DSCI 641 - Final Project

Final Project Report
Drexel University, College of Computing and Informatics
DSCI 641: Recommender Systems
Dr. Michael Ekstrand
March 19th, 2024

Group Member	Email
Hung Do	hd386@drexel.edu
Matt McKenna	mm5679@drexel.edu
Ao Wang	aw3338@drexel.edu

Problem Statement

The domain of our project is anime, a style of animation originating from Japan characterized by its colorful graphics, vibrant characters, and diverse themes that cater to a wide range of audiences. It encompasses various genres, from action and romance to science fiction and fantasy, and is available in formats like TV series, films, and web shorts.

Our intended users of the application are anime enthusiasts looking for new anime to watch that matches their tastes or preferences. This project aims to provide users with a personalized and intriguing anime discovery experience. With this goal in mind, we strive to increase user satisfaction and loyalty to our system while increasing sales and revenue to anime titles that users might not have seen before through growing exposure and visibility.

Overall, the primary problem this system addresses is discovering new anime that aligns with a user's preferences, especially given the vast and ever-growing library of anime content. Users may also seek to find anime that is similar to their favorites or popular among viewers with similar tastes.

If we were to build out a fully functional system with a user interface, like a web application, it would be capable of tracking which anime a user has watched. It would be similar to MovieLens, which offers "Non-commercial, personalized movie recommendations" (movielens.org), and our system could provide the same within the Anime domain.

The primary characteristic of our application’s effectiveness is relevance; the recommended anime should closely match the user's stated preferences or viewing history. Additionally, recommendations should have diversity, so that users are exposed to a variety of genres or themes to explore different anime styles.

Relevance is evaluated with metrics like Hit Rate, which measures how often a list of recommendations has a relevant item, and nDCG, which accounts for the rank ordering of the recommendations.

We would like to explore an additional measure of effectiveness: diversity. Anime users appreciate exploring multiple genres, and we would like our recommendations to offer a variety of genres rather than steer toward a single one.

All users will have the same experience: they can select and rate all the Anime movies and TV shows they wish to. Then, they will navigate to a page of recommendations showing 20 anime items will be displayed. Users will have the ability to add anime to a queue or to give a rating once they've been seen.

Overall, our well-designed anime recommendation system can enhance the viewing experience for all anime watchers by personalizing content discovery, educating users, fostering connections with the community, and keeping their to-watch lists fresh and exciting.

Data Description

Ideal Data

Firstly, we have user-provided data, which includes user preferences (genres, favorite titles, disliked titles), viewing history (titles watched, watch duration, re-watched titles), ratings and reviews (user ratings and text reviews for watched titles). This data is crucial for understanding individual tastes and preferences, the cornerstone of personalized recommendations.

To get an understanding of each anime's content, we would gather the title's metadata, such as the genre, director, studio, voice actors, and release year.

Finally, we will consider popularity metrics like view counts, likes, and social media mentions. This content data helps categorize anime and find similarities between titles, aiding in content-based filtering.

Moreover, we can add contextual data to our recommendation systems, such as temporal data (time of day, week, or year when users are active) and device data (such as mobile or TV). The most critical contextual data we can add is geographical data, as users from different regions have different popular titles.

All this data can be obtained from multiple sources:

1. We can gather the data through user profiles and interactions within the application.
2. We can collect it through partnerships with anime studios and streaming platforms for content data.
3. We might also collect data through public APIs for social media mentions and popularity metrics.
4. We can resort to surveys to gather detailed user preferences and feedback.

We would be mindful of obtaining this data ethically, while ensuring user privacy and consent. The goal is to create a system that understands not only what users like but also why they like it, leading to highly tailored and satisfying recommendations.

Our Data

For our project, we'll focus on anime datasets found on Kaggle. Both datasets are from the myanimelist.net API and contain information on user preference data. The first dataset from 2017 had 73,516 users for 12,294 anime. Users can add anime to their completed list and give it a rating.

The application takes two datasets: anime and ratings. The anime dataset contains all information about an anime, such as the full name, genre, number of episodes in the show, the type of anime it is, and the number of community members in myanimelist.net that are the anime's "group." The type of anime could be a movie, TV show, OVA (older anime before released on TV, but through DVDs, etc), and more. The genre column is a comma-separated list of anime genres. Additionally, each anime has a unique ID.

The ratings dataset contains all the users that have watched anime and the score they've given the item. There are three columns: the user ID, anime ID, and score. The score is between 1 to 10, where -1 means they've watched the anime, but didn't rate it.

Below are column-level summaries of each dataset.

Anime Dataset

- Anime_id: myanimelist.net's unique id identifying an anime
- Name: full name of anime
- Genre: comma separated list of genres for this anime
- Type: movie, TV, OVA, etc
- Episodes: how many episodes in this show. (1 if movie and "Unknown" if show is ongoing)
- Rating: average user rating (out of 10) of the anime
- Members: number of community members that are in this anime's "group"

Rating dataset

- User_id: non-identifiable randomly generated user id
- Anime_id: the anime that this user has rated
- Rating: rating out of 10 this user has assigned (-1 if the user watched it but didn't assign a rating)

Algorithms

Baseline Algorithms

Lift

In the context of our recommendation system, the lift algorithm helps in identifying titles that are more likely to be bought together than just by chance. In addition, it is particularly useful because it takes into account the popularity of individual items. It helps in avoiding misleading recommendations that could happen due to the high sale volume of an item rather than an actual association between the items.

In order to use it, we first have to create an instance of the algorithm, then we wrap the instance in a recommender and fit the algorithm on our training data. Here, it only considers items with at least 2 ratings. If an item only has 1 rating, it will have very high lift with the other items the user has rated, but on the basis of a single vote, and also be useless as a seed item for other users' recommendations. Then, a sparse matrix is made, where lift is computed using co-occurrences, individual item occurrence counts, and the total number of items. Finally, we use the eval dataset to generate recommendations.by looking up the user’s ratings and scoring the items using them.

Popular

The Lenskit Popular model is designed for non-personalized recommendation systems, particularly to recommend the most popular items to users. When the model is fit with the training data, it first starts with the popularity scoring, where the algorithm scores items based on their popularity. This scoring is typically measured by the frequency of ratings or interactions they receive from users.The PopScore class in LensKit implements this by scoring items with methods like 'quantile', 'rank', or 'count'. Then, the Popular class uses these popularity scores to recommend items. Finally, it recommends the most popular items that have not yet been rated by the user.

In essence, the LensKit Popular algorithm is a simple yet effective way to recommend items that are broadly appealing to a wide audience. It's particularly useful when personalization is not required or when there's insufficient data to build personalized models. Despite not being personalized, we think it’s an interesting model for our use case because of how the anime market can be driven by immensely popular titles.

Implicit MF

We used the Lenskit ImplicitMF model to explore the user and item relationships. We used 50 features to train.

This algorithm starts by representing the user-item interactions in a matrix form, where rows correspond to users and columns to items. The values in this matrix are derived from implicit feedback, which could be binary or based on interaction strength. Next is matrix factorization, where the core idea is to factorize the user-item matrix into 2 lower-dimensional matrices - one representing latent user preferences and the other representing latent item attributes. The latent factors are learned during the training process. Then, the algorithm uses a cost function that measures the difference between the observed user-item interactions and the interactions predicted by the latent factors. It goes on to apply an optimization technique, such as stochastic gradient descent, to minimize this cost function. Once the model is trained, it can predict the level of interest a user might have in an unseen anime by computing the dot product of the user’s and anime’s latent factors. The system then recommends the anime with the highest predicted interest scores to the user.

Item KNN with Implicit Feedback

The item-based KNN algorithm with implicit feedback is a collaborative filtering approach used in recommendation systems. It’s also particularly suited for situations where explicit ratings are sparse or unavailable, such as when recommending anime based on user viewing habits. The maximum neighborhood size is 20 and the k-value is 2.

In order to fit the model with the training data, the algorithm first computes similarities between items based on implicit feedback. Then, for each anime, the algorithm defines a ‘neighborhood’ of similar anime based on the similarity matrix. The prediction of a user’s interest in an anime they haven’t seen is computed by having the algorithm aggregate the implicit feedback from the user for the ‘k’ most similar anime. This aggregation could be a weighted sum where the weights are the similarity scores. Finally, the system recommends anime with the highest predicted interest scores to the user. These scores are based on the user’s implicit feedback on similar items and the item-item similarities.

Content-Based

Content-based filtering is a recommendation system technique that suggests items by comparing the content of the items and a user profile. The content here can include the items' descriptions, attributes, or features. Like in previous assignments, we've used movie tags or genres, authors, and subject headings for books. The user profile, on the other hand, captures the preferences or interests of a user, often derived from their past interactions with the items (e.g., ratings, views, purchases).

We used the anime genre for our project to recommend other anime they might enjoy. This recommendation system is primarily targeted towards our Anime Newbies and Single-Genre Fans since either they have a limited number of rated items or many items of the same genre. For example, if a new user has seen Haiykuu, a popular volleyball anime, and rated it highly, it would be beneficial for the system to recommend more highly rated volleyball or sports anime, as they'll likely enjoy that anime.

In the dataset, each anime genre is a list of words. We're using Term Frequency-Inverse Document Frequency (TF-IDF) to quantify the genres. TF-IDF helps quantify how crucial a genre is to anime relative to how common that genre is across all anime. For example, suppose an anime belongs to a rare genre, such as sports. In that case, that genre will receive a higher importance score for that particular anime, making it a defining characteristic of the anime.

Additionally, TF-IDF can improve the precision of recommendations by highlighting the uniqueness of genres in each anime. Anime with higher TF-IDF scores for specific genres will likely be more similar, allowing for more targeted and relevant recommendations. On the other hand, common genres appearing in many anime titles might not be instrumental in distinguishing between anime because they do not provide unique information. TF-IDF naturally downweights these genres, focusing on what makes each anime unique.

After calculating the TF-IDF vectors for each anime genre, we weighted them based on the users' ratings. As an example, given the scenario that a user watched *Kaguya-sama: Love Is War,* a popular romantic comedy, and rated it very high with a score of 9 and also watched *Junji Ito Collection*, a horror anime, rating it low with a score of 2, the recommender would weight the animes with comedy and romance higher and downweight animes with the horror genre, perhaps introducing the user to anime with a different genre.

PyTorch Model

The Two-Tower architecture has become a popular approach to building recommendation systems for its efficiency and effectiveness in handling large amounts of data. We believed it was particularly well-suited for this problem because of the amount of data available for each of the users and for each anime.

The first tower in this architecture is the “query tower,” which is intended to learn user interests. For us, user watch data formed the “query tower.” Specifically, the training dataset with the implicit feedback of which anime each user watched served this purpose.

The second tower is the “item tower.” This piece includes all available item features. For us, this was a genre. We used the attribute data to tag items to their multiple genres.

The two-tower architecture creates embedding layers for each tower, then takes a dot product to create a prediction as part of the model fit process. We used the torchtag model developed in Assignment 2 to create these embeddings and fit the model.

Experiments

For each of the above algorithms, we generated 20 recommendations. For our use case, we felt that longer lists would lose value as it would require more scrolling than a user is willing to do.

We split the data using the cross-fold partitioning method in the Lenskit package so that the training dataset was 90% of the over dataset, dev dataset was 5%, and eval set was 5%. The distribution was set so that each of the dev and eval would have a minimum of 5 items unseen to the training data for 20% of the user base. Our typical approach was to fit each model on the training dataset and evaluate on the eval set.

We chose three methods to evaluate the usefulness of these recommendations:

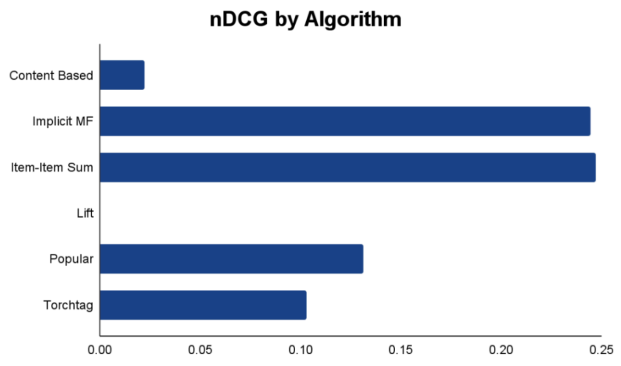
- Hit Rate to measure that our list produced at least one relevant recommendation
- nDCG to measure the quality of the lists' rank-ordering
- Diversity to measure how often a user was given an acceptable range of anime genres

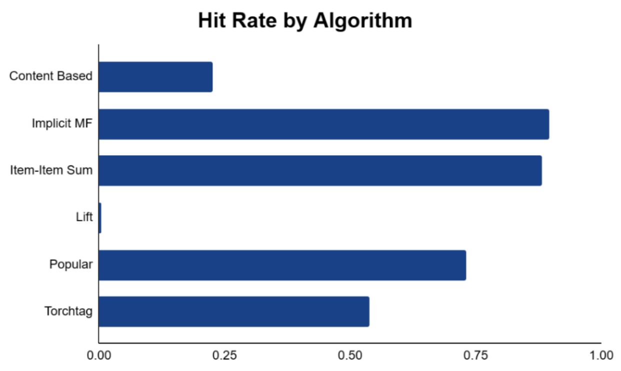
Hit Rate and nDCG were calculated with the Lenskit package. We devised our own way to compute diversity by determining any recommendation set that didn't have a dominant (over 75% of the recommendations) genre as diverse.

Torchtag is a neural network, which has more hyperparameters than our other models. For this model, we tuned the important hyperparameters; namely, number of features (dimensions) of the embedding, the model regularization, the learning rate, the number of training epochs.

We used nDCG of the model recommendations on the testing dataset to determine which hyperparameters produced the best performance. We found that a high number of embedding features (200) and training epochs (7) improved predictions, while the other variables didn't have major effects.

The experiment produced interesting results. The Implicit MF and the Item-Item Sum approaches produced the best nDCG and Hit Rate. Given that each had hit rates close to 90%, we feel that implementing either one would satisfy the use case.





Our diversity analysis was interesting, as well. As expected, our content-based strategy had barely any diversity (<1%). Popular and Lift methods had near perfect diversity, while Implicit MF, Item-Item Sum, and Torchtag had roughly 80-90% diversity. We found that recommendation sets that met the diversity threshold had slightly stronger hit rates than diverse sets.

Building and implementing this proposed application would be a challenge. Inputting and outputting static data would not be sufficient. We would want to think through:

- How to input newly-released animes into our datasets
- How to display recommendations in a more visual way than just the anime name
- How to incorporate the recency of a user-item interaction

Reflection

This project posed several interesting challenges. The first challenge was finding a domain and formulating a problem statement that we cared about solving. We found the anime domain to be interesting, and the outputs of our recommendation system were acceptable solutions to our problem statement. While the data - with some missing data and a limited number of fields was not ideal - it was clear and neat, as well as robust enough to power our analysis.

The second challenge was on the algorithms. We wanted to explore this problem from a number of angles that varied by complexity. Our best performing algorithms were ones that discovered item-item relationships, and methods that relied on content-related attributes were less predictive. However, these content approaches can be helpful in certain situations, such as a “cold start,” when a new item is introduced.

In conclusion, this project allowed us to put into practice the techniques learned in this class, as well as apply critical thinking in approaches to the business problem.