**Anime Recommendation System**

AA 2024/2025

Sistemi Intelligenti per Internet

Professor: Giuseppe Sansonetti

Pablo Matanza Barahona - Matricola: 633616

Félix Alonso Fraguas – Matricola: 633190

**Table of Contents:**

1. Summary……………………………………………………………………(pg 3)
2. Chosen Programming Paradigm…………………………………(pg 3)
3. Tools and Libraries Used…………………………………………….(pg 3)
4. Scalability Considerations…………………………………………..(pg 4)
5. Limitations………………………………………..……………………….(pg 5)
6. Performance Analysis…………………………………………………(pg 5)
7. Code Explained…………………………………………………………..(pg 6)
8. Conclusion……………………………………………………………….(pg 12)
9. External Links…………………………………………………………..(pg 12)

**1. Summary**

This project is a basic item-based recommendation system for anime, built using user rating data. It works by comparing which users have watched and rated the same anime shows, and then recommends similar anime based on these shared preferences. To measure how similar two anime shows are, the system uses the Jaccard similarity, which is a method that checks how many users liked both shows.

The data includes information about anime titles and user ratings. The project focuses on cleaning and organizing the data, creating a user-item matrix, calculating similarities between anime, and showing the final recommendations in a clear table.

Everything is coded in Python, using libraries like ‘pandas’, ‘numpy’, and ‘seaborn’ for handling data and creating graphs. While it doesn’t use machine learning, this method is easy to understand and a good starting point.

**2. Chosen Programming Paradigm**

The project follows the imperative programming paradigm, using procedural programming as its main structure. This means that the code is organized in functions that perform specific tasks, and the program runs step by step, following a clear sequence of instructions.

We use functions to handle tasks like reading data, cleaning it, creating useful data structures, and generating recommendations. This makes the code easier to read, maintain, and reuse. Although the code doesn’t use object-oriented programming, its modular design helps keep things organized and avoids repetition.

This approach was chosen because it fits well with data processing tasks and allows full control over each step of the recommendation process.

**3. Tools and Libraries Used:**

This project was developed using Python because of its simplicity and the big range of libraries it offers for data analysis and visualization.

Here are the main tools and libraries used:

* **Pandas:** For reading, cleaning, and manipulating the data. It helps us create DataFrames and perform operations like filtering and grouping.
* **NumPy:** Used for some numerical operations and efficient data handling.
* **Seaborn:** A visualization library used to plot the distribution of user ratings.
* **JSON:** To save and load dictionaries that store the mapping between anime names and their IDs.
* **Jupyter Notebook:** Used during the development process to explore data, test functions, and display results in an interactive way.
* **DefaultDict** (from collections): Used to create efficient data structures for user-item relationships.

These tools made it easier to build the recommendation system and analyze the data in a flexible and organized way.

**4. Scalability Considerations:**

While the project works well with a limited dataset, scaling it to larger amounts of data would require several changes and improvements.

Currently, the recommendation system uses a Jaccard similarity approach that compares items by checking which users interacted with them. This method is simple and easy to understand, but it can become slow and inefficient when working with millions of users or items.

Some scalability limitations are:

* **Memory Usage:** The entire dataset is loaded into memory, which may not be possible with very large files.
* **Time Complexity:** Calculating the similarity between one item and many others takes a lot of time, especially as the number of users and items increases.
* **No Optimization Techniques:** The current implementation does not use indexing, parallel computing, or dimensionality reduction to speed up the process.

To make the system scalable for larger datasets, we could:

* Use more efficient similarity measures.
* Implement approximate nearest neighbors (as we have seen in class) algorithms.
* Switch to a model-based approach (like deep learning).
* Store and process the data using big data tools like Apache.

These improvements would allow the system to handle more users, more animes, and faster recommendations.

**5. Limitations:**

This project, while being functional and useful for small to medium datasets, has several limitations that should be taken in count in the future to improve its performance and usability.

Limitations:

* **Data Size:** The current approach loads all data into memory, which limits the size of datasets that can be handled.
* **Performance:** The similarity calculation is done using basic Jaccard similarity without any optimizations, making it slow for large datasets.
* **Cold Start Problem:** New users or new animes with few ratings have limited recommendations, since the system relies on existing user-item interactions.
* **Limited Features:** Only user ratings and anime IDs are used. Other useful information like genres, tags, or user profiles are not considered.
* **Lack of Real-Time Updates:** The system does not update recommendations dynamically when new data arrives.

**6. Performance Analysis:**

The performance of the recommendation system was evaluated based on some information that we searched:

* **Data Loading:** Reading and merging the datasets is efficient for moderate-sized data (up to 100,000 ratings). The use of Pandas ensures fast data manipulation but can consume significant memory with larger datasets.
* **Similarity Calculation:** The Jaccard similarity function computes set intersections and unions, which are straightforward but can become slow when the number of users and items grows. The current implementation calculates similarity on-demand without caching results, leading to repeated computations for popular items.
* **Recommendation Generation:** Retrieving the most similar items involves iterating over all items rated by users who rated the target anime. For small datasets, this is fast; however, scalability issues arise when the number of users or items increases substantially.
* **Memory Usage:** Storing user-item interactions in dictionaries and Pandas DataFrames requires considerable memory, limiting scalability.

Overall, the system performs well for educational purposes and small datasets, but optimizing algorithms and data structures is necessary to handle larger datasets or real-time recommendations effectively.

**7. Code Explained:**

**Utils.py**



**read\_data():** This function reads the raw data files containing anime information and user ratings. It uses the Pandas library to load two CSV files:

* anime.csv: Contains details about each anime.
* rating.csv: Contains user ratings for different animes.

After loading these files into separate dataframes, it returns them for future processing.

**get\_dataframe():** This function processes the data loaded by read\_data()

It calls read\_data() to get the anime and ratings dataframes.

It cleans the ratings by replacing any rating value of -1 with 0. This is important because a -1 rating usually means "no rating" or missing data, so replacing it with zero allows easier handling in later steps.

It merges the ratings dataframe with the anime dataframe using the anime\_id as the key. This adds anime details to each rating entry. It shuffles the combined dataframe randomly to avoid any ordering bias during analysis or training.

Finally, it returns this merged and shuffled dataframe for use in the rest of the project.

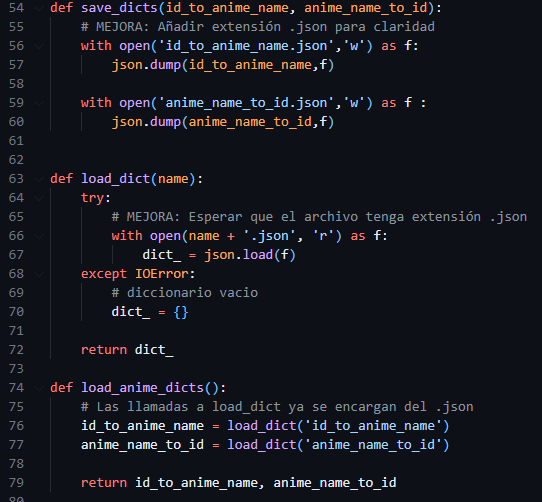


**get\_rating\_matrix():** This function prepares the user-item rating matrix used for analysis:

* It gets the merged dataframe from get\_dataframe().
* It selects only the necessary columns: user IDs, anime IDs, ratings, and anime names.
* It filters out animes with fewer than 1000 ratings to focus on popular titles.
* It limits the data to 100,000 entries for performance reasons.
* It creates a matrix where rows are users, columns are animes, and values are the ratings (missing ratings filled with 0).
* Returns this matrix for further use.

**plot\_rating\_distribution():** This function creates and saves a plot showing how ratings are distributed:

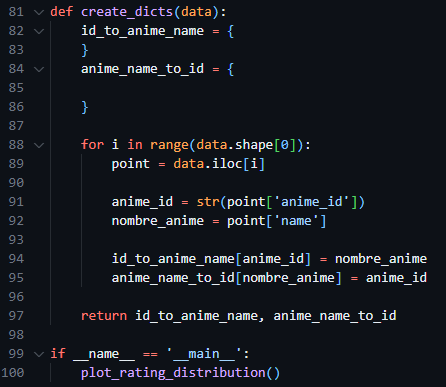
* It loads the merged dataframe.
* Uses Seaborn to plot a count of each rating value.
* Saves the plot as an image file.

****

**save\_dicts():** saves two dictionaries to JSON files. It takes a dictionary that maps anime IDs to names and another that maps anime names to IDs, and writes them into separate files with a .json extension. This ensures the data is stored in a clear and accessible format.

**load\_dict**():loads a dictionary from a JSON file given its base name. It tries to open the file with a .json extension and read its contents. If the file does not exist or cannot be opened, it returns an empty dictionary instead, preventing errors during loading.

**load\_anime\_dicts():** calls load\_dict twice to load both the ID-to-name and name-to-ID dictionaries. It returns these two dictionaries so they can be used later in the program.

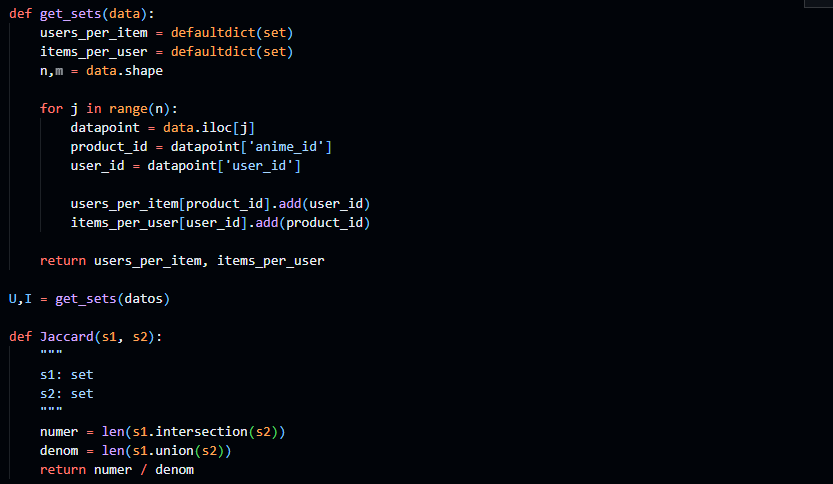


**create\_dicts():** builds two dictionaries from the given data. It starts with two empty dictionaries: one to map anime IDs to their names, and another to map anime names back to their IDs. Then it goes through each row of the data. For each entry, it extracts the anime ID and the anime name. The ID is converted to a string for consistency. These values are added to both dictionaries: the ID-to-name dictionary stores the name using the ID as the key, and the name-to-ID dictionary stores the ID using the name as the key.

At the end, both dictionaries are returned for use elsewhere in the program.

The last part, inside the block **if \_\_name\_\_ == '\_\_main\_\_':** runs the function plot\_rating\_distribution() when the script is executed directly, generating a plot that shows how ratings are distributed.

**Untitled.ipynb**

****

**get\_sets():** takes a dataset and creates two dictionaries that store relationships between users and items. It uses defaultdict with sets to keep track of unique entries. One dictionary, users\_per\_item, maps each item (anime) to the set of users who rated it. The other, items\_per\_user, maps each user to the set of items they rated.

To build these dictionaries, the function loops through each row of the data. For every record, it extracts the anime\_id and the user\_id. Then, it adds the user to the set associated with that anime in users\_per\_item, and also adds the anime to the set for that user in items\_per\_user.

Finally, it returns both dictionaries for use later in the program.

**Jaccard()**: calculates the Jaccard similarity between two sets. It measures how similar the sets are by dividing the size of their intersection by the size of their union. This value ranges from 0 (no similarity) to 1 (identical sets).

****

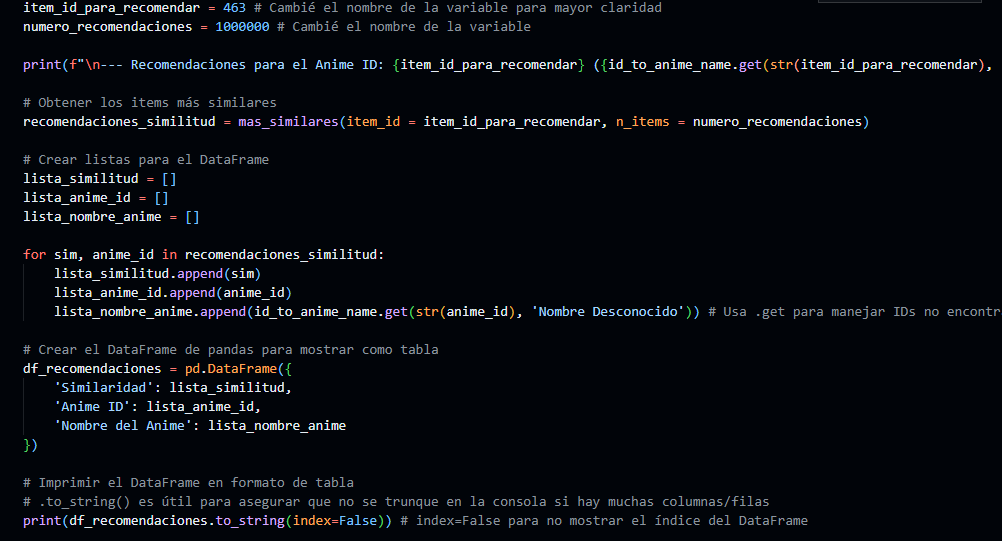
**mas\_similares()** finds the most similar items to a given item based on user overlap. It takes two inputs: item\_id, which is the identifier of the item you want recommendations for, and n\_items, the number of similar items to return.

First, it gets the set of users who rated the target item from the dictionary U. Then, it collects all the items rated by these users into a set called items. This way, it gathers candidate items that might be similar to the target item.

Next, the function loops through each candidate item. It skips the original item itself to avoid recommending it again. For each candidate, it calculates the Jaccard similarity between the users who rated the target item and the users who rated the candidate item. This similarity score measures how much their audiences overlap.

It stores each similarity score together with the item ID in a list called similares. After processing all candidates, it sorts this list in descending order by similarity, so the most similar items come first.

Finally, the function returns the top n\_items most similar items as recommendations.



This code section is designed to generate and display recommendations for an anime based on similarity.

First, you can select the anime item you want recommendations for by setting the variable item\_id\_para\_recomendar. Some example anime IDs are provided as comments, such as Naruto (20) or Fullmetal Alchemist (5114). In this case, the selected anime ID is 463. The variable numero\_recomendaciones sets how many recommendations you want to get. It is set to a very large number here to get as many recommendations as possible.

The code then prints a message showing the selected anime ID and its name. It looks up the anime name using the dictionary id\_to\_anime\_name. If the name is not found, it shows "Nombre no encontrado" (Name not found).

Next, the function mas\_similares is called to find the most similar anime items to the selected one. It returns a list of pairs containing the similarity score and the anime ID.

To organize the results, three lists are created to store similarity scores, anime IDs, and anime names. The code loops through the recommended items, appending each similarity and ID to the respective lists. It looks up each anime name using the dictionary, defaulting to "Nombre Desconocido" (Unknown Name) if not found.

Then, a pandas DataFrame is created with these lists, giving a structured table of similarity, anime ID, and anime name.

Finally, the DataFrame is printed as a table to the console. The to\_string() method is used to ensure the full table is shown without truncation, and the index column is hidden for cleaner output.

**8. Conclusion:**

In this project, we developed a recommendation system for anime based on user ratings and similarity measures. We used the Jaccard similarity to find items that share similar audiences, allowing us to suggest anime that users might enjoy based on their previous preferences. The data processing steps ensures the quality of input by filtering popular anime and organizing the data into usable formats.

Although the system performs well in providing relevant recommendations, there are limitations. For example, it relies only on user ratings and does not consider other factors such as genre or content features. Future improvements could include integrating more complex algorithms, incorporating additional data sources, and optimizing for scalability with larger datasets.

Overall, this project demonstrates how simple collaborative filtering techniques can be effectively applied to create useful recommendations, providing a strong foundation for further development.

**9. External Links:**

Anime database: <https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database>

GitHub: <https://github.com/Pablo0-mb/Anime-Recommendation-System>