University of Macau

澳門大學

UNIVERSIDADE DE MACAU

UNIVERSITY OF MACAU

CISC3014

MULTIMEDIA COMPUTING

Project

DC227268, DIEGO HUANG WU

DC325448, Cheong Kin Ngai

2024

Table of content

Introduction

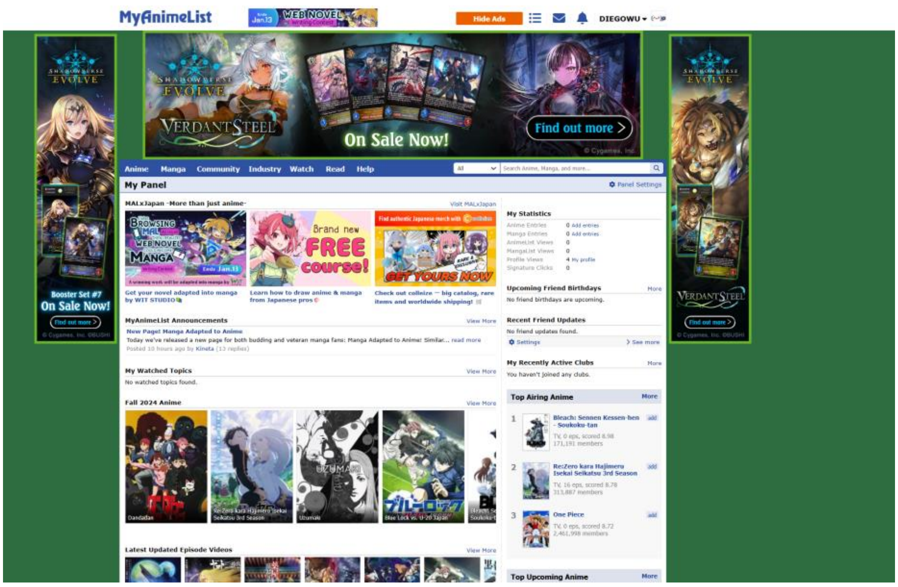
Anime has become a significant cultural phenomenon, especially in East Asia, with a vast and ever-growing catalog. However, finding anime that aligns with individual preferences remains a challenge due to the subjective nature of taste.

This project aims to develop a personalized anime recommendation system using user ratings and anime attributes. By leveraging machine learning techniques, it seeks to enhance the discovery process and contribute to research in content recommendation systems.

Our report, along with the full project files, can be accessed on our GitHub repository at the following link: [GitHub - MALSugoi: A recommendation system for anime based on MyAnimeList ratings].

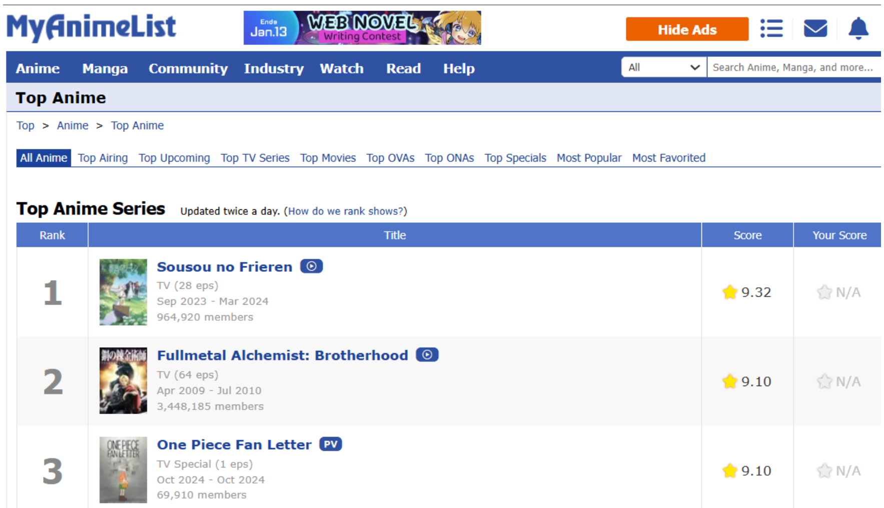
Background

The data for this project is sourced from MyAnimeList (MAL), a widely-used anime and manga social cataloging platform. MAL allows users to organize, rate, and review anime and manga, providing a rich dataset for building personalized recommendation systems.

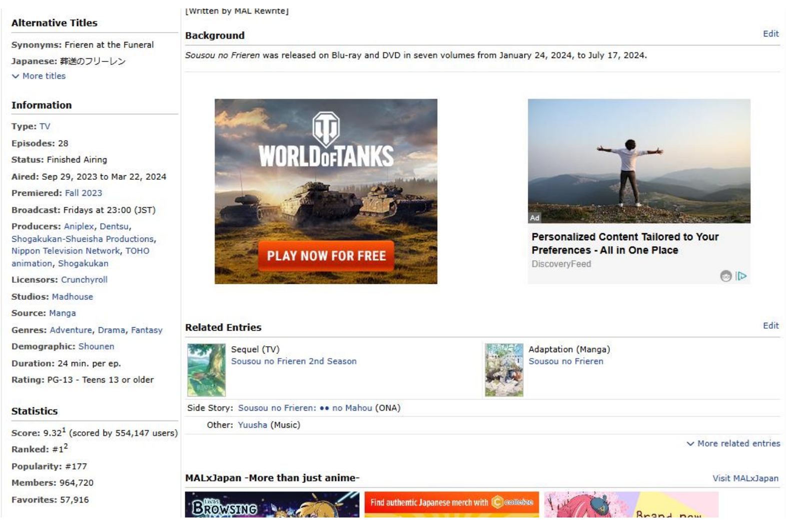


The dataset used to train and develop the model

1. Anime Data(3000 animes approximately for training model)



Anime Series



Anime data

Code

```python
# Scrape detailed anime info from individual anime pages
def get_anime_details(browser, anime_url):
    anime_info = {}
    try:
        print(f"Accessing details page: {anime_url}")
        browser.get(anime_url)

        WebDriverWait(browser, 10).until(
            EC.presence_of_element_located((By.CLASS_NAME, "anime-detail-header-stats"))
        )

        page_source = browser.page_source
        soup = BeautifulSoup(page_source, 'html.parser')

        score_tag = soup.find('span', {'itemprop': 'ratingValue'})
        anime_info['details_score'] = score_tag.text.strip() if score_tag else 'N/A'

        ranked_tag = soup.find(string='Ranked:')
        anime_info['ranked'] = ranked_tag.parent.next_sibling.strip() if ranked_tag and ranked_tag.parent else 'N/A'

        popularity_tag = soup.find(string='Popularity:')
        anime_info['popularity'] = popularity_tag.parent.next_sibling.strip() if popularity_tag and popularity_tag.parent else 'N/A'

        members_tag = soup.find(string='Members:')
        anime_info['members'] = members_tag.parent.next_sibling.strip() if members_tag and members_tag.parent else 'N/A'

        favorites_tag = soup.find(string='Favorites:')
        anime_info['favorites'] = favorites_tag.parent.next_sibling.strip() if favorites_tag and favorites_tag.parent else 'N/A'

        genres_tag = soup.find_all('span', {'itemprop': 'genre'})
        genres = [genre.text.strip() for genre in genres_tag]
        anime_info['genres'] = ', '.join(genres) if genres else 'N/A'

    except Exception as e:
        print(f"Failed to load details page: {e}")
        anime_info = {
            'details_score': 'N/A',
            'ranked': 'N/A',
            'popularity': 'N/A',
            'members': 'N/A',
            'favorites': 'N/A',
            'genres': 'N/A'
        }

    return anime_info
```

We collect anime data from top-ranked series, followed by detailed information obtained through web scraping. The code implementation can be found in anime_info_scraper.py. Below is a **snippet** of our anime information scraper code.

Here is a snippet of our anime information. For more details, refer to anime_data.csv. The dataset includes attributes such as title, score, genres, ranked, popularity, members, and favorites.

```
title,score,genres,ranked,popularity,members,favorites
Sousou no Frieren,9.32,"Adventure, Drama, Fantasy, Shounen",#1,#187,"941,889","56,180"
One Piece Fan Letter,9.16,"Action, Adventure, Fantasy, Shounen",#2,#3026,"57,604","1,564"
Steins;Gate,9.07,"Drama, Sci-Fi, Suspense, Psychological, Time Travel",#4,#14,"2,633,363","192,236"
Fullmetal Alchemist: Brotherhood,9.09,"Action, Adventure, Drama, Fantasy, Military, Shounen",#3,#3,"3,437,111","229,464"
```

2. User Data (3000 animes approximately for training model)

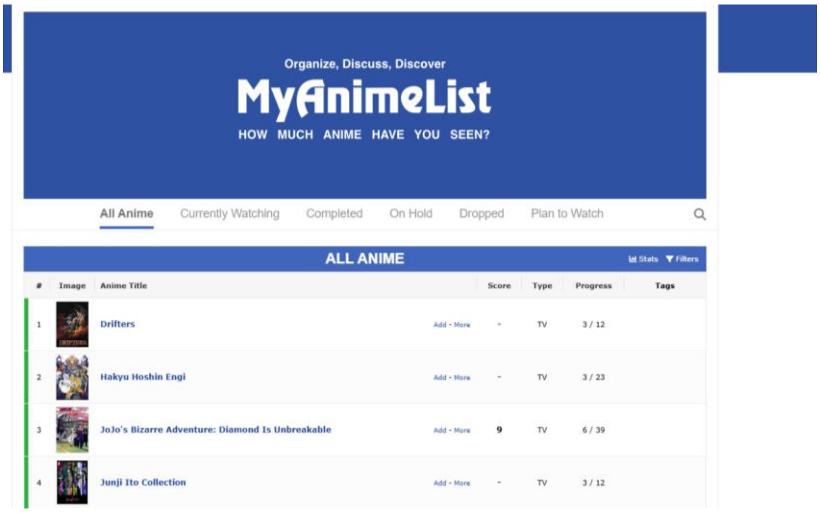
We scraped data from approximately 1,000 users. However, after filtering out ghost accounts, the final dataset includes around 600 users.

Code

We randomly selected users aged between 18 and 90. Gender and location were disregarded, as many users provide fictitious locations, and our project focuses solely on user preferences rather than demographic attributes.The full implementation can be found in user_animelist_info_scraper.py. Below is a snippet of our user-animelist scraper code.

```python
def scrape_usernames(base_url, num_pages):
    """
    Scrape usernames from multiple pages.

    Args:
        base_url (str): The base URL to scrape from.
        num_pages (int): The number of pages to scrape.

    Returns:
        list: A combined list of all extracted usernames.
    """
    all_usernames = []

    for i in range(num_pages):
        # Calculate the `show` parameter for pagination
        offset = i * 24

        # Construct the URL for the current page
        if i == 0:
            page_url = f"{base_url}"
        else:
            page_url = f"{base_url}&show={offset}"

        # Fetch the HTML code
        html_code = fetch_html(page_url)

        # Check if fetch was successful
        if not html_code.startswith("An error occurred"):
            # Extract usernames and add them to the list
            all_usernames.extend(extract_usernames(html_code))
        else:
            print(f"Failed to fetch page {i + 1}: {html_code}")

    return all_usernames
```

To proceed, we retrieved each user's anime list, formatted similarly to the sample shown in the image below:



User Anime List

```python
def extract_anime_info(html_code):
    """
    Extracts the anime title and score from the user's anime list page HTML.

    Args:
        html_code (str): The HTML content of the user's anime list page.

    Returns:
        list: A list of tuples where each tuple contains (anime_title, score).
    """
    soup = BeautifulSoup(html_code, 'html.parser')
    anime_info = []
    # Find all rows in the anime list table
    rows = soup.find_all('tr', class_='list-table-data')

    for row in rows:
        # Locate the container with title information
        title_container = row.find('td', class_='data title clearfix')
        if title_container:
            # Locate the anime title within the container
            title_tag = title_container.find('a', class_='link sort')
            anime_title = title_tag.text.strip() if title_tag else "Unknown"
        else:
            anime_title = "Unknown"

        # Extract the score
        score_container = row.find('td', class_='data score')
        score = (
            score_container.find('span', class_='score-label').text.strip()
            if score_container and score_container.find('span', class_='score-label')
            else "-"
        )

        # Append the extracted info as a tuple (anime_title, score)
        anime_info.append((anime_title, score))

    return anime_info
```

Code to extract the data

```
indou,The Beast Player Erin,9
indou,The Irresponsible Captain Tylor,9
indou,The Twelve Kingdoms,9
indou,The Vision of Escaflowne,10
indou,Toward the Terra,7
indou,Toward the Terra (TV),10
indou,Trigun,7
indou,TWO-MIX: White Reflection,8
indou,Vampire Princess Miyu,7
indou,Violinist of Hamelin,7
indou,Voices of a Distant Star,7
indou,X,7
indou,His and Her Circumstances,9
indou,Planetes,8
indou,Scrapped Princess,-
indou,Ah! My Goddess,6
indou,Damekko Doubutsu,6
indou,InuYasha,6
indou,Kimba the White Lion,6
indou,Kodocha,8
```

Above is an example of the user-anime information, consisting of three columns: username, animename, and score. For the complete dataset, please refer to anime_info.csv.

Collaborative filtering (item-item)

Based on the scraped user-anime list, we constructed a user-anime rating matrix. This matrix represents the ratings each user has assigned to different anime, serving as the foundation for our collaborative filtering model.

```
# 我们先处理出来我们的rating matrix
csv_file_path = r'data/user_animelist/anime_info.csv'
df = pd.read_csv(csv_file_path)
matrix = df_to_matrix(df)
rating_matrix = process_matrix(matrix)
print(rating_matrix)
```

```
                                          13thokazaki  AAelpaso  AHKO  AMVP  \
"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi         0.0       0.0   0.0   0.0
"Bungaku Shoujo" Memoire                          0.0       0.0   0.0   0.0
"Bungaku Shoujo" Movie                            0.0       0.0   0.0   0.0
'Tis Time for "Torture," Princess                 0.0       0.0   0.0   0.0
.hack//G.U. Returner                              0.0       0.0   0.0   0.0
...                                               ...       ...   ...   ...
xxxHOLiC◆Kei                                      0.0       0.0   0.0   0.0
Ōoku: The Inner Chambers                          0.0       0.0   0.0   0.0
∀ Gundam                                          0.0       0.0   0.0   0.0
∀ Gundam I: Earth Light                           0.0       0.0   0.0   0.0
∀ Gundam II: Moonlight Butterfly                  0.0       0.0   0.0   0.0

                                          Abraxus  Absolom  Ainsley  \
"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi     0.0      0.0      0.0
"Bungaku Shoujo" Memoire                      0.0      0.0      0.0
"Bungaku Shoujo" Movie                        0.0      0.0      0.0
'Tis Time for "Torture," Princess             0.0      0.0      0.0
.hack//G.U. Returner                          0.0      0.0      0.0
...                                           ...      ...      ...
xxxHOLiC◆Kei                                  0.0      0.0      0.0
Ōoku: The Inner Chambers                      0.0      0.0      0.0
∀ Gundam                                      0.0      0.0      0.0
∀ Gundam I: Earth Light                       0.0      0.0      0.0
∀ Gundam II: Moonlight Butterfly              0.0      0.0      0.0
...
∀ Gundam I: Earth Light                        0.0          0.0
∀ Gundam II: Moonlight Butterfly               0.0          0.0

[5915 rows x 601 columns]
```

user-anime rating matrix

```python
def df_to_matrix(df):
    data = []
    users = set()
    animes = set()

    for index, row in df.iterrows():
        user, anime, rating = row['username'], row['anime'], row['rating']
        # 将评分为 '-' 的情况直接设置为 0
        if rating == '-':
            rating = 0  # 将字符串 '0' 传递给 convert_to_int
        # 使用 convert_to_int 函数转换评分
        rating = convert_to_int(rating)
        data.append((user, anime, rating))
        users.add(user)
        animes.add(anime)

    users = sorted(list(users))
    animes = sorted(list(animes))

    matrix = pd.DataFrame(np.zeros((len(animes), len(users))), index=animes, columns=users)

    for user, anime, rating in data:
        matrix.at[anime, user] = rating

    return matrix.astype(int)  # 确保所有评分都是整数
```

code for calculating rating matrix.

Implementation details

Here we use Pearson correlation as similarity,.

1) subtract mean rating m_i from each movie i. We subtract average for calculating cosine similarity.

```python
def process_matrix(matrix):
    # 创建一个空列表来存储每行的处理结果
    results = []

    # 遍历矩阵的每一行
    for index, row in matrix.iterrows():
        count = 0  # 记录正整数的数量
        row_sum = 0  # 记录这一行正整数的总和

        # 遍历每一行中的每个元素
        for value in row:
            if value > 0:  # 只处理正整数
                count += 1
                row_sum += value

        # 计算 sum / count
        if count > 0:  # 避免除以零
            divide_result = row_sum / count
        else:
            divide_result = 0  # 如果没有正整数，则除以零时返回0

        # 对每个元素进行操作：如果是正整数，除以 divide_result；如果是0，保持不变
        new_row = []
        for value in row:
            if value > 0:
                new_row.append(value / divide_result)  # 除以 result
            else:
                new_row.append(value)  # 如果是0，保持原样

        # 将新的行添加到结果列表
        results.append(new_row)

    # 返回一个新的 DataFrame，包含处理后的矩阵
    result_matrix = pd.DataFrame(results, columns=matrix.columns, index=matrix.index)
    return result_matrix
```

2) Compute the similarity matrix for animes.

```python
def parallel_cosine_similarity(matrix):
    num_rows = matrix.shape[0]
    similarity_matrix = np.zeros((num_rows, num_rows), dtype=float)

    def calculate_similarity(i):
        row_a = matrix.iloc[i, :].values.copy()  # 确保是可写的
        row_similarities = []
        for j in range(num_rows):
            row_b = matrix.iloc[j, :].values.copy()
            sim_value = cosine(row_a, row_b)
            row_similarities.append(sim_value)
        return row_similarities

    # 使用并行计算获取每一行的相似度
    results = Parallel(n_jobs=-1)(delayed(calculate_similarity)(i) for i in range(num_rows))

    # 填充相似度矩阵
    for i in range(num_rows):
        for j in range(num_rows):
            similarity_matrix[i, j] = results[i][j]

    return pd.DataFrame(similarity_matrix, index=matrix.index, columns=matrix.index)
def cosine(a, b):
    norm_a = np.linalg.norm(a)
    norm_b = np.linalg.norm(b)

    # 如果其中一个向量的范数为0，返回相似度为0（可以根据需求调整）
    if norm_a == 0 or norm_b == 0:
        return 0.0

    cos_sim = np.dot(a, b) / (norm_a * norm_b)
    return cos_sim
```

```
sim = parallel_cosine_similarity(rating_matrix)
print(sim)
```

```
                                          "Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi  \
"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi                                  1.000000
"Bungaku Shoujo" Memoire                                                   0.551202
"Bungaku Shoujo" Movie                                                     0.616530
'Tis Time for "Torture," Princess                                          0.000000
.hack//G.U. Returner                                                       0.271279
...                                                                             ...
xxxHOLiC◆Kei                                                               0.000000
Ōoku: The Inner Chambers                                                   0.000000
∀ Gundam                                                                   0.166478
∀ Gundam I: Earth Light                                                    0.000000
∀ Gundam II: Moonlight Butterfly                                           0.000000

                                          "Bungaku Shoujo" Memoire  \
"Bungaku Shoujo" Kyou no Oyatsu: Hatsukoi                 0.551202
"Bungaku Shoujo" Memoire                                  1.000000
"Bungaku Shoujo" Movie                                    0.564721
'Tis Time for "Torture," Princess                         0.000000
.hack//G.U. Returner                                      0.000000
...                                                            ...
xxxHOLiC◆Kei                                              0.000000
Ōoku: The Inner Chambers                                  0.000000
∀ Gundam                                                  0.179124
∀ Gundam I: Earth Light                                   0.000000
∀ Gundam II: Moonlight Butterfly                          0.000000
...
∀ Gundam I: Earth Light                                          0.0
∀ Gundam II: Moonlight Butterfly                                 0.0

[5915 rows x 5915 columns]
```

Similarity matrix (similarity between two animes)

3) We predict the user's rating for an anime which the user hasn't rated based on this formula.

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

```
Click to add a breakpoint mmendation(similarity_matrix, matrix, top_n=5000, user_index=None, anime_index=None):
    """
    计算用户对动漫的推荐评分 r_ix。

    参数:
    - similarity_matrix: 动漫之间的相似度矩阵 (DataFrame)
    - matrix: 用户对动漫的评分矩阵 (DataFrame)
    - top_n: 考虑的与动漫 i 相似的前 N 个动漫 (int)
    - user_index: 用户的索引 (str)
    - anime_index: 当前动漫的索引 (str)

    返回:
    - 推荐评分 r_ix (float)
    """

    # 检查 anime_index 是否在 rating_matrix 中
    if anime_index not in matrix.index:
        print(f"动漫 {anime_index} 不在评分矩阵中。")
        return 0  # 或者返回一个合适的默认值

    # 检查 user_index 是否在 rating_matrix 中
    if user_index not in matrix.columns:
        print(f"用户 {user_index} 不在评分矩阵中。")
        return 0  # 或者返回一个合适的默认值

    # 获取用户对该动漫的评分
    user_rating = matrix.loc[anime_index, user_index]  # 使用 .loc

    # 如果评分不为 0，直接返回该评分
    if user_rating != 0:
        return user_rating

    # 提取动漫 i 的相似度和用户的评分
    sim_scores = similarity_matrix.loc[anime_index, :].values  # 使用 .loc
    ratings = matrix.loc[:, user_index].values  # 使用 .loc，获取用户的评分

    # 找到与动漫 i 最相似的前 N 个动漫的索引
    similar_indices = np.argsort(sim_scores)[-top_n:][::-1]

    # 计算推荐评分 r_ix
    numerator = 0  # 分子
    denominator = 0  # 分母
```

4) We get a recommendation list for a user based on rating in ascending order

```python
def calculate_all_recommendations(similarity_matrix, rating_matrix, user_index, top_n=100):
    """
    计算用户对所有动漫的推荐评分，并选出前 N 部评分最高的动漫。

    参数:
    - similarity_matrix: 动漫之间的相似度矩阵 (DataFrame)
    - rating_matrix: 用户对动漫的评分矩阵 (DataFrame)
    - user_index: 用户的索引 (str)
    - top_n: 返回前 N 部评分最高的动漫 (int)

    返回:
    - 推荐动漫及其评分 (DataFrame)
    """
    n_animes = rating_matrix.shape[0]  # 动漫数量
    recommended_ratings = []

    for anime_index in rating_matrix.index:  # 使用行名 (动漫名称)
        rating = item_based_recommendation(similarity_matrix, rating_matrix, user_index=user_index, anime_index=anime_index)
        recommended_ratings.append((anime_index, rating))

    # 创建 DataFrame 存储推荐评分
    ratings_df = pd.DataFrame(recommended_ratings, columns=['Anime Name', 'Predicted Rating'])

    # 选出评分最高的前 N 部动漫
    top_recommendations = ratings_df.sort_values(by='Predicted Rating', ascending=False).head(top_n)

    return top_recommendations
```

```python
matrix = calculate_all_recommendations(sim, test_matrix, u)
print(matrix)
```

```
                                Anime Name  Predicted Rating
618                                Berserk         10.000000
5743                           Wolf's Rain         10.000000
1657                                  FLCL         10.000000
1576                       Eden of The East        10.000000
2778                                 Karas         10.000000
...                                    ...               ...
1785                   Flying Phantom Ship          8.652643
3161  Lupin III Episode 0: The First Contact        8.597436
1232                      Dance with Devils          8.593071
4404                 Resident Evil 4D Executor      8.564154
2312                      Heroman Specials          8.564154

[100 rows x 2 columns]
```

Our test example

5) Evaluating our collaborative filtering model

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

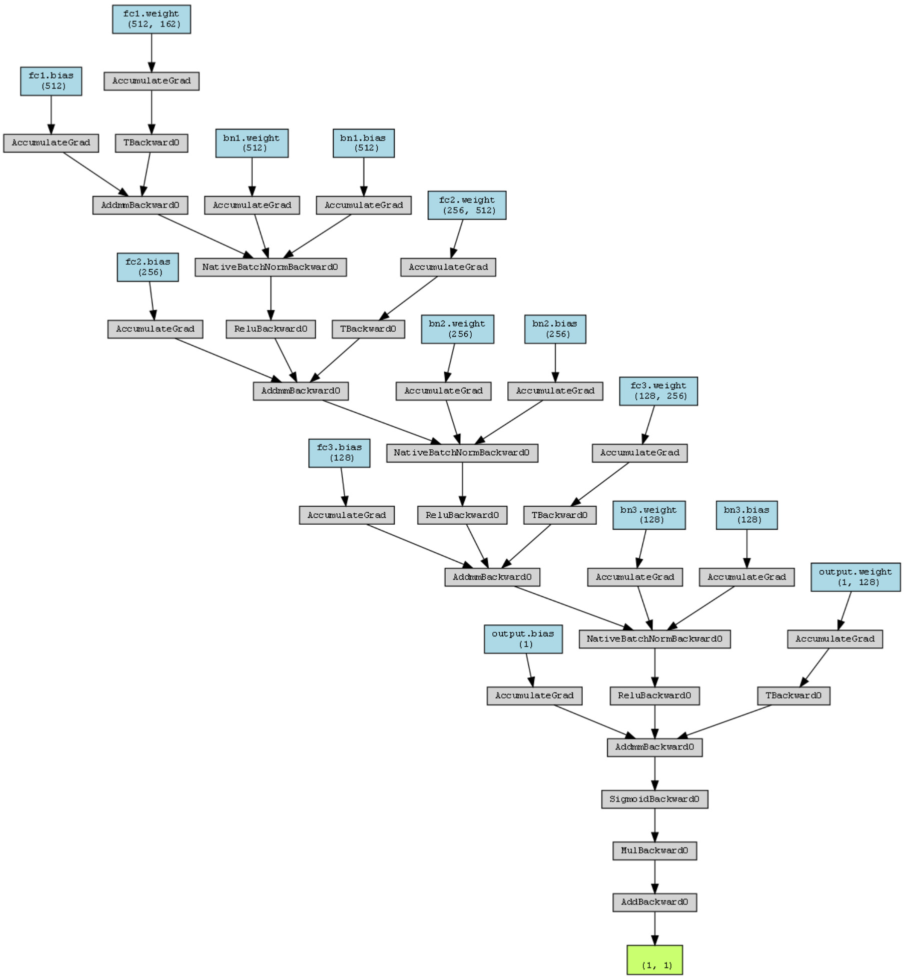
```python
#我们就计算前面500个用户的评分误差 (error)
filled_matrix, original_ratings = fill_zero_ratings(sim, test_matrix, top_n=5000)
mae = calculate_mae(original_ratings, filled_matrix)
print(f"Mean Absolute Error: {mae}")
```

```
[20]
...  Mean Absolute Error: 7.659398003428456
```

The high MAE is primarily caused by the dataset's sparsity, which limits the model's ability to learn user preferences and accurately predict missing ratings.

Deep Neural Networks

Architecture



We experimented with more complex neural network architectures in an attempt to improve the performance. However, the results did not show significant improvement. This may be attributed to the high sparsity of the dataset, which limits the model's ability to learn meaningful patterns and relationships effectively.

Data

Anime Features

The **anime features** come from the same list of features as the user's historical anime:
anime_features = ['score', 'ranked', 'popularity', 'members', 'favorites'] + genres

User Historical Rating Features

To represent user history:

- Each anime is represented by features such as score, ranked, popularity, members, favorites, and one-hot encoded genres.

If a user rates *One Piece* with a score of 9, and its feature vector is [0.2, 0.1, -0.1, ...], the weighted contribution of this anime to the user's history is calculated by multiplying the rating by the feature values:
Weighted Features = [9 * 0.2, 9 * 0.1, 9 * -0.1, ...] = [1.8, 0.9, -0.9, ...]
These **weighted features** are aggregated (e.g., summed or averaged) across all anime in the user's history to form the **User Historical Features**.

The final input is the concatenation of these two parts. For example:

Experiments

To evaluate the performance of our model, we constructed a dataset from the original user-anime rating data. For each user, one of their existing ratings was masked to serve as the prediction target during training. This ensures that the model learns from the available historical data while being tested on unseen inputs. A total of **15,000 samples** were generated for this purpose.

The dataset was then split into **training** and **validation** sets using an 80:20 ratio. The training set was used to optimize the model parameters, while the validation set was used to evaluate the model's generalization capability.

The final results were assessed using the **Mean Squared Error (MSE)** metric, which measures the average squared difference between the predicted ratings and the actual ratings. Lower MSE values indicate better predictive performance.

The final results on the validation set are as follows:

- **Test Loss (MSE):** 1.6835
- **Mean Absolute Error (MAE):** 0.9887

- **Mean Squared Error (MSE):** 1.6645

These results demonstrate the model's ability to predict user ratings with reasonable accuracy.

Conclusion

This project successfully developed a personalized anime recommendation system based on user ratings and anime features. By leveraging collaborative filtering techniques and deep neural networks, we explored multiple approaches to predict user preferences and generate recommendations.

The collaborative filtering model, utilizing Pearson correlation, demonstrated the ability to provide meaningful recommendations but faced challenges due to the dataset's inherent sparsity. Similarly, while the neural network model offered a more complex representation of user preferences, its performance was also limited by the sparsity of the data.

Despite these challenges, the final model achieved a Mean Squared Error (MSE) of 1.6835 and a Mean Absolute Error (MAE) of 0.9887 on the validation set, reflecting reasonable predictive accuracy. These results highlight the potential of leveraging user-anime interaction data for personalized recommendations while emphasizing the importance of addressing sparsity for further performance improvements.

Future work could explore techniques such as matrix factorization, hybrid recommendation systems, or additional data sources to enhance the model's ability to learn patterns and provide more accurate recommendations. Overall, this project contributes to the growing field of anime recommendation systems and demonstrates the power of machine learning in improving the user experience for content discovery.

Libraries we use

```
# python=3.10
nest_asyncio
aiohttp
beautifulsoup4
tqdm
numpy
pandas
selenium
webdriver-manager
celery
jupyter
torchsummary
scikit-learn
requests

# (PyTorch, torchvision, torchaudio for CUDA 12.1 support)
torch --index-url https://download.pytorch.org/whl/cu121
torchvision --index-url https://download.pytorch.org/whl/cu121
torchaudio --index-url https://download.pytorch.org/whl/cu121
```