Two Towers Recommendation System

Marcel Fernández
marcelfernandez.serrano@ru.nl
Radboud University
Nijmegen, Netherlands

Alejandro Pastor Rubio
alejandropastor.rubio@ru.nl
Radboud University
Nijmegen, Netherlands

Xuezheng Zhang
xuezheng.zhang@ru.nl
Radboud University
Nijmegen, Netherlands

ABSTRACT

Recommendation systems play a crucial role in personalizing the user experience by targeting different user preferences. In this paper, we implement and evaluate a two-tower model recommendation system. We explore two different recommendation methods: weighted average and clustering-based methods. The former focuses on user-item relationships, and the latter improves accuracy through clustering techniques. To evaluate the performance, we study the performance in different user scenarios using anime ratings dataset. The trade-off between accuracy and computational efficiency is optimized on the basis of the results. Experimental shows that the two tower model approach implemented in this paper achieves a good balance between computational efficiency and recommendation accuracy, providing an effective solution for large-scale applications.

ACM Reference Format:
Marcel Fernández, Alejandro Pastor Rubio, and Xuezheng Zhang. 2024. Two Towers Recommendation System. In *Proceedings of Information Retrieval.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In our daily lives, we encounter recommendation systems everywhere. We find them when listening to music, buying clothes, or even when typing an email and receiving suggestions. - In this paper, we focus on recommending animes to improve user experiences. Our aim is to promote engagement by predicting user preferences and providing tailored recommendations. However, as users of these recommendation systems, we must take two factors into account: the amount of time spent making these recommendations and the quality of the recommendations. Both characteristics must be addressed to improve the user experience. The source code for our implementation of the recommendation system is available on Github[1].

1.1 Problem Statement

To provide useful suggestions, models must find a balance between accuracy and performance. Weighted Average Recommendation is computationally efficient, but it may lack precision when modeling a wide range of user preferences. While more advanced, clustering-based recommendation requires preprocessing and heavily depends on the quality of clustering. We propose assessment criteria that allow for meaningful comparisons between different techniques. This metric evaluates the relevance of recommendations by calculating the distance between recommended items and cluster centroids associated with a user's rated items.

2 RELATED WORK

Utility matrices based recommendation systems usually rely on collaborative filtering techniques. User preferences are predicted by analyzing historical behavioral data. These systems provide a straightforward framework for recommending items but face challenges like sparsity and cold start problems. Through this study, we expanded our knowledge by exploring advanced architectures such as the Two-Tower Model, which addresses these limitations by learning independent representations for users and items through deep neural networks while leveraging interaction modeling for enhanced performance. This approach aligns with recent advancements in recommendation systems, as discussed in Yantao Yu, et al. [4]. Our main goal is to explore the trade-offs between different methods, and we draw on the work of Harrie Oosterhuis and Maarten de Rijke [2], which presents innovative approaches to build robust and practical recommendation systems. In personalized recommendation system, Cheng et al. [1] optimized the representation of the two-tower model and the method of calculating dot product. It provides ideas for optimizing the efficiency of similarity calculation in this paper.

3 EXPERIMENTAL SETUP

3.1 Dataset Selection

We selected the Anime Ratings dataset [2], which includes user_id, anime_id, and rating in rating.csv. This dataset can be merged with anime.csv to access additional details, such as anime names and genres, which are useful for visualization.

3.2 Data Preprocessing

To prepare the dataset for building a recommendation system, we performed several preprocessing steps.
First of all, we *excluded* users with *only one anime review* (3,249 users) because they lack sufficient data to support the recommendation system. Identifying patterns in user preferences is impossible with a single review, and removing such users enhances model accuracy and evaluation. To continue, we *shuffle the data*, shuffling ensures randomized data order, preventing the model from learning

[1] https://github.com/U198734/Two-Towers-Recommendation-System

[2] https://www.kaggle.com/code/hasibalmuzdadid/anime-ratings-analysis-recommender-system/notebook

Information Retrieval, December 2024, Nijmegen, Netherlands
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

spurious patterns (e.g., sequence-based biases). It also avoids data leakage between training and validation sets, enabling more reliable evaluation and generalization. After shuffling, 500 users were reserved for the *validation set*. This unseen data helps evaluate the model's performance, detect overfitting, and fine-tune hyperparameters. A separate validation set guarantees the model is robust and generalizes well to real-world scenarios.

4 TRAINING MODEL

In this section, we describe the process of constructing a recommendation model based on user preferences, including the dataset segmentation process, embedding representation, model design and efficiency improvement.

4.1 Dataset Splitting Methodology

To divide the dataset into training and testing sets, we employed a user-centric approach. First, for each unique user ID extracted from the dataset, all associated entries (i.e., all ratings given by that user) were collected. These entries were then split into training and testing sets, with 90% of the entries allocated to the training set and the remaining 10% to the testing set. The resulting datasets, X_{train} and X_{test}, each consist of two fields: user IDs and anime IDs. The corresponding target values, Y_{train} and Y_{test}, represent the ratings. To ensure the integrity of the split, we validated that there is no overlap between the user IDs in the training and testing sets. Specifically, no user ID present in X_{train} is found in X_{test}, and vice versa. This precaution assures a clean separation of users between the two subsets, facilitating reliable model evaluation.

4.2 Embedding Creation and Representation

To represent users and items in a latent feature space, we developed a recommendation model using Flax's linen module[3]. The embedding generation is handled by a neural network module called *SimpleRecSystem*, designed to learn low-dimensional representations for both users and items.

4.3 Model Architecture

The Two Tower model consists of three primary components:

(1) **User Embeddings:** This component maps each user to a dense vector of size $n_{\text{factors}} = 32$. The embedding layer is initialized using Flax's `linen.Embed` module, where the total number of unique users serves as the input size.
(2) **Item Embeddings:** Similarly, items are mapped to vectors in the same n_{factors}-dimensional space. The embedding layer for items is also defined using `linen.Embed` with n_{items} as the input size.
(3) **Interaction Mechanism:** The model computes the interaction between user and item embeddings by performing an element-wise multiplication of their respective embeddings. The result is reduced to a single scalar value for each user-item pair by summing along the latent dimensions. The final output is passed through a ReLU activation function to ensure non-negativity.

[3]https://flax-linen.readthedocs.io/en/latest/quick_start.html

4.4 Loss Functions

To optimize the recommendation model, we employed Mean Squared Error loss functions to evaluate its performance during training: The MSE loss function calculates the average squared difference between the model's predictions (s) and the actual target values (y). It is defined as:

$$\text{MSELoss}(s, y) = \frac{1}{n}\sum_{i=1}^{n}(s_i - y_i)^2$$

This function penalizes larger errors more heavily, encouraging the model to minimize significant deviations.

4.5 Hyperparameter Selection

To train the recommendation model effectively, we carefully selected the following hyperparameters to balance training speed and model performance:

(1) **Number of Epochs:** Trained for 35 epochs to minimize loss while avoiding overfitting.
(2) **Batch Size:** A batch size of 10,000 enabled efficient processing of samples.
(3) **Learning Rate:** Set to 0.01 for a balance between speed and stability.
(4) **Optimizer:** Used `optax.nadam`[4] for adaptive parameter updates, combining NAG and Adam benefits.
(5) **Model Initialization:** Parameters were initialized with JAX's `random.randint` in the range $[1, 20]$ for diverse starting weights.

These hyperparameters were chosen based on experimentation and domain knowledge to ensure robust training and convergence to an optimal solution.

4.6 Efficiency with jax.jit

The *jax.jit* library[5] enhances computational efficiency by just-in-time (JIT) compiling Python functions into optimized machine code using XLA. This speeds up execution, fuses operations to reduce overhead, and enables seamless GPU/TPU acceleration, significantly improving performance for large-scale computations.

5 TESTING

To delve deeper into the testing phase, we will divide this section into three parts: The first part focuses on making predictions for users the model has seen during the training phase. The second part involves recommending anime to a user the model hasn't seen before, based on their previous ratings. The third part addresses the specific case of introducing a new user to the system without any prior ratings.

5.1 Testing Predictions for Seen Users

We have implemented a method to make predictions for users the model has encountered during training. A user is considered "seen" by the model if we have their representation in the embedding space. Using this representation, we apply the weights learned during training to predict ratings for anime the user has not yet

[4]https://optax.readthedocs.io/en/latest/
[5]https://jax.readthedocs.io/en/latest/jax.html#jax.jit

interacted with. This enables us to recommend anime the user is likely to enjoy. This approach is reminiscent of a linear regression problem, as we use the predicted ratings to make recommendations.

5.2 Recommendations for Unseen Users with Prior Ratings

A complex situation arises when we need to recommend anime for an unseen user, one who does not have a mapping in the embedding space. To address this problem, we cannot rely on the simple regression approach used in the previous section.
In this section, we will describe and compare two different algorithms: one that employs a straightforward technique, such as a weighted average, and another that uses a more advanced approach involving clustering.

5.2.1 Weighted Average Approach

In this technique, since we cannot use the direct user mapping in the embedding space, we rely on the user's previously rated anime. One advantage of the Two-Tower model is that it allows us to leverage the item (anime) embeddings to infer the user's position in the embedding space and provide meaningful recommendations. To achieve this, we use the item tower, which represents the anime embeddings. By utilizing the embeddings of previously rated anime, we compute the user's embedding as a weighted average of these vectors. The weight for each anime embedding is derived from the corresponding user rating, ensuring that higher-rated anime contribute more significantly to the calculated user embedding.
Let e_{ai} be the embedding vector of anime ai and r_i be the rating given by the user to anime a_i. The user embedding e_u is calculated as:

$$\mathbf{e}_u = \frac{\sum_{i=1}^{n} r_i \cdot \mathbf{e}_{ai}}{\sum_{i=1}^{n} r_i}$$

Where:

- n is the number of anime rated by the user.
- r_i is the rating assigned by the user to anime ai.
- e_{ai} is the embedding vector of anime ai obtained from the item tower.

5.2.2 Clustering-Based Approach

This approach builds on the technique described in Section 4.2.1, but instead of relying solely on a weighted average, it incorporates clustering. Once the user's embedding is located within the embedding space (Like in Section *5.2.1*), the nearest cluster is identified. From this cluster, the system recommends the most popular movie that the user has not yet seen, ensuring relevance while leveraging the cluster's aggregated preferences.
We work with two different clustering algorithms to guarantee that the distribution represented in the embedding space is well-defined by clusters. Specifically, we use *K-means* clustering[6] and Gaussian Mixture models[7]. The main difference between these methods lies in their underlying assumptions and how they define clusters.
K-means assumes that clusters are spherical and separated by equal variance, assigning each point to the nearest cluster centroid. In contrast, Gaussian Mixture models are more flexible, allowing clusters to have different shapes, sizes, and orientations by modeling them as a combination of Gaussian distributions.

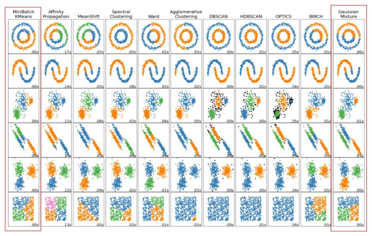


Figure 1: Different Clustering Methods [3]

To determine the optimal number of clusters, we employed the simplest method, the Elbow method[8]. The optimal number of clusters was found to be 150.

5.3 Cold-Start Scenarios for New Users

To address the cold start problem[9] for new users with no prior information, we employed a popularity-based approach. Each anime is assigned a *weighted score*, combining its average rating with the number of members who have rated it. We then select the top-rated anime based on this score. By focusing on popular titles with high ratings, we assume that if many users have liked an anime, it is likely that the new user will also enjoy it, even in the absence of user-specific data.

6 EVALUATION

To evaluate the performance of the recommendation system, we focus on unseen users by first splitting their data in half and shuffling the order. After splitting into training and evaluation sets, the training set data is used to compute the average weight embeddings. When it comes to similarity, we use standard metrics such as Precision@k and MAP (Mean Average Precision).
DCG[10] is also a ranking evaluation metric commonly used in recommendation systems, which takes into account both relevance and positional preferences. We normalize it by dividing by the idealized DCG (IDCG) to obtain NDCG. Then we evaluate clustering and the average weight approach to compare the performance of the two methods.

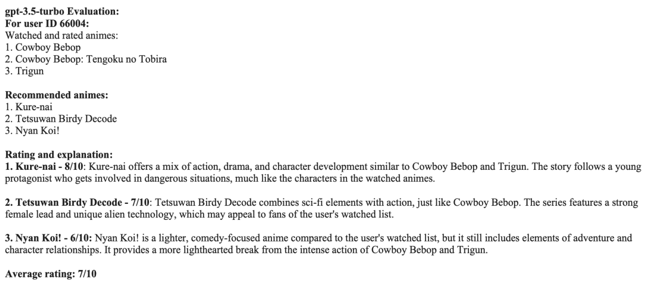
Method	Precision@k	MAP	NDCG
K-means	0.0254	0.0560	0.0911
Weight Average	0.0148	0.0803	0.0920

Table 1: Evaluation using different methods

These evaluation metrics may underestimate our model's effectiveness. This is because the goal of these metrics is to exactly match the evaluation set, which is extremely difficult in our case due to the large number of animes. However, users are likely to enjoy items beyond those they have already rated. To address this limitation, we adopt an alternative evaluation method that does not rely on an

[6]https://scikit-learn.org/1.5/modules/clustering.html#k-means
[7]https://scikit-learn.org/1.5/modules/mixture.html
[8]https://en.wikipedia.org/wiki/Elbow_method_(clustering)
[9]https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)
[10]https://en.wikipedia.org/wiki/Discounted_cumulative_gain

evaluation set to assess our recommendations.
In order to achieve the method described above and interpret recommendations from different perspectives, we apply the large language model (ChatGPT 3.5) to perform the evaluation. The anime that users have already watched and rated and the anime recommended by our recommendation system on this basis are used as input to the prompt of the language mode. According to the prompt instructions, the language model scores each of these recommended anime and provides a short explanation of them. From the responses, the reasonableness of the recommendations can be intuitively obtained.



gpt-3.5-turbo Evaluation:
For user ID 66004:
Watched and rated animes:
1. Cowboy Bebop
2. Cowboy Bebop: Tengoku no Tobira
3. Trigun

Recommended animes:
1. Kure-nai
2. Tetsuwan Birdy Decode
3. Nyan Koi!

Rating and explanation:
1. Kure-nai - 8/10: Kure-nai offers a mix of action, drama, and character development similar to Cowboy Bebop and Trigun. The story follows a young protagonist who gets involved in dangerous situations, much like the characters in the watched animes.

2. Tetsuwan Birdy Decode - 7/10: Tetsuwan Birdy Decode combines sci-fi elements with action, just like Cowboy Bebop. The series features a strong female lead and unique alien technology, which may appeal to fans of the user's watched list.

3. Nyan Koi! - 6/10: Nyan Koi! is a lighter, comedy-focused anime compared to the user's watched list, but it still includes elements of adventure and character relationships. It provides a more lighthearted break from the intense action of Cowboy Bebop and Trigun.

Average rating: 7/10

Figure 2: Evaluation using LLM

7 RESULTS AND DISCUSSION

First, we want to highlight the main differences between the two methods (described in Sections *5.2.1* and *5.2.2*): 1) In the weighted average approach, we calculate the distance to all animes in the data set to identify the closest one. In contrast, the Cluster-based approach only requires computing the distances to the k clusters. Once the nearest cluster is identified, we select some of the animes within that cluster. 2)The Weighted Average approach does not require any offline preprocessing, whereas the Cluster-based method necessitates precomputing the clustering for all the animes in the dataset.
To focus on our results, we will evaluate the trade-offs between accuracy and computational efficiency. In the performance analysis, the main computational cost lies in calculating the distances between the user and the anime entries.
For computational efficiency, the Weighted Average approach requires calculating N distances, where N represents the total number of animes in the dataset. In contrast, the Cluster-based method only requires computing k distances, where k is the number of clusters. Given that $N » k$, it is evident that the Cluster-based method is significantly more computationally efficient.
Both the weighted average and the clustering-based method need to compute the embedding vectors of all the anime for recommendation. Compared to weighted average methods, clustering-based methods need to complete the offline computation of clusters in advance. This makes weighted average methods simpler in the initial stage and more flexible, especially for scenarios where data changes frequently. While clustering methods reduce the computation in the recommendation phase by reusing the clustering results.
In the accuracy analysis, we hypothesize the cluster-based method, by comparing fewer animes, may impact accuracy to some extent.

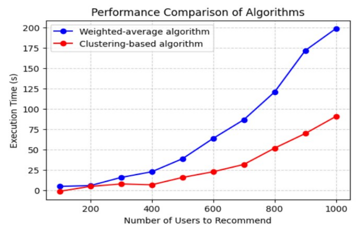


Figure 3: Performance Plot

To analyze this, we initially considered conducting *A/B testing* to determine user preferences. However, due to time constraints and the limited number of users available, we opted for an alternative approach. We used "*seen*" users to predict ratings for some of the animes they had already rated. We then computed the error for two groups: the first group included all the rated animes, while the second group consisted only of animes from the closest cluster to the user. The results showed that the accuracy within the cluster-based group was higher than when considering all animes. This suggests that the clustering approach is effective in terms of both accuracy and performance. However, it is important to note that using clustering may sometimes result in missing out on good recommendations that are not in the closest cluster.

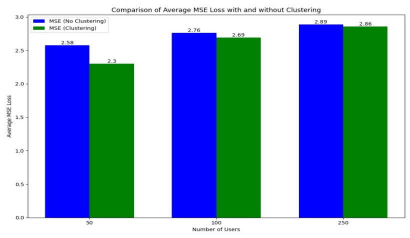


Figure 4: Comparision of Average MSE Loss with and Without Clustering

Ultimately, the trade-off between performance and accuracy depends on the intended use of the recommendation system. For large-scale systems where speed is crucial, the Cluster-based method is preferable. However, if precision in recommendations is a higher priority and performance is less of a concern, the Weighted Average method would be more suitable.

REFERENCES

[1] J. Cheng. 2023. Understanding the Two-Tower Model in Personalized Recommendation Systems. HackerNoon. https://hackernoon.com/understanding-the-two-tower-model-in-personalized-recommendation-systems Retrieved from HackerNoon.
[2] Harrie Oosterhuis and Maarten de Rijke. 2020. Policy-Aware Unbiased Learning to Rank for Top-k Rankings. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 489–498. https://doi.org/10.1145/3397271.3401102
[3] Scikit-learn developers. 2024. Clustering Methods. https://scikit-learn.org/1.5/modules/clustering.html Accessed: 2024-12-11.
[4] Yantao Yu, Weipeng Wang, Zhoutian Feng, and Daiyue Xue. 2021. A dual augmented two-tower model for online large-scale recommendation. *DLP-KDD* (2021).