Recommendation Systems for an Anime Dataset

Dingjia Chen*
University of Illinois
Urbana-Champaign
Urbana, Illinois, USA
dingjia2@illinois.edu

Fengqing Qiu*
University of Illinois
Urbana-Champaign
Urbana, Illinois, USA
fqiu3@illinois.edu

Yilun Fu*
University of Illinois
Urbana-Champaign
Urbana, Illinois, USA
yilunf2@illinois.edu

ABSTRACT

As a method of avoiding information overload and supplying users with relevant suggestions, recommendation systems have become a vital part of a vast array of web services, such as video-sharing platforms. The purpose of this project is to try different approaches and develop systems for recommending anime to users based on existing data. This project evaluates the performance of multiple approaches and finally determines the relatively optimal model.

CCS CONCEPTS

• **Information systems** → **Recommendation systems**.

KEYWORDS

data mining, self-supervised learning, machine learning

ACM Reference Format:
Dingjia Chen, Fengqing Qiu, and Yilun Fu. 2022. Recommendation Systems for an Anime Dataset. In *Proceedings of (Data Mining)*. ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The enormous range of Japanese cartoons available nowadays makes it challenging for the many individuals who enjoy watching them to decide which to watch. To assist viewers in selecting the ideal cartoons to watch, it would make sense to be able to build up a recommendation system.

This project implemented diverse recommendation systems for anime lovers. There are numerous ways to utilize the dataset, including content-based filtering(CBF), collaborative filtering (CF) and so on [1]. Systems that take a content-based recommendation approach can construct a model or profile of the user's interests based on the characteristics of the things they have previously evaluated [6]. Collaborative filtering is the process of filtering or evaluating items using the opinions of other people, and it brings together the opinions of large interconnected communities on the web, supporting filtering of substantial quantities of data [8]. We tried both of the methods to find the better one.

In the meantime, to avoid the time consumed by accurate annotations, which is frequently employed in supervised learning, self-supervised learning (SSL) is emerging as a new paradigm for extracting informative knowledge independent of manual labelling [12]. In [15], CF and SSL were merged into a framework named SelfCF. It turned out that the framework could achieve greater precision than a self-supervised framework BUIR. Therefore, after assessing the performance of CBF and CF, we also attempted to construct frames that apply SSL to determine which implementation is ideal.

The remaining structure of this paper is organized as follows. In Section 2, we briefly introduce the development of CBF, CF and SSL. In Section 3, we detailed describe the dataset we used and our approaches, and the corresponding results are displayed in Section 4. Finally, we summarize our contributions and find the possible future work in Section 5.

Data Mining, December 2022, Urbana, IL
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

2 RELATED WORKS

In this section, we first review the CB method and the CF technique, then summarize the self-supervised learning in recommendation systems.

2.1 Content-based Filtering

Content-based filtering is a method used in recommendation systems to make suggestions to users based on their preferences and past behavior [2]. It is a widely used and effective approach to recommendation systems, and has been applied to a variety of different domains, including online retail, music and video streaming, and social media. It works by analyzing the content of the items that a user has previously interacted with [5] (such as items they have purchased or rated highly) and using that information to identify similar items that the user might be interested in.

One of the key challenges in implementing content-based filtering is the need to accurately represent the content of the items being recommended [9]. This can be done using a variety of techniques, such as natural language processing to extract features from text descriptions of the items, or using machine learning algorithms to learn the underlying patterns in the data. The following stage, after the content of the objects has been represented, is to compare this representation to the user's previous interactions to identify related items. This can be done using a variety of similarity measures, such as cosine similarity or Euclidean distance.

Content-based filtering can make personalized recommendations to each user based on their individual preferences and history. This makes the recommendations more relevant and useful to the user. However, it can also suffer from the "cold start" problem, where there is insufficient data about a new user to make accurate recommendations.

2.2 Collaborative filtering

Collaborative filtering is another widely used method in recommendation systems. It works by creating a model of the user's interests based on their interactions with items, and then using that model to identify other users who have similar interests. The recommendations are then made based on the items that these similar users have interacted with.

There are two main approaches to collaborative filtering: user-based and item-based [3]. User-based collaborative filtering works by finding other users who have similar preferences to the user of interest, and then making recommendations based on what those similar users have interacted with. Item-based collaborative filtering, on the other hand, works by identifying items that are similar to ones that the user has interacted with in the past, and then making recommendations based on those similar items.

Collaborative filtering can make recommendations even for users who have a limited history of interactions with the system. This can help to overcome the "cold start" problem that can affect other methods, such as the content-based filtering mentioned above.

However, collaborative filtering can also suffer from some limitations [7]. For example, it can be difficult to accurately model the preferences of users, and the recommendations may not always be as relevant or useful as desired. Additionally, collaborative filtering can be sensitive to the presence of noisy or inaccurate data in the system.

2.3 Self-supervised Learning

The performance of neural architecture-based recommendation systems can be limited by data sparsity, since data acquisition in recommendation systems is costly, and most users only consume a tiny set of a great range of items. Self-supervised learning (SSL) alleviates this issue by automatically extracting informative and transferrable knowledge from unlabeled data. In other words, SSL reduces reliance on manual labels and enables training on massive unlabeled data.

The self-supervised recommendation systems are also called SSR, and SSR is multi-task learning which contains a self-supervised task (a.k.a. pretext task) and a recommendation task. Yu et al. have listed some essential features of SSR which include “SSR have a pretext task(s) to (pre-)train the recommendation model with the augmented data” and “recommendation task is the unique primary task, and the pretext task plays a secondary role to enhance recommendation” [14].

2.3.1 Different types of SSR.

There are four kinds of SSR based on their features of pretext tasks:

- Contrast-based method. It is the dominant branch in SSR. This kind of method is based on the concept of mutual information maximization, which predicts the agreement between two augmented instances in order to learn. It treats every instance (e.g. user/item/sequence) as a class, and then pulls views of the same instance closer in the embedding space and pushes views of different instances apart.
- Generation-based method. It is inspired by masked language models. It aims to reconstruct the input data and use the input data as supervision signals. In other words, it learns to predict a portion of the available data from the rest.
- Predictive method. In predictive methods, new samples or labels are derived from the original data to direct the pretext task.
- Hybrid method. It adopts multiple pretext tasks to better leverage the advantages of various types of supervision signals. More than one encoder and projection head are probably needed for a hybrid method.

2.3.2 Data Augmentation and Training Schemes.

In SSR, data augmentation is crucial for learning quality and generalizable representations. There are three commonly used data augmentation approaches for different data types:

- Sequence-based (sequence of users' historic behaviors) augmentation: item masking, item cropping, item substitution, item substitution.
- Graph-based (user-item graph with adjacency matrix) augmentation: edge/node dropout, graph diffusion, subgraph sampling, random walk.
- Feature-based (user/item features, operating in the attribute/ embedding space) augmentation: feature dropout, feature clustering, feature mixing.

2.3.3 Typical Training Schemes.

- Joint learning (JL) is mostly used in contrastive methods. The pretext task and the recommendation task are jointly optimized with a shared encoder.
- Pre-training and fine-tuning (PF) is used in most generative methods and a few contrastive methods. For a proper initialization of the encoder's parameters, the encoder is first pre-trained with pretext tasks using the augmented data. It is then fine-tuned on the original data for the recommendation.
- Integrated Learning (IL) is primarily used by predictive methods based on pseudo-labels as well as a few contrastive methods. In this setting, the pretext task and the recommendation task work well together and are combined into a single goal.

2.3.4 Commonly used datasets.

- Yelp2018
- Amazon-Book
- Douban-Book
- LastFM
- Alibaba-iFashion.

3 METHODOLOGY

The problem we wish to tackle is to parse and interpret all the data obtained from the dataset and utilize it to make it easier for consumers to locate the anime in which they are likely to be interested. We split our project into several parts.

3.1 Dataset

This Anime Recommendation dataset (Anime2020) [10] we selected contains information about 17562 anime and the preference of 325772 different users. This dataset in particular contains:

- the user's list of anime. Include the statuses dropped, finished, intend to watch, watching now, and on hold;

- ratings that people have assigned to the animes that they have finished watching;
- information about the anime like name, score, episodes, etc.
- a csv file about the anime's synopsis

The data was imported and preprocessed including data cleansing and analysis by the anime's synopsis and users' rating data.

3.2 Data import

The process of data importing includes loading the database and preprocessing. However, in order to prevent the memory limit from being exceeded due to the large amount of data, the data used for self-supervised learning is different from that for other methods.

For methods like content-based filtering and collaborative filtering, we decide to make full use of the whole dataset. The most important foundation of the recommendation system is the ratings of animes. So, the first step is to remove data without rating. Besides, we remove duplicates based on the animes' names. The subsequent steps are slightly varied due to the different requirements of different techniques. In each implementation subsection, more information will be provided.

The complete data volume reaches tens of millions, which is, for self-supervised learning, a big challenge both for running time and storage space. Hence, we selected the first 3.3% users of the dataset. To find the animes that the user is interested in more accurately, we remove the animes that the user's rating is lower than 7 from each user's watch list. In addition, dividing the training set and test set is an essential step for self-supervised learning. For each user, we divided the first 80% into the training set and the rest into the test set.

3.3 Content-based Filtering

Content-based Filtering utilizes the similarity between items to recommend similar item to the user's preferences. Content-based recommendation systems generate recommendations by relying on attributes of items. To find which feature benefits building a more accurate recommendation system, we tried two different features: 1) the anime's synopsis itself, and 2) a hybrid feature combined by the anime's genres ($weight = 3$) and the anime's synopsis ($weight = 1$).

The types of features are strings. To calculate the similarity between them, we convert strings to a matrix of TF-IDF features or a matrix of token counts. In this way, we can construct the similarity matrix using cosine. For each high-rated anime ($rating >= 7$) in the user's watch list, the system finds the top 10 animes with the highest similarity. Finally, it recommends the top 20 animes with the highest rating.

3.4 Collaborative filtering

Collaborative Filtering (CF) uses the similarity between users and items together to recommend items. There are two main approaches in the memory-based model, item-based and user-based. In practice, item-item outperforms user-user in many use cases. The reason is items are "simpler" than users, items belong to a small set of "genres", but users have varied tastes. The workflow is:

- Finding similarities of all the item pairs by calculating item similarity scores based on all the user ratings.

Algorithm 1 Content-based Recommendation System

```
1:  Feature construction: 1)single feature: synopsis; 2)hybrid feature: 3*genres+1*synopsis
2:  Convert feature to 1)a matrix of TF-IDF features; 2)a matrix of token counts
3:  Calculate cosine similarity matrix
4:  Get the user's watch list
5:  for anime in the watch list do
6:      if $anime.rating >= 7$ then
7:          list.append(anime)
8:      end if
9:      for anime in list do
10:         find top 10 recommendations with the highest similarity and $rating > 6$
11:     end for
12: end for
Output: Top 20 recommendations with the highest rating
```

- Identify the top n items that are most similar to the item of interest.
- Calculate the weighted average score for the most similar items by the user using the following formula:

$$r_{x_i} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{ij}}{\sum_{j \in N(i;x)} s_{ij}} \tag{1}$$

where s_{ij} is the similarity of item i and j, r_{ij} is the user x's rating on item j, and $N(i;x)$ set items rated by x similar to i.
- Rank items based on the score and pick the top n items to recommend.

There are different ways to measure similarities like Jaccard similarity, Pearson correlation, and cosine similarity. Since Jaccard similarity has the form in

$$J(A,B) = |A \cap B|/|A \cup B| \tag{2}$$

which will result in the issue of ignoring rating values, for instance, when A and B watch the same anime. Therefore, we did not take this measurement.

3.5 Self-supervised Learning

Self-supervised learning can lessen the reliance on manual labeling. This allows for training on vast amounts of unlabeled data, which recently attracted a lot of attention. The essential features of SSR are increasing the number of monitoring signals via semi-automation using the raw data itself for profit and having a pretext task to (pre-)train the recommendation model with the augmented data. Afterward, we need to make a recommendation task that relates to the pretext task. There are diverse models and in this project, SGL, XSimGCL, and SelfCF were applied. We utilized the 3rd party code [14] to accomplish this task.

3.5.1 *SGL: Self-supervised Graph Learning.*

Self-supervised learning on user-item graphs is able to improve the accuracy and robustness of GCNs for the recommendation. Specifically, we generate multiple views of a node, maximizing the agreement between different views of the same node compared to that of other nodes. We devise three operators to generate the views

— node dropout, edge dropout, and random walk — that change the graph structure in different manners [11].

The bipartite graph contains the collaborative filtering signal because it is constructed from user-item interactions that have been observed. Higher-order paths from a user to an item represent the user's possible interest in the object. Node dropout, edge dropout and random walk are used to mine the inherent patterns in the graph structure.

We treat the views of the same node as the positive pairs, and the views of any different nodes as the negative pairs. The contrastive loss, InfoNCE is used to maximize the agreement of positive pairs and minimize that of negative pairs:

$$\mathcal{L}_{ssl}^{user} = \sum_{u \in \mathcal{U}} -log \frac{exp(s(z'_u, z''_u/\tau)}{\sum_{v \in \mathcal{U}} exp(s(z'_u, z''_v/\tau)}$$

where $s(\cdot)$ measures the similarity between two vectors, which is set as a cosine similarity function; τ is the hyper-parameter, known as the temperature in softmax. Analogously, we obtain the contrastive loss of the item side $\mathcal{L}_{ssl}^{user}$. The objective function of a self-supervised task is $\mathcal{L}_{ssl} = \mathcal{L}_{ssl}^{user} + \mathcal{L}_{ssl}^{item}$. A multi-task training strategy is used to jointly optimize the classic recommendation task: $\mathcal{L} = \mathcal{L}_{main} + \lambda_1 \mathcal{L}_{ssl} + \lambda_2 \|\Theta\|_2^2$, where Θ is the set of model parameters in $\mathcal{L}_{main}$.

3.5.2 *XSimGCL: Extremely Simple Graph Contrastive Learning.*

XSimGCL not only inherits SimGCL's noise-based augmentation but also drastically reduces the computational complexity by streamlining the propagation process [13]. The difference is that SimGCL contrasts two final representations learned with different noises and relies on the ordinary representations for a recommendation, whereas XSimGCL uses the same perturbed representations for both tasks, and replaces the final-layer contrast in SimGCL with the cross-layer contrast. The architectures of these two different methods are displayed in 1.

3.5.3 *SelfCF: Self-supervised Collaborative Filtering.*

The main idea of SelfCF is to augment the output embeddings generated by backbone networks because it is infeasible to augment the raw input of user/item ids [16]. Three methods are used to introduce embedding perturbation in the framework. The historical embedding and embedding dropout are general techniques for output augmentation in the framework, while the edge pruning is specially designed for graph-based CF models (shown in Figure 2).

A symmetrized loss function as the negative cosine similarity between $(\dot{e_u}, \tilde{e_i})$ and $(\tilde{e_u}, \dot{e_i})$:

$$\mathcal{L} = \frac{1}{2} C(\dot{e_u}, \tilde{e_i}) + \frac{1}{2} C(\tilde{e_u}, \dot{e_i})$$

where $C(\cdot, \cdot)$ is defined as $C(e_u, e_i) = -\frac{(e_u)^T e_i}{\|e_u\|_2 \|e_i\|_2}$.

In SSL, we minimize the predicted loss between u and i for each positive interaction (u, i). We both predict the interaction probability of item i with u and the probability of user u with i. Given (e_u, e_i) being the output of the encoder f, the recommendation score is calculated as: $s(e_u, e_i) = h(e_u) \cdot (e_i)^T + e_u \cdot (e_i)^T$.

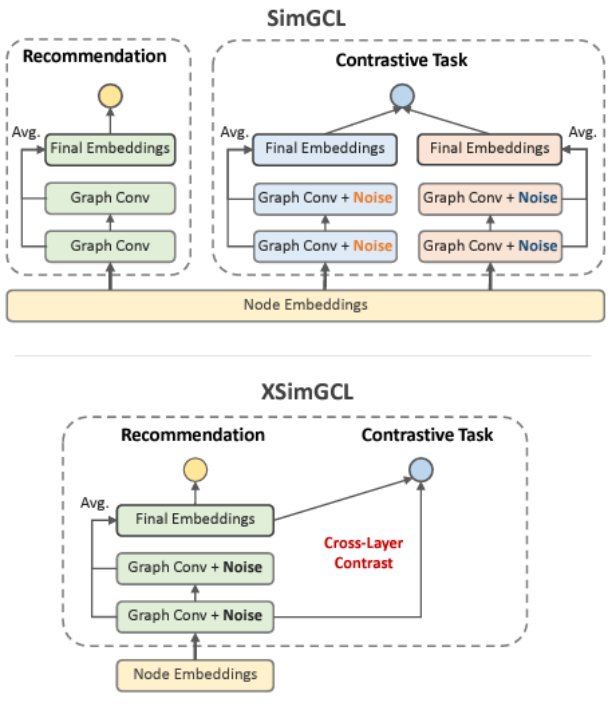


Figure 1: The architectures of SimGCL and XSimGCL

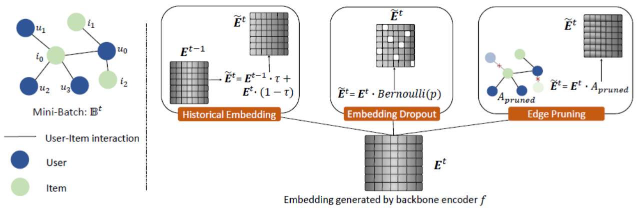

u_1
i_1
u_0
i_0
u_2
u_3
i_2
Mini-Batch: $\mathbb{B}^t$
User-Item interaction
User
Item
$\widetilde{E}^t$
E^{t-1}
$\widetilde{E}^t = E^{t-1} \cdot \tau + E^t \cdot (1-\tau)$
Historical Embedding
$\widetilde{E}^t$
$\widetilde{E}^t = E^t \cdot Bernoulli(p)$
Embedding Dropout
$\widetilde{E}^t$
A_{pruned}
$\widetilde{E}^t = E^t \cdot A_{pruned}$
Edge Pruning
E^t
Embedding generated by backbone encoder f

Figure 2: Illustration of output perturbation performed on a mini-batch

3.6 Evaluation

We compared the performance of content filtering and collaborative filtering on the anime rating dataset. We also experimented with self-supervised learning techniques.

There are many factors to be used for evaluation. In this project, the following factors were used:

- Hit rate: $hit\ rate = (hits\ in\ test)/(number\ of\ users)$. To calculate the hit rate, top N recommendations are generated for all of the users in the test data set. If the top N recommendations generated include items that users rated — 1 hit. The recommendation system is better with a higher hit rate.
- Precision: precision is the number of true positives divided by the number of total positive predictions. In other words, precision finds out what fraction of predicted positives is actually positive. The formula is: $Precision = \frac{TP}{TP+FP}$.

- Recall: similar to the precision, the recall is calculated by: $Recall = \frac{TP}{TP+FN}$. It measures the model's ability to predict the positives.
- Normalize Discounted Cumulative Gain (NDCG): it is a measure of ranking quality that is often used to measure the effectiveness of web search engine algorithms or related applications. It is computed by: $nDCG_p = \frac{DCG_p}{IDCG_p}$, where $DCG_p = \sum_{i=1}^{p} \frac{2^r el_i - 1}{log_2(i+1)}$ and $IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^r el_i - 1}{log_2(i+1)}$.

4 RESULT

4.1 Content-based Filtering

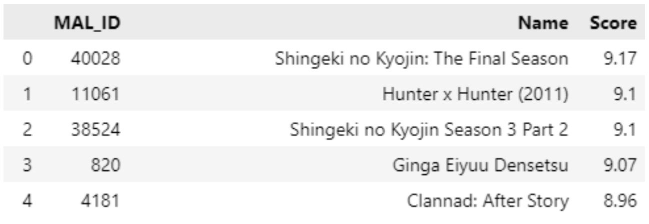
This method is based on the anime's synopsis, therefore, we removed data without a synopsis. The input detail is listed below:

Table 1: Input data for CBF

# of Users	# of Items	# of Interactions
309478	10817	56546235

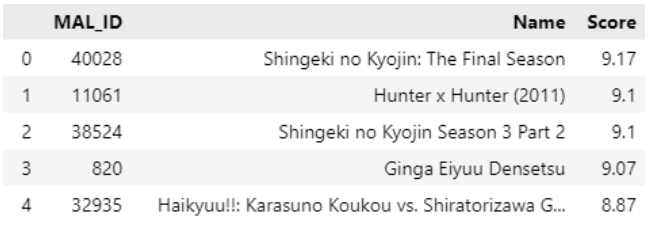
As mentioned in 3.3, we used two features to build the recommendation systems: 1) the anime's synopsis itself, and 2) a hybrid feature combined by the anime's genres (*weight* = 3) and the anime's synopsis (*weight* = 1).

Figure 3 and 4 display the Top 5 recommendations for the user whose user_id = 100.



	MAL_ID	Name	Score
0	40028	Shingeki no Kyojin: The Final Season	9.17
1	11061	Hunter x Hunter (2011)	9.1
2	38524	Shingeki no Kyojin Season 3 Part 2	9.1
3	820	Ginga Eiyuu Densetsu	9.07
4	4181	Clannad: After Story	8.96

Figure 3: Sample output using only synopsis



	MAL_ID	Name	Score
0	40028	Shingeki no Kyojin: The Final Season	9.17
1	11061	Hunter x Hunter (2011)	9.1
2	38524	Shingeki no Kyojin Season 3 Part 2	9.1
3	820	Ginga Eiyuu Densetsu	9.07
4	32935	Haikyuu!!: Karasuno Koukou vs. Shiratorizawa G...	8.87

Figure 4: Sample output using the hybrid feature

The performance of these two implementations is evaluated by hit rates. For individuals, remove one anime from the user's list, if the system can recommend the removed anime based on the other animes, then it's considered a "hit". Count the number of "hit" and compute their percentage. We randomly chose 32 users and compared the corresponding results. There were 25 results of the implementation only using synopsis larger than that using the hybrid feature. It turned out that the implementation only using synopsis performed better that using the hybrid feature.

Table 2: Part of hit rate results

user_id	only synopsis	hybrid feature
11433	0.2	0.1
56992	0.4	0.35
22880	0.3	0.4
45625	0.45	0.6
45625	0.45	0.6
114070	0.4	0.4
136858	0.75	0.3
148270	0.3	0.0
193768	0.6	0.5
216496	0.5	0.1

4.2 Collaborative filtering

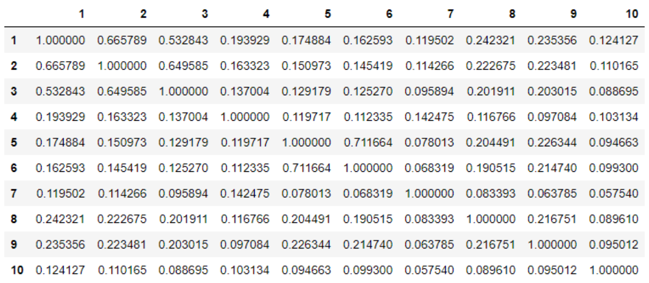
The amount of data exceeding 50 million requires huge storage space and time for CF. Consequently, we were only concerned about animes whose 'popularity' is in the top 10% because no one will recommend anime without good popularity.

Table 3: Input data for CF

# of Users	# of Items	# of Interactions
95077	1213	25273198

To accomplish better performance. we finally chose the Cosine Similarity to calculate the similarity between items. As for cosine similarity, we have to fill the missing values with 0. And this will cause a problem of treating missing ratings as negative. The way to handle this problem is to use a centered cosine to make 0 the average rating for all the anime a user watches. In this case, a positive rating means a user liked the anime more than average, while a negative rating indicates that he liked the anime less than average. The magnitude of the ratings also shows how much he liked/liked an anime. This also helps us to capture intuition better with handling "tough raters" and "easy raters".

As for Pearson correlation, it is the same as centered cosine similarity.

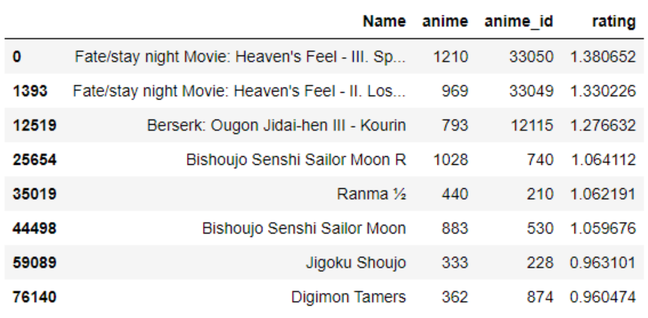


	1	2	3	4	5	6	7	8	9	10
1	1.000000	0.665789	0.532843	0.193929	0.174884	0.162593	0.119502	0.242321	0.235356	0.124127
2	0.665789	1.000000	0.649585	0.163323	0.150973	0.145419	0.114266	0.222675	0.223481	0.110165
3	0.532843	0.649585	1.000000	0.137004	0.129179	0.125270	0.095894	0.201911	0.203015	0.088695
4	0.193929	0.163323	0.137004	1.000000	0.119717	0.112335	0.142475	0.116766	0.097084	0.103134
5	0.174884	0.150973	0.129179	0.119717	1.000000	0.711664	0.078013	0.204491	0.226344	0.094663
6	0.162593	0.145419	0.125270	0.112335	0.711664	1.000000	0.068319	0.190515	0.214740	0.099300
7	0.119502	0.114266	0.095894	0.142475	0.078013	0.068319	1.000000	0.083393	0.063785	0.057540
8	0.242321	0.222675	0.201911	0.116766	0.204491	0.190515	0.083393	1.000000	0.216751	0.089610
9	0.235356	0.223481	0.203015	0.097084	0.226344	0.214740	0.063785	0.216751	1.000000	0.095012
10	0.124127	0.110165	0.088695	0.103134	0.094663	0.099300	0.057540	0.089610	0.095012	1.000000

Figure 5: Anime similarity(cosine)

For a specific user, we created a list of the anime that the user has watched and rated. Rank the similarities between an anime the user hasn't watched and other watched anime. We then predicted the rating of the unwatched anime by calculating the weighted average score of the top N of the most similar watched ones. Finally, rank items based on the score and pick the top n items to recommend.

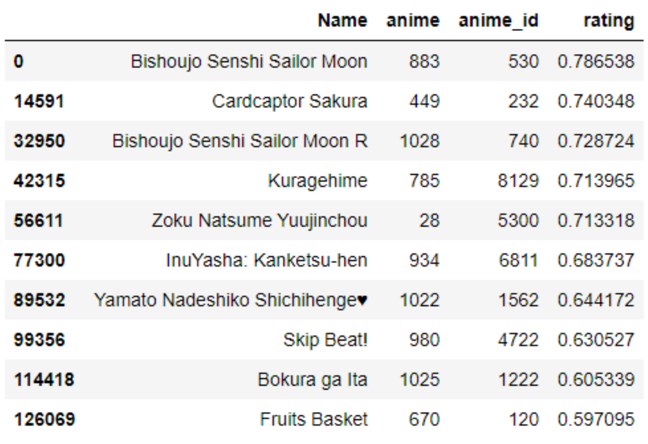
For this example, the top 8 recommendations for user 66 based on the most similar 4 anime are showing below:



	Name	anime	anime_id	rating
0	Fate/stay night Movie: Heaven's Feel - III. Sp...	1210	33050	1.380652
1393	Fate/stay night Movie: Heaven's Feel - II. Los...	969	33049	1.330226
12519	Berserk: Ougon Jidai-hen III - Kourin	793	12115	1.276632
25654	Bishoujo Senshi Sailor Moon R	1028	740	1.064112
35019	Ranma ½	440	210	1.062191
44498	Bishoujo Senshi Sailor Moon	883	530	1.059676
59089	Jigoku Shoujo	333	228	0.963101
76140	Digimon Tamers	362	874	0.960474

Figure 6: Sample output 1

We can change the base settings to get different recommendations. the top 10 recommendations for user 99 based on the most similar 10 anime are listed below:



	Name	anime	anime_id	rating
0	Bishoujo Senshi Sailor Moon	883	530	0.786538
14591	Cardcaptor Sakura	449	232	0.740348
32950	Bishoujo Senshi Sailor Moon R	1028	740	0.728724
42315	Kuragehime	785	8129	0.713965
56611	Zoku Natsume Yuujinchou	28	5300	0.713318
77300	InuYasha: Kanketsu-hen	934	6811	0.683737
89532	Yamato Nadeshiko Shichihenge♥	1022	1562	0.644172
99356	Skip Beat!	980	4722	0.630527
114418	Bokura ga Ita	1025	1222	0.605339
126069	Fruits Basket	670	120	0.597095

Figure 7: Sample output 2

4.3 Self-supervised Learning

Table 4 summarizes the statistics of the datasets. In the table, data density is measured as the number of interactions divided by the product of the number of users and the number of items.

- Yelp2018: the dataset provided in this page [4]. It contains a subset of yelp's user data. It is set as the benchmark.
- Anime2020(A): our original dataset [10], which indicates that it is not pre-processed for the use of self-supervised learning models.
- Anime2020(B): the dataset pre-processed, only including data whose rating is greater than 7 of the first 3.3% users.

Table 4: Statistics of the experimented data

Dataset	# of Users	# of Items	# of Interactions	Density
Yelp2018	19539	21266	450884	0.11%
Anime2020(A)	310059	17562	57633278	1.06%
Anime2020(B)	10371	11503	1480667	1.24%

The performance of selected self-supervised learning models is displayed in Table 5. By comparing the results, the following conclusions can be made:

- The results of Yelp2018 show that all factors are relatively low, which means the performance of these SSR models remains to be improved.
- In the case of Yelp2018, the performance ranking from the best to the worst is XSimGCL, SGL, and SelfCF.
- In the case of Anime2020(B), the performance ranking from the best to the worst is SelfCF, SGL, and XSimGCL.
- Comparing the results of two datasets, the performance in Anime2020(B) is worse than that of Yelp2018. It may be because most data is abandoned during the pre-processing. The remaining data cannot reflect the truth.

Table 5: Performance comparison of different SSR models

Dataset	Factors		SGL	SelfCF	XSimGCL
Yelp2018	Top 10	Hit rate	0.0337	0.0235	0.0371
		Precision	0.0344	0.0241	0.0379
		Recall	0.0387	0.0256	0.0421
		NDCG	0.0450	0.0303	0.0493
	Top 20	Hit rate	0.0576	0.0412	0.0638
		Precision	0.0295	0.0211	0.0327
		Recall	0.0657	0.0449	0.0722
		NDCG	0.0549	0.0376	0.0604
	Runtime		1209s	3545s	686.9s
Anime2020(B)	Top 10	Hit rate	0.0138	0.0168	0.0126
		Precision	0.0407	0.0495	0.0372
		Recall	0.0327	0.0364	0.0359
		NDCG	0.0461	0.0560	0.0457
	Top 20	Hit rate	0.0285	0.0344	0.0245
		Precision	0.0420	0.0508	0.0361
		Recall	0.0595	0.0669	0.0630
		NDCG	0.0557	0.0663	0.0536
	Runtime		2314s	2327s	458.3s

5 CONCLUSION

In this project, we built different recommendation systems for an Anime dataset. Specifically, we finished:

- implementing two content-based filtering methods: one only uses animes' synopsis and the other combines animes' synopsis and genres by weight. Usually, the former can get a higher hit rate for individuals.
- implementing one collaborative filtering system: it uses the item-based model and recommends the most similar animes with the user watched.

- testing three self-supervised learning models: SGL, SelfCF and XSimGCL. The Yelp2018 and the anime dataset were fed into the models. Usually, these models got better results in Yelp2018. And SelfCF performed best in the anime dataset, while XSimGCL did best in Yelp2018.

All the recommendation systems can get top N recommendations for users. However, limited by the characteristics of different methods, small storage space and short running time, we applied different pre-processing approaches for three parts of the project. Hence, a side-by-side comparison of the three methods is meaningless.

CBF and CF are relatively mature methods that have slowed development in recent years. According to our experiments, they can get relatively higher hit rates but are time-consuming. There's little space for improvement in these two methods. As for future work, we should pay more attention to self-supervised learning.

Self-supervised learning is a powerful method and is likely to continue to play an important part in the future development of recommendation systems since it can learn from huge volumes of user data without the need for costly and time-consuming manual labeling. As the amount of available user data continues to increase, self-supervised learning will enable recommendation systems to utilize this data to produce more precise and personalized recommendations. In addition, as self-supervised learning algorithms continue to advance, it is probable that they will become more effective at capturing the intricate interactions between users, items, and ratings, enabling recommendation systems to make more accurate predictions and deliver better recommendations.

In order to improve the performance of recommendation systems, researchers may explore new methods of incorporating additional types of data, such as users' behavior and their social media data. In addition, researchers may investigate methods for enhancing the scalability of recommendation systems, and methods for mitigating the possible biases in the collected dataset.

REFERENCES

[1] Xavier Amatriain, Nuria Oliver, Josep M Pujol, et al. 2011. Data mining methods for recommender systems. In *Recommender systems handbook*. Springer, 39–71.

[2] S Wesley Changchien, Chin-Feng Lee, and Yu-Jung Hsu. 2004. On-line personalized sales promotion in electronic commerce. *Expert Systems with Applications* 27, 1 (2004), 35–52.

[3] Chein-Shung Hwang and Pei-Jung Tsai. 2005. A collaborative recommender system based on user association clusters. In *International Conference on Web Information Systems Engineering*. Springer, 463–469.

[4] Xin Xia Tong Chen Jundong Li Zi Huang Junliang Yu, Hongzhi Yin. [n. d.]. SELFRec. https://github.com/Coder-Yu/SELFRec.

[5] Richard D Lawrence, George S Almasi, Vladimir Kotlyar, Marisa Viveros, and Sastry S Duri. 2001. Personalization of supermarket product recommendations. In *Applications of data mining to electronic commerce*. Springer, 11–32.

[6] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. 2011. Content-based recommender systems: State of the art and trends. *Recommender systems handbook* (2011), 73–105.

[7] Miquel Montaner, Beatriz López, and Josep Lluís De La Rosa. 2003. A taxonomy of recommender agents on the internet. *Artificial intelligence review* 19, 4 (2003), 285–330.

[8] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 291–324.

[9] Lalita Sharma and Anju Gera. 2013. A survey of recommendation system: Research challenges. *International Journal of Engineering Trends and Technology (IJETT)* 4, 5 (2013), 1989–1992.

[10] Hernan Valdivieso. [n. d.]. Anime Recommendation Database 2020. https://www.kaggle.com/datasets/hernan4444/anime-recommendation-database-2020?select=watching_status.csv.

[11] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 726–735.

[12] Lirong Wu, Haitao Lin, Cheng Tan, Zhangyang Gao, and Stan Z. Li. 2021. Self-supervised Learning on Graphs: Contrastive, Generative,or Predictive. *IEEE Transactions on Knowledge and Data Engineering* (2021), 1–1. https://doi.org/10.1109/TKDE.2021.3131584

[13] Junliang Yu, Xin Xia, Tong Chen, Lizhen Cui, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2022. XSimGCL: Towards Extremely Simple Graph Contrastive Learning for Recommendation. *arXiv preprint arXiv:2209.02544* (2022).

[14] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Jundong Li, and Zi Huang. 2022. Self-Supervised Learning for Recommender Systems: A Survey. *arXiv preprint arXiv:2203.15876* (2022).

[15] Xin Zhou, Aixin Sun, Yong Liu, Jie Zhang, and Chunyan Miao. 2021. SelfCF: A Simple Framework for Self-supervised Collaborative Filtering. https://doi.org/10.48550/ARXIV.2107.03019

[16] Xin Zhou, Aixin Sun, Yong Liu, Jie Zhang, and Chunyan Miao. 2021. SelfCF: A Simple Framework for Self-supervised Collaborative Filtering. *arXiv preprint arXiv:2107.03019* (2021).