Anime Recommender System [TEAM-X]

THOMAS NIKOLOPOULOS, ORFEAS-DIMITRIOS TSELOS

January 28, 2024

Contents

Abstract

In this report, we delve into the development of a recommendation engine for anime content, leveraging collaborative filtering techniques, such as Singular Value Decomposition (SVD), Cosine Similarity and Neural Network Embeddings, to predict user preferences based on historical rating data.

The motivation behind this project stems from the increasing demand for personalized content suggestions, enhancing user experience in many aspects of our daily life. Our initial idea was to get involved with movies datasets due to the popularity of the cinema industry, especially with streaming services such as Netflix. After careful consideration, though, we concluded that due to this increased popularity there are many recommendation systems available online and that a lot of the other teams would also choose these datasets for their project. This is why, we chose to create a recommender system for anime as they are also popular and it's another form of Movies/TV-Shows recommendation.

This report provides insights into the motivation behind the recommendation system, the methodology employed for model development, and initial results obtained through evaluation metrics. The findings contribute to the ongoing exploration of personalized content recommendation systems, offering valuable implications for enhancing user engagement in anime streaming platforms.

1 Introduction

In streaming services, recommendation systems play a vital role in improving user experience by suggesting content tailored to individual preferences. These systems analyze what users watch, like, and dislike to offer personalized suggestions. They work by examining user behavior and content attributes to suggest shows or movies similar to what a user has enjoyed before. Techniques such as collaborative filtering, content-based analysis, and advanced machine learning help create these recommendations. By constantly learning from user interactions and feedback, these systems aim to provide better suggestions, ultimately enhancing user satisfaction and engagement on streaming platforms.

This report explores the application of collaborative filtering techniques. Collaborative filtering leverages the collective preferences of a user community to predict the preferences of individual users. The goal is to build a recommendation system that can accurately predict how a user would rate or interact with an item based on the preferences of users with similar tastes.

To achieve this, in this project, we tried to create our own recommendation system specifically for Anime Shows. We decided to use this Anime Dataset. This data set contains information on user preference data from `73,516 users on 12,294 anime`. Each user is able to add anime to their completed list and give it a rating and this data set is a compilation of those ratings.

The contents of the data set are the following.

Anime.csv

- **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).*
- **rating**: *average rating out of 10 for this anime.*
- **members**: *number of community members that are in this anime's "group".*

Rating.csv

- **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).

This system will utilize user ratings and watch-history to provide users with recommendations of anime to watch. Our methodology involves the integration of user ratings from a dataset with anime information, leading to the creation of a comprehensive dataset. The collaborative filtering model is trained on this dataset, allowing it to discern patterns and relationships between users and anime titles.

We will base our application on Jupyter Notebook and we will we use Python for development, utilizing its vast collection of machine learning libraries such as Pandas, Sci-kit, Surprise and many more.

2 Literature Review

Recommendation systems have been a focus of considerable research, with diverse approaches aiming to enhance the accuracy and efficiency of recommendations. In this review, we categorize existing literature into three main strategies: collaborative filtering through matrix factorization, with emphasis on Singular Value Decomposition (SVD), collaborative filtering based on item similarity, particularly utilizing Cosine Similarity and Neural Network Embeddings.

- **Collaborative Filtering with Matrix Factorization:** Matrix factorization techniques, notably Singular Value Decomposition (SVD), have garnered widespread attention in recommendation systems. Koren et al. (2009) *(which won the Netflix Grand Prize)* pioneered the application of SVD in collaborative filtering, emphasizing its capability to capture latent factors in the user-item interaction matrix. SVD decomposes the matrix into latent factors, providing a concise representation of user preferences and item characteristics.
- **Collaborative Filtering Based on Item Similarity** Item-based collaborative filtering relies on assessing the similarity between items to generate recommendations. Cosine Similarity, proposed by Sarwar et al. (2001), measures the cosine of the angle between item vectors, determining their similarity. This method is particularly effective in handling sparse datasets and showcasing the relationships between items. However, the choice of similarity metrics and scalability issues remain areas of exploration in the literature.
- **Neural Network Embeddings** An embedding involves the conversion of a discrete, categorical variable into a vector of continuous numbers. In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables. They prove advantageous by diminishing the dimensionality of categorical variables and effectively depicting categories in the transformed space. Neural network embeddings serve these key purposes: Identifying nearest neighbors in the embedding space, Providing recommendations based on user interests or clustering of categories, Serving as input to a machine learning model for supervised tasks and Enabling visualization of concepts and relationships between categories.

In exploring the landscape of anime recommendation systems, these approaches have provided valuable insights. The state-of-the-art involves a combination of collaborative filtering techniques, content-based methods and hybrid models that leverage the strengths of both. While collaborative filtering offers personalized recommendations, content-based methods enhance diversity and handle the cold-start problem. As for problems such as machine translation and handling categorical variables the embedding technique has proved to be a practical application with word embeddings and entity embeddings respectively. The literature indicates that successful recommendation systems often involve a delicate balance between these strategies, tailored to the characteristics of the dataset and user preferences.

3 Dataset and Features

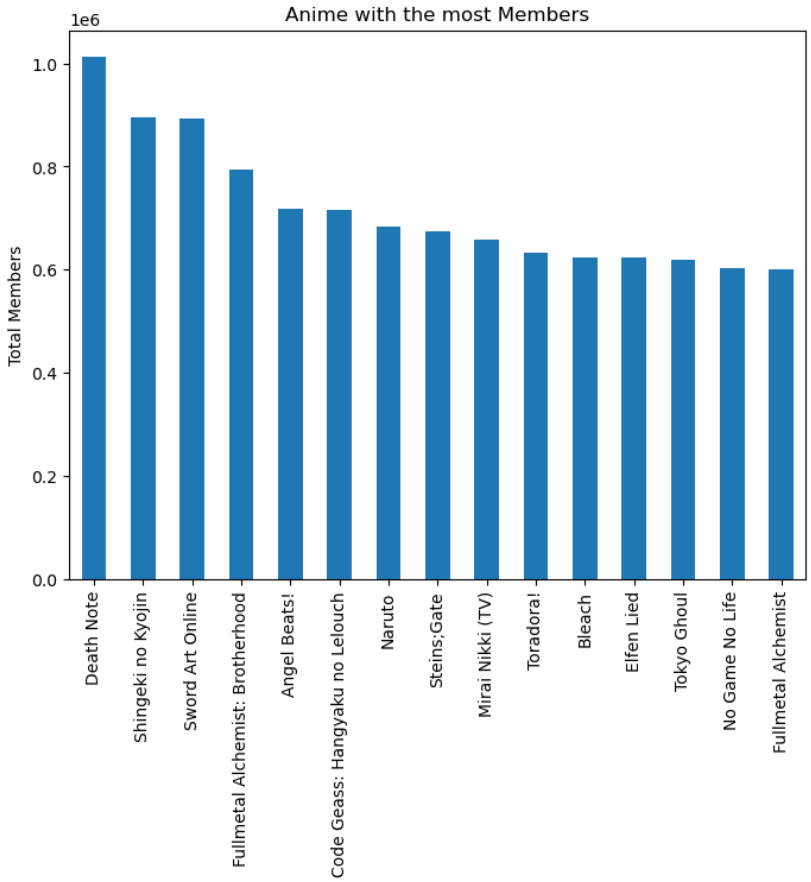
3.1 Feature Cleaning

First things first, we had to make sure that our data was nice and clean before we began to do anything with it. So we had to check and remove any null values or duplicate data. Luckily our data set was very clean from the get go and we did not have to do much. There were very few missing values from our data, so little in fact that we could just remove them entirely and not have any significant impact to our data.

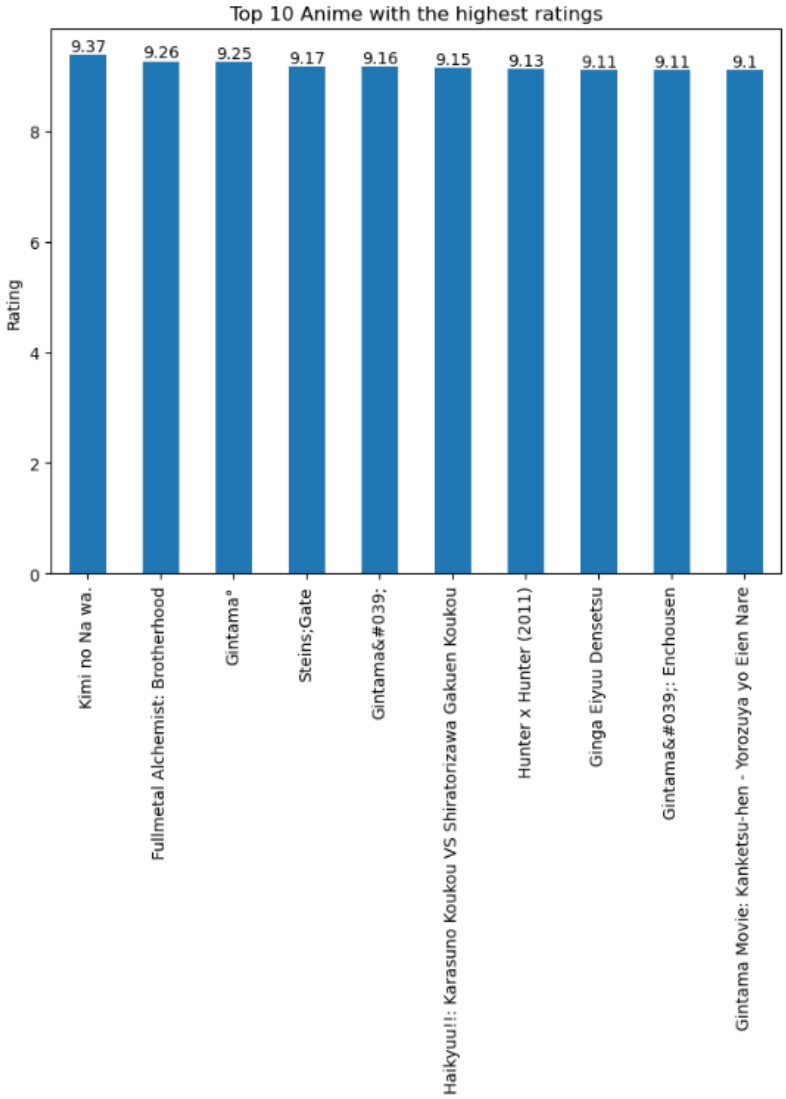
3.2 Exploratory Data Analysis

We proceed with Exploratory data analysis in order to visualize our data and help ourselves understand them better.

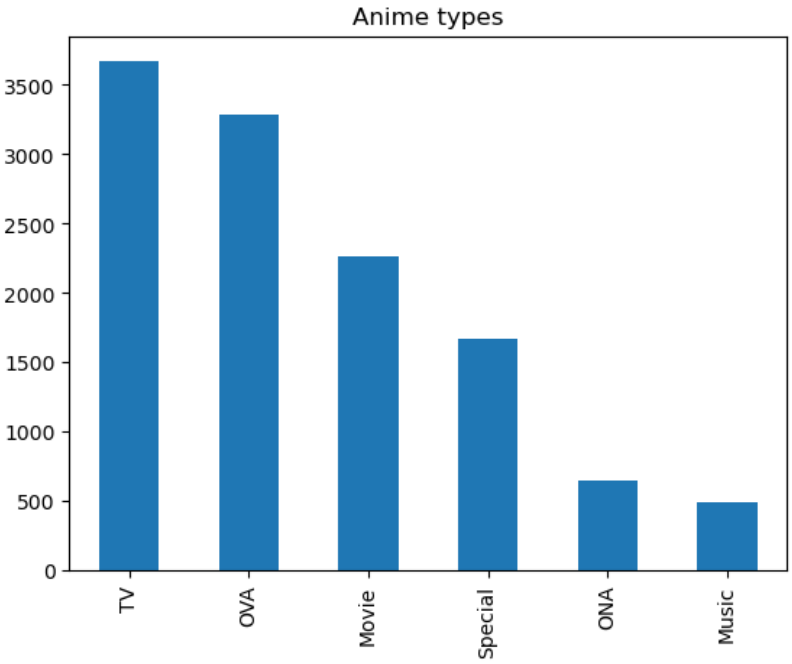
1. **Top Anime by member count**: This plot shows the Anime with the most amount of members in their community or in other words, it shows us the most popular Anime.



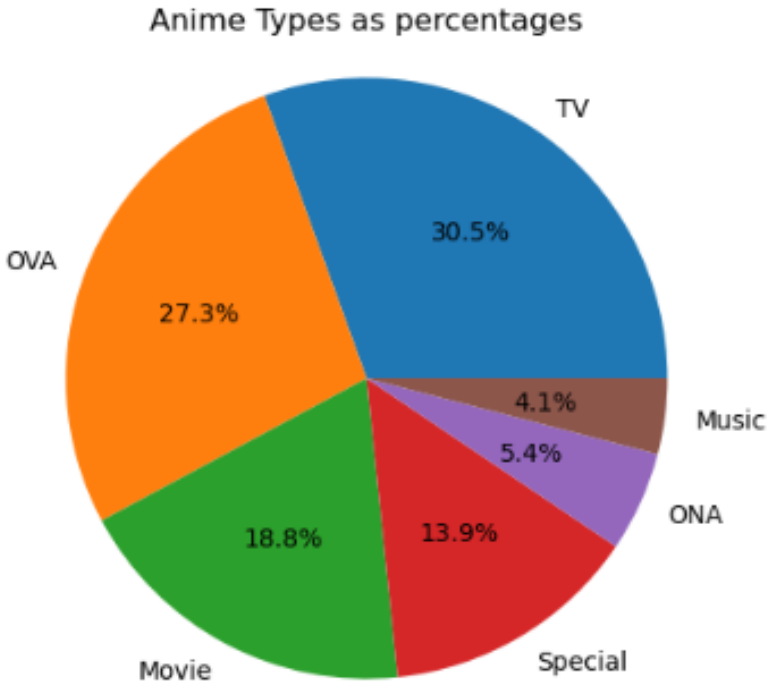
2. **Top Anime by rating**: These are the top 10 Anime based on their rating.



3. **The most popular types of Anime**

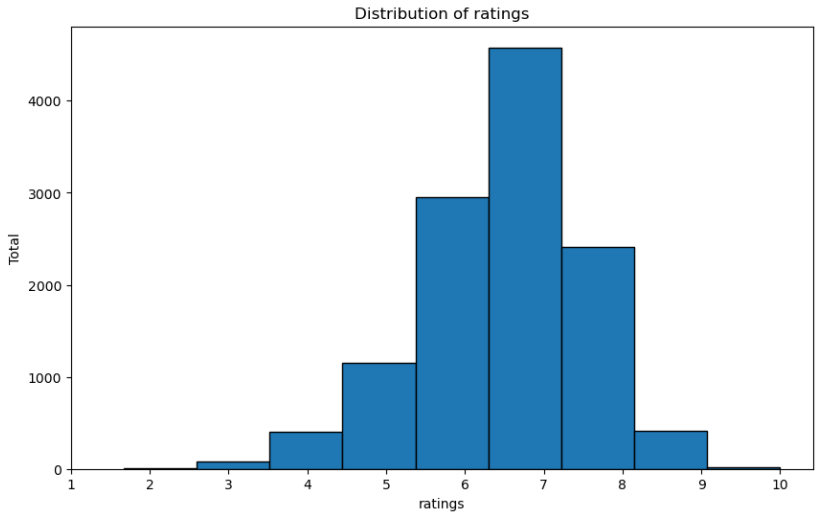


4. **And here is their distribution**



5. **The rating distribution**

rating	value
mean	6.478264
std	1.023857
min	1.670000
25%	5.890000
50%	6.570000
75%	7.180000
max	10.000000



3.3 Problems with our Dataset

Due to some trial and error during the development of the project we realized that the **ratings.csv** dataset is missing some anime_id's and so not every anime in our **anime.csv** dataset has ratings.

Dataset	unique anime_id's
anime.csv	12294
rating.csv	11200

While this does not pose any real threat to our recommendation engine since the differene is not significant, it does mean that there will be a few Anime that we will have no way of predicting/recommmending simply because there are no records of it.

3.4 Data Preparation for Machine Learning

By deafault the ratings dataset uses '-1' to represent missing ratings. We will be replacing these values with NaN so we can normalize the data and use cosine similarity in order to create our recommender system.

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

(a) Before

	user_id	anime_id	rating
0	1	20	NaN
1	1	24	NaN
2	1	79	NaN
3	1	226	NaN
4	1	241	NaN

(b) After

After that we have to merge our two datasets, anime and rating, in order for our algorithms to work. And out of the merged dataset we only have to keep the following columns: 'user_id', 'name', 'user_rating'

	user_id	name	user_rating
0	1	Naruto	NaN
1	3	Naruto	8.0
2	5	Naruto	6.0
3	6	Naruto	NaN
4	10	Naruto	NaN

In order to use cosine simalirty to find all the similarities between the anime shows we have to pivot our Data so our user ratings are on the columns and the anime names on the rows.

We also need to normalize our data so we can work with them. So, after this operation, each column (user ratings) will be normalized using Min-Max scaling based on its own mean, minimum, and maximum values.

Keep in mind that Min-Max scaling squeezes the values of each row to a range between 0 and 1.

$$\text{Scaled Value} = \frac{\text{Original Value} - \text{Min Value}}{\text{Max Value} - \text{Min Value}}$$

We then drop all users who did not rate anything to cut down on computational costs.

The final matrix looks like this:

user_id	3	5	7	8	10	11	12	14	16	17	...	2989	2990
name													
.hack//Roots	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0
.hack//Sign	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0
.hack//Tasogare no Udewa Densetsu	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0
009-1	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0
07-Ghost	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
gdgd Fairies 2	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	-0.099164	...	0.0	0.0

(a) Note* All users with only one rating or who had rated everything the same have been dropped due to Min-Max

Moreover we are transforming this matrix to a sparse matrix representation. This can be particularly useful when working with large datasets where most of the values are 0, as it saves memory compared to storing the entire dense matrix. Also certain algorithms and libraries, such as Scikit-learn, can handle sparse matrices more efficiently and thus, imnprove our computational time.

4 Methods

In this section, we provide an overview of the learning algorithms used for building the collaborative filtering recommendation system. The primary algorithms employed are **Singular Value Decomposition (SVD)**, a matrix factorization technique widely used for collaborative filtering in recommendation systems and **Cosine Similarity** which is a measure that quantifies the cosine of the angle between two vectors. In the context of our recommendation system, it is used to measure the similarity between users and items based on their preferences.

4.1 Singular Value Decomposition (SVD)

SVD is a matrix factorization technique employed to decompose the user-item interaction matrix R into three matrices: U, Σ, and V^T. Mathematically, this decomposition is represented as:

$$R = U\Sigma V^T$$

- U represents the user-feature matrix, capturing latent features characterizing user preferences.
- Σ is a diagonal matrix containing singular values, indicating the importance of each latent feature.
- V^T is the item-feature matrix, encapsulating latent features representing anime characteristics.

The prediction of missing or unrated entries is achieved through the product $U\Sigma V^T$. The iterative optimization process refines these factorized matrices to minimize the difference between predicted and observed ratings. Regularization terms are often introduced to prevent overfitting.

In our anime recommendation context, U and V^T embody user preferences and anime features, respectively. The latent features encapsulate patterns in user behaviors and anime characteristics, facilitating accurate predictions for missing ratings.

4.2 Cosine Similarity

Cosine Similarity offers a complementary approach to collaborative filtering by quantifying the similarity between user or item vectors based on their preferences. For two vectors A and B, the cosine similarity is computed as:

$$\text{cosine\_similarity}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

- $A \cdot B$ is the dot product of vectors A and B.
- $\|A\|$ and $\|B\|$ are the Euclidean norms of vectors A and B.

For user-based collaborative filtering, cosine similarity is calculated between the target user's preferences and other users' preferences. Similarly, for item-based collaborative filtering, it is computed between the target item's features and the features of other items. High cosine similarity values indicate a strong alignment in user preferences or item characteristics.

This comprehensive overview elucidates the foundational principles of both SVD and Cosine Similarity within our collaborative filtering recommendation system. The subsequent sections will delve into implementation specifics, model evaluations, and the results obtained through the application of these methodologies to our anime dataset.

4.3 Implementation using Surprise Library for SVD

To implement the Singular Value Decomposition (SVD) approach, we leveraged the Surprise library, a powerful Python library designed for building and evaluating recommendation systems. Surprise provides a user-friendly interface for implementing collaborative filtering algorithms, including SVD.

The Surprise library facilitates the entire recommendation system pipeline, from loading datasets and defining rating scales to training models and making predictions. For our SVD-based collaborative filtering model, we used the Surprise's `SVD` class, which encapsulates the SVD algorithm and associated functionalities.

Here's a brief overview of how we integrated the Surprise library into our recommendation system:

- **Data Loading:** We used the `Dataset.load_from_df()` method to load the anime rating data into the Surprise Dataset. This method efficiently handles the conversion of user-item-rating data into a format suitable for training collaborative filtering models.
- **Model Training:** The `SVD` class provides a straightforward interface for training the SVD model. We utilized the `fit()` method to train the model on our dataset. The library handles the optimization process, adjusting the factorized matrices to minimize the prediction error.

- **Prediction:** Making predictions for user-item pairs is simplified with the `predict()` method. Given a user ID and an anime ID, the model estimates the rating that the user would assign to the anime.
- **Evaluation:** Surprise includes various evaluation metrics to assess the performance of recommendation models. We utilized metrics such as Root Mean Squared Error (RMSE) to evaluate the accuracy of our SVD-based model.

The Surprise library streamlines the implementation of collaborative filtering algorithms, providing a flexible and efficient framework for building recommendation systems. Its simplicity and integration with popular algorithms make it a valuable tool for researchers and practitioners in the field of recommender systems.

4.4 Implementation with Cosine Similarity

For the implementation of this approach, we didn't rely on a specific library like Surprise. Instead, we developed a custom solution using standard Python libraries such as NumPy, scikit-learn and SciPy.

The process unfolded through the following key steps:

- **Data Preparation:**
  - We organized the anime dataset to create a user-item matrix, where rows represented users, columns represented anime titles, and entries corresponded to user ratings. Like we showed in section 3.4.
- **Item-Based Cosine Similarity:**
  - For item-based cosine similarity, we transposed the user-item matrix, treating each column as a vector representing an anime title.
  - Similar to the user-based approach, we calculated the cosine similarity between the target anime's vector and the vectors of all other anime titles.
- **Recommendation Generation:**
  - Based on the computed cosine similarities, we identified users or items with the highest similarity to the target user or item.
  - Recommendations were generated by selecting items that had high cosine similarity with the target, suggesting similar user preferences.

To manage the potentially large and sparse user-item interaction matrix, we utilized the `csr_matrix` class from SciPy. This class efficiently represents sparse matrices in Compressed Sparse Row (CSR) format. The resulting sparse matrix representation is memory-efficient and particularly suitable for collaborative filtering scenarios, where many user-item interactions are not explicitly recorded.

By combining the functionality of scikit-learn for cosine similarity calculations and SciPy for sparse matrix representation, we achieved a scalable and efficient solution for handling user-item interactions in our collaborative filtering recommendation system.

name	.hack//Roots	.hack//Sign	.hack//Tasogare no Udewa Densetsu	009-1	07-Ghost	11eyes	12-sai.: Chicchana Mune no Tokimeki	3 Choume no Tama: Uchi no Tama Shirimasenka?	30-sai no Hoken Taiiku	91 Days	...	Zombie-Loan
name												
.hack//Roots	1.000000	0.221592	0.280241	-0.062339	0.050119	0.005539	0.000000	0.000000	0.105405	-0.063973	...	0.045173
.hack//Sign	0.221592	1.000000	0.283246	0.086160	0.047388	0.021872	0.000000	0.000000	0.048897	-0.070921	...	-0.002167
.hack//Tasogare no Udewa Densetsu	0.280241	0.283246	1.000000	0.262065	0.029278	0.008328	0.000000	0.000000	0.011485	-0.018164	...	0.102779
009-1	-0.062339	0.086160	0.262065	1.000000	-0.026273	0.013174	0.000000	0.000000	0.000000	0.000000	...	0.004628
07-Ghost	0.050119	0.047388	0.029278	-0.026273	1.000000	0.135311	0.004813	-0.017066	0.110150	-0.066043	...	0.094195
...	...	...	...	...	...	...	...	...	...	...	...	...
gdgd Fairies 2	0.000000	0.000000	0.000000	0.000000	0.000115	0.001426	0.000000	0.000000	0.000000	-0.020977	...	-0.001606

Figure 4: This is the final matrix with all the cosine values computed. The higher the value, the stronger the correlation.

4.5 Keras Embeddings Implementation

For the implementation of the Embeddings technique, we first imported useful libraries in Google Collab such as scikit-learn and tensorflow-keras. Apart from an important feature cleaning, this implementation required the following stages:

- **Multiprocessing:**
  - We determined the amount of CPU cores available. This stage is useful as it allows us to use all cores for the training of the model to decrease runtime.
- **Input, Embedding, Flatten layers:**
  - The input layer in a neural network is represented as $X_{\text{in}} = [x_1, x_2, ..., x_t]$, where t is the sequence length. This can be expressed as:

    $$X_{\text{in}} = [x_1, x_2, ..., x_t]$$

    The embedding layer converts categorical data into continuous vectors. Let $E(X_{\text{in}}) = [e_1, e_2, ..., e_t]$ be the output of the embedding layer. This can be represented as:

    $$E(X_{\text{in}}) = [e_1, e_2, ..., e_t]$$

    Mathematically, the embedding operation can be denoted as:

    $$E(X_{\text{in}}) = \text{Embedding}(X_{\text{in}}, \text{input\_dim} = V, \text{output\_dim} = D)$$

    where V is the vocabulary size and D is the dimensionality of the embedding. The flatten layer reshapes a 2D tensor Y into a flat vector Z. If Y has dimensions $(\text{batch\_size}, m, n)$, then Z is a flat vector with dimensions $(\text{batch\_size}, m \times n)$. This can be expressed as:

    $$Z = \text{Flatten}(Y)$$

    Mathematically, the flatten operation is denoted as:

    $$Z = \text{Flatten}(Y)$$

    where Y is the output of the preceding layer.
- **Dot Product:**
  - To combine the embeddings using a dot product, let's consider the dot product of the corresponding elements in the embeddings:

    $$\text{Combined Embedding} = e_{11} \cdot e_{21} \cdot \ldots \cdot e_{k1},\ e_{12} \cdot e_{22} \cdot \ldots \cdot e_{k2},\ \ldots,\ e_{1t} \cdot e_{2t} \cdot \ldots \cdot e_{kt}$$

    Mathematically, the combined embedding using a dot product can be expressed as:

    $$\text{Combined Embedding} = E_1(X_{\text{in}_1}) \cdot E_2(X_{\text{in}_2}) \cdot \ldots \cdot E_k(X_{\text{in}_k})$$

    This dot product operation captures relationships and interactions between different categorical features. In summary, when combining embeddings using a dot product, each element in the combined embedding is obtained by multiplying the corresponding elements from individual embeddings. This technique is useful for capturing interactions between different categorical features in a neural network.
- **Model Creation, Training and Evaluation**
  - The model is then compiled with the Adam optimizer and mean squared error loss function. The data, is then split into training and testing sets using an 80-20 split ratio. Subsequently, the model is trained on the training set for 5 epochs with a batch size of 64 and a validation split of 20%. After training, the model is evaluated on the test set. The Root Mean Squared Error (RMSE) between the actual ratings and predictions is calculated and printed as a measure of the model's performance on the test set.

Following the stages above (code used found in the Embeddings notebook), we finally have implemented a model ready to use for recommendations. We will review recommender implementation in section 5.3.

5 Experiments/Results/Discussion

In this section, we detail the experiments conducted to evaluate our anime recommendation system, presenting both quantitative and qualitative results.

5.1 SVD Implementation

For our first experiment we used **SVD** in order to predict what a specific user would like to watch next based on his previews history and other users like him. The parameters, including the number of latent factors for SVD, were left as they are because they are already optimized for accuracy and efficiency by the Surprise library. The details of the methods we used are described in Section 4.3 and also on the Jupyter Notebook.

In order to extract the information from our processed data we have created the `get_user_recommendations()` function which takes two parameters: `user_id` and `anime_name` and outputs the top 5 Anime this specific user is most likely to like, along with their ratings. The way the function works is explained below:

1. **Finding Anime ID:**
   - It searches the `anime_data` DataFrame for an anime whose name contains the provided `anime_name` (case-insensitive).
   - It retrieves the corresponding `anime_id` of the found anime.
2. **Selecting Unrated Anime:**
   - It filters out anime that the user has already rated for the given `anime_id`.
3. **Generating Recommendations:**
   - For the unrated anime, it generates a list of tuples containing `(anime_id, estimated_rating)` for the specified `user_id`.
   - The estimated ratings are obtained using the collaborative filtering model (`model.predict`).
4. **Sorting and Selecting Top Recommendations:**
   - It sorts the list of recommendations based on the estimated ratings in descending order.
   - It selects the top 5 anime names from the sorted recommendations.

Here is an example output of this recommender engine:

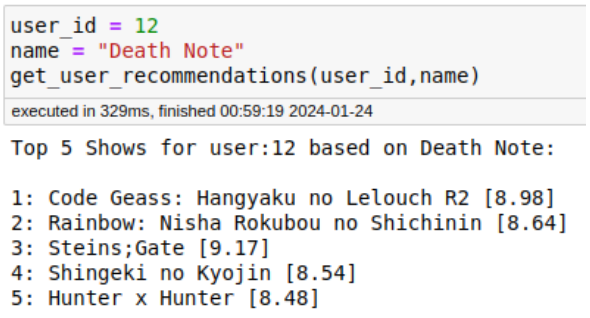


Figure 5: The value in the brackets is the rating of the show

5.1.1 SVD over KNN

During the development of our project we experimented with the **k-Nearest Neighbors (KNN)** for collaborative filtering. However we noticed that this algorithm is too slow for our data and the results were not as promising as with SVD. After some research we realised that SVD is more suitable for this problem. Here are the main reasons:

- **Data Sparsity:** SVD is often more suitable for sparse datasets, where many users have not rated many items. It can more effectively handle missing values in the user-item interaction matrix. KNN on the other hand, might struggle with sparsity because it relies on finding similar users or items based on a limited number of common ratings.
- **Diversity of Recommendations:** SVD might provide more diverse recommendations as it captures latent factors that might not be apparent from the raw ratings. KNN tends to recommend items that are more directly similar based on user-item interactions, potentially leading to less diverse recommendations.

5.2 Cosine Similarity Implementation

For our second experiment we computed the Cosine Similarity of all the items in the dataset. In this case, the anime titles. The details of the process are explained in Section 4.4 and also in the Jupyter Notebook. In order to extract the information from our processed data we have created the `recommend_anime()` function which takes one parameter: `name` and outputs the top 10 most similar Anime along with their ratings. The way the function works is explained below:

1. The function takes a name
2. It uses the `anime_sim_df`, which is used to store the similarity values, and sorts it in descending order based on the values for the given name.
3. It iterates through the top 10 shows (excluding the first one, as it would be the anime itself) and prints the show's name along with its rating

Here is an example output of this recommender engine:

```
recommend_anime('Death Note')
executed in 18ms, finished 01:03:41 2024-01-24
Top 10 Shows based on Death Note:

1: Code Geass: Hangyaku no Lelouch R2 [8.98]
2: Code Geass: Hangyaku no Lelouch [8.83]
3: Shingeki no Kyojin [8.54]
4: Fullmetal Alchemist: Brotherhood [9.26]
5: Steins;Gate [9.17]
6: Hunter x Hunter (2011) [9.13]
7: One Punch Man [8.82]
8: Kiseijuu: Sei no Kakuritsu [8.59]
9: Boku dake ga Inai Machi [8.65]
10: Great Teacher Onizuka [8.77]
```

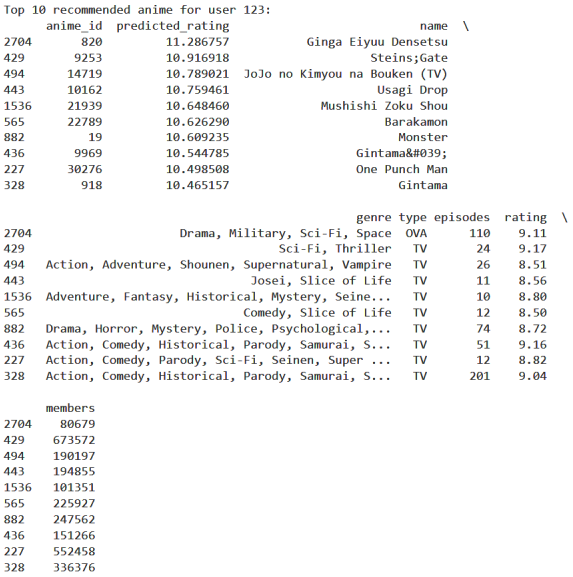
Figure 6: The value in the brackets is the rating of the show

5.3 Results of the Embeddings Model

Having implemented the Embedding Model, we can now use it to create a fully operational Recommender System. To achieve this we follow these steps:

1. We create a recommender function designed for generating personalized anime recommendations tailored to a specific user through collaborative filtering. Taking parameters such as the user ID, collaborative filtering model, anime mapping, anime dataset, and the desired number of recommendations (N), the function employs the model to predict ratings for the user across all available anime.
2. The results are then organized into a DataFrame, incorporating anime IDs, predicted ratings, and additional information from the anime dataset.
3. The recommendations are then sorted in descending order based on predicted ratings, and the top N recommendations are extracted.

4. Lastly, the function is called using custom values for the user's id and the number of anime to be recommended. Below we see the `10` anime that were recommended to user `123` using the embedding model and the function described above as well as information about them:



Top 10 recommended anime for user 123:

	anime_id	predicted_rating	name	genre	type	episodes	rating	members
2704	820	11.286757	Ginga Eiyuu Densetsu	Drama, Military, Sci-Fi, Space	OVA	110	9.11	80679
429	9253	10.916918	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
494	14719	10.789021	JoJo no Kimyou na Bouken (TV)	Action, Adventure, Shounen, Supernatural, Vampire	TV	26	8.51	190197
443	10162	10.759461	Usagi Drop	Josei, Slice of Life	TV	11	8.56	194855
1536	21939	10.648460	Mushishi Zoku Shou	Adventure, Fantasy, Historical, Mystery, Seine...	TV	10	8.80	101351
565	22789	10.626290	Barakamon	Comedy, Slice of Life	TV	12	8.50	225927
882	19	10.609235	Monster	Drama, Horror, Mystery, Police, Psychological,...	TV	74	8.72	247562
436	9969	10.544785	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266
227	30276	10.498508	One Punch Man	Action, Comedy, Parody, Sci-Fi, Seinen, Super ...	TV	12	8.82	552458
328	918	10.465157	Gintama	Action, Comedy, Historical, Parody, Samurai, S...	TV	201	9.04	336376

Figure 7: Anime recommendations for user 123

This collaborative filtering mechanism exploits the learned relationships between users and anime, offering personalized suggestions to enhance the user experience.

6 Conclusion/Future Work

In conclusion, our Anime Recommender engine, leveraging SVD and Cosine Similarity, demonstrated notable success in capturing user preferences. SVD, with its latent feature modeling, outperformed other methods like KNN, and Cosine Similarity enhanced item-based collaborative filtering.

The successful implementation of embeddings in our model marks a significant milestone, with several advantages for our machine learning application. By efficiently mapping categorical variables to continuous vectors, the model exhibits improved training efficiency and generalization capabilities.

For future work, fine-tuning SVD hyperparameters, exploring real-time adaptability, and incorporating sentiment analysis from user reviews could further refine recommendations. Hybrid models, combining collaborative and content-based filtering, offer a promising avenue.

7 Contributions

Both members of the team worked hard and contributed to the final form of the project. More specifically Thomas Nikolopoulos (3085) worked on the SVD and Cosine Similarity algorithms while Orfeas - Dimitrios Tselos (2963) worked on the Keras Embeddings Implementation notebook on Google Collab.

References

[1] Kaggle - Anime Recommendations Database

[2] Kaggle - Collaborative Filtering On Anime Data

[3] FreeCodeCamp - How to Build a Movie Recommendation System Based on Collaborative Filtering

[4] Matrix Factorization Techniques for Recommender Systems

[5] The BellKor Solution to the Netflix Grand Prize - Yehuda Koren - August 2009

[6] Item-based collaborative filtering recommendation algorithms - Badrul Sarwar,George Karypis, Joseph Konstan, John Riedl - May 2001

[7] Neural Network Embeddings Explained - Will Koehrsen - Oct 2, 2018

[8] What are embeddings in machine learning? - Cloudflare

[9] Embedding - Wikipedia, the free Encyclopedia