedding-based Filtering, and variations of Content-Based Filtering: Popularity-based Recommenders, Similarity-based Recommenders, and Text Similarity-based Recommenders.

1 Introduction

In this project, we worked on building recommendation engines using the "The Movie Dataset" which has metadata on over 45,000 movies and 26 million ratings from over 270,000 users. The dataset contains ratings and metadata about the movies like cast and crew that worked on the movie, keywords describing the movie and other fields like languages, overview and revenue. Several preprocessing steps were undertaken to ensure data quality and suitability for model training:

- Handling Null Values and Correcting Data Types: Null values were addressed by imputation or removal, depending on the context. Data types were corrected to ensure consistency and accuracy in analysis.
- Removing Niche Movies: To focus on mainstream movies with sufficient ratings for reliable recommendations, movies with a vote count below a certain threshold (e.g., 36) were filtered out.
- Data Integration: Various tables containing information about movies, users, and ratings were integrated to create a comprehensive movie database with relevant features.
- Cleaning JSON Columns: Some columns containing JSON data were cleaned and structured to extract relevant information for analysis.
- Train-Test Split: The dataset was split into training and test sets to evaluate the model's performance accurately. For example, a user with 100 ratings for 100 movies might have 80 movies in the training set and 20 movies in the test set, ensuring that the model is evaluated on simulated new movie data.
- Feature Engineering: Multicategory columns were split, and categorical features were one-hot encoded, while numerical features were scaled to ensure consistent scaling across features.

CS 6140 Final Project under Professor Paul Hand.

2 Methods

2.1 Collaborative Filtering

In our approach for Collaborative filtering, we use the ratings given by the users to provide recommendations closest to the movies a user has rated highly. We treat the recommendation problem as an optimization problem and solve the minimization using Single Value Decomposition (SVD), and use Root Mean Square Error (RMSE) to assess the efficacy of our model.

2.2 Content-based Filtering

Content-based filtering aims to recommend items based on the characteristics of the features. The goal here is to recommend the user items which are similar to the items that are highly rated by them. For example, we might recommend "The Dark Knight Rises" and "Inception" to someone who has highly rated "Batman Begins" and "The Prestige" as they are directed by Christopher Nolan.

2.2.1 Popularity-based Recommenders

The `movie_metadata.csv` file has two fields of that allude to the popularity of the movies: `vote_average` and `vote_count`. We can rank the movies by their `vote_average` to rank them by thier average rating. However, there might be some movies which have a high rating with a very small amount of votes. To mitigate this we can use a weighted rating:

$$WR = (v/(v+m)*r) + (m/(v+m)*c)$$

Here, v is the `vote_count`, r is the `vote_average`, m is the 90% quantile of the `vote_count`, c is the `vote_average`. This scheme tempers the previously stated effect.

2.2.2 Similarity-based Recommenders

Instead of only considering the popularity of the movies, we consider related features. Using these features, we compute the similarity of the movies the user has rated with the rest of the movies. We use the weight of each of these similarities with the user's ratings to get an estimated ordering on the movies we want to recommend to the user.

This measure of similarity is quite straightforward for say something like `genres` field where we can one-hot encode the genres and use cosine similarity to compare the movies. For other fields like `keywords` field, we need to clean the data. For fields with a large number of categories like `crew` and `cast`, one-hot encoding these features can get cumbersome. Intead, we concatenate all the category names and use the `CountVectorizer` in `scikit-learn` to get a matrix which represents the presence of all the categories in the movies. We then use cosine similarity to compare the movies. To make matters simpler, we combine the strings from multiple such features into a "word soup" to reduce training time and complexity.

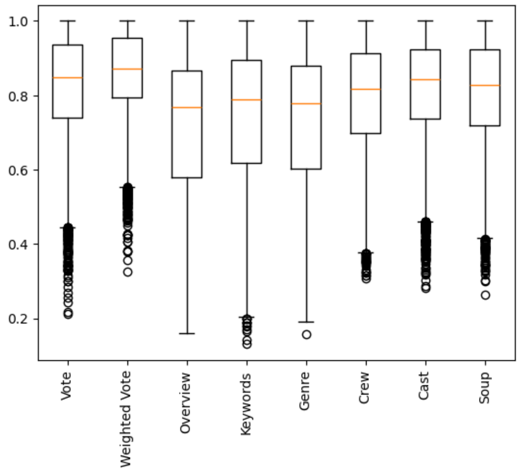
2.2.3 Text Similarity-based Recommenders

We use the movie `overview` for this type of recommender. We remove all the stop words and perform stemming to reduce similar words, as comparing only frequency of the words in the overview is not enough. We need a measure of importance for the words and TF-IDF (term frequency–inverse document frequency) is the perfect candidate for this job.

2.2.4 Comparison

Surprisingly, recommending popular movies seems to be the best performing approach. The overview field shows a lot of promise but it doesn't contain enough text for TF-IDF to do its magic. To truly get the most of this feature, we need a much more sophisticated way to extract the insight contained in the overview. Another interesting observation is that, all the similarity based rankers don't perform as well. A problem with the keywords and genre fields is that they cast the net too wide. In the future, instead of using these fields to rank the movies, we could use them to narrow down the search space and then use a more expensive but powerful technique to determine the final ranks.

Figure 1: Comparing content-based recommenders

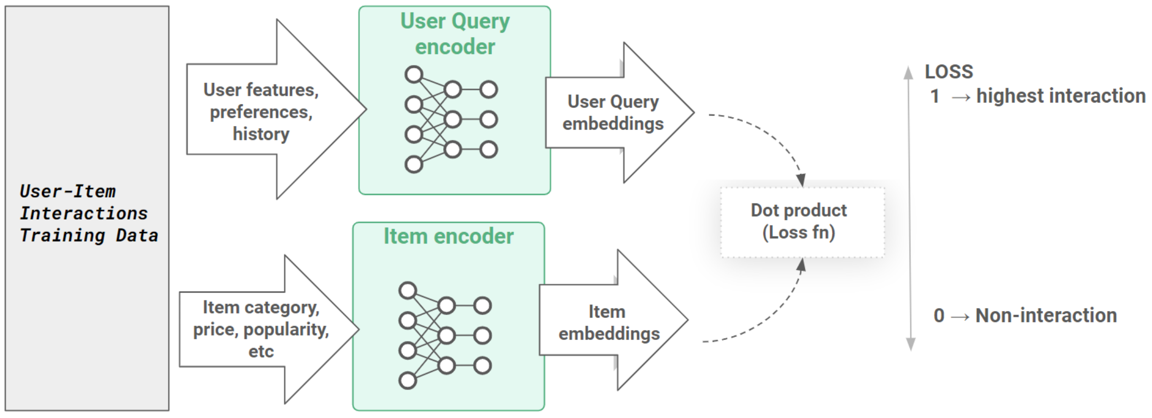


2.3 Neural Network Architecture

The two-tower neural network [6,7] approach comprises distinct architectures for user and item representations:

The **User Tower** involves a neural network generating user embeddings from user vectors, which are extracted from user-specific features. Meanwhile, the **Item Tower** entails another neural network generating item embeddings from movie vectors, derived from movie-specific features. The model optimizes a loss function aiming to minimize cosine similarity between user and item embeddings, ensuring normalization within [-1, 1]. Specifically, for movies liked by users (ratings ≥ 3.5), the model minimizes the distance between the respective embeddings, while for disliked movies (ratings < 3.5), it maximizes the distance between embeddings.

Figure 2: Two tower neural network diagrammatically shown.



2.3.1 Training

The experimental setup for training involves several key parameters and architectural specifications:

Threshold for Movie Likability: Movies rated at or above 3.5 are deemed favorable by users, influencing model training accordingly.

Training Parameters: The model is trained with a batch size of 64 over 10 epochs, using the Adam optimizer with a learning rate of 0.001.

Neural Network Architectures: The User Neural Network, with an input dimensionality of 100 and output of 768, incorporates ReLU activation functions. Similarly, the Movie Neural Network has an input dimensionality of 1536 and an output of 768, also utilizing ReLU activation functions.

Loss Function: During training, the model uses a Mean Squared Error (MSE) loss function with Cosine Similarity of the normalized user embedding and normalized item embedding, focusing on whether users favor particular movies or not based on the threshold of movie likability.

2.4 Evaluation

The evaluation process encompasses the following steps:

- Each of the models scores are calculated and stored.
- NDCG Calculation: For each user in the test set, the model's scores are ranked, along with the corresponding ratings. Normalized Discounted Cumulative Gain (NDCG) at 5 is used to compare the top 5 rankings of predictions versus the actual ratings.
- Aggregate NDCG Calculation: The average NDCG values for all users are computed to assess the overall performance of the recommendation algorithm.

3 Conclusion

Based on the average NDCG scores obtained from our approaches, we find that that each method has its strengths and limitations. Collaborative filtering achieved an average NDCG score of 0.69. This method focuses solely on user-item interactions, neglecting content-based features. While its performance was reasonable, it's worth noting that relying solely on past interactions may limit its ability to capture users' nuanced preferences. Integrating content-based features could potentially enhance its performance by providing a more holistic view of user interests.

Content-based filtering outperformed the other methods with an average NDCG score of 0.86. By leveraging item attributes, this approach offers personalized recommendations based on item similarities. However, solely relying on certain item features might overlook collaborative signals present in user-item interactions. Additionally, ordering recommendations by popular vote, while effective in enhancing user satisfaction, may lead to a lack of diversity in recommendations.

The embedding-based method, particularly the two-tower neural network approach, yielded an average NDCG score of 0.62. This hybrid approach combines collaborative and content-based filtering by learning low-dimensional representations of users and items. However, a notable limitation was the small size of the user vector, which lacked variability compared to the item vector. Incorporating additional user information, such as demographics and search history, could address this limitation and enhance the method's performance.

In conclusion, each approach offers unique advantages and challenges in recommendation systems. Future research could focus on integrating hybrid approaches that leverage the strengths of collaborative, content-based, and embedding-based methods while addressing their respective limitations. Additionally, exploring techniques to incorporate additional user information and improve recommendation diversity could further enhance recommendation accuracy and user satisfaction.

Please find links to the repository for all of our work below in the references [8,9,10].

References

[1] The Movies Dataset: https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

[2] Van Meteren, Robin, and Maarten Van Someren. "Using content-based filtering for recommendation." Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop. Vol. 30. 2000.

[3] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[4] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[5] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[6] Yang, Ji, et al. "Mixed negative sampling for learning two-tower neural networks in recommendations." Companion proceedings of the web conference 2020. 2020.

[7] Covington, Paul, Jay Adams, and Emre Sargin. "Deep neural networks for youtube recommendations." Proceedings of the 10th ACM conference on recommender systems. 2016.

[8] Content-based filtering https://github.com/ArchitBhonsle/the-movies-dataset

[9] Two Tower Neural Network (Embedding Based Method) :https://github.com/harsh4799/machine_learning_project

[10] Collaborative Filtering : https://github.com/omkarubale/CS6140_Project_CollaborativeFiltering