User-based Ensemble Steam Game retrieval system

Dongjian Chen
MSI student, School of Information
University of Michigan, Ann Arbor
djichen@umich.edu

Shukai Fan
MSI student, School of Information
University of Michigan, Ann Arbor
kennyfan@umich.edu

Tianyi Ge
MSI student, School of Information
University of Michigan, Ann Arbor
gty@umich.edu

Abstract

In this project, we use data mining and machine learning skills to build a ensemble user-based steam game retrieval system. We build our retrieval system based on the traditional game search engine based on query-document matching[10] and add a ensemble recommendation model which use additional user features and item features to predict users' preferences for items. The combination of search model and recommendation can provide results not only based on searching query but also based on user preference. After the design and implementation of the models, we also design a specific evaluation process.

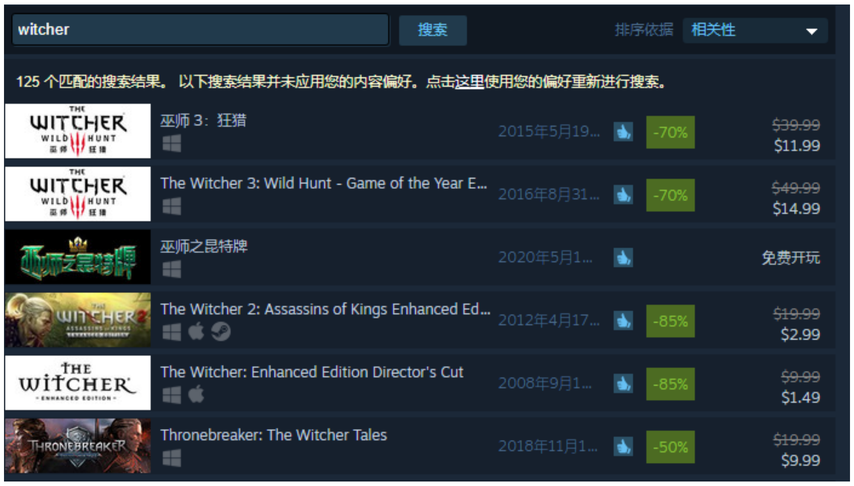
1 Introduction

1.1 Motivation

Steam platform is one of the worldwide largest game platform. As many other large global platform, Steam provides a searching engine that can help users retrieve the game they want by entering specific queries as shown in Fig 1. However, under some conditions, users like me may not only want results from retrieval engine to be as relevant to the query as possible but also want the results meet our preferences. The preference of a steam user may be judged by some specific features such as the features of games he already owns or his past behaviors on the platform. If the preference of a user can be considered in a game retrieval process, we believe the user can get a better, personalized results. For example, when a user who hates action games searches a role-playing game with a vague query which returns a lot of action games as searching results, he may feel unsatisfied with the retrieval engine. So we decide to build a ensemble user-preference based game retrieval engine.

1.2 Problem Statement

The problem in our project is to combine a classical searching model with a user-preference based recommendation model and raise a recommendation method that can take advantage of rich user and item data based on data mining methods. For the recommendation model part, a traditional method is collaborating filter which use the memory of some simple user behaviors and item relationships to predict users preference features[1]. However, some additional features will be neglected in this traditional method so we decide to apply another machine learning model to take advantage of some additional features of user and items we find. Finally, we combine two recommendation models and apply it on the results of the searching model and get the final personalized results. We also design a evaluation process to do experiments on our ensemble retrieval model to prove its effectiveness in real scenarios.



witcher 搜索 排序依据 相关性

125 个匹配的搜索结果。 以下搜索结果并未应用您的内容偏好。点击这里使用您的偏好重新进行搜索。

巫师 3：狂猎 2015年5月19... -70% $39.99 $11.99

The Witcher 3: Wild Hunt - Game of the Year E... 2016年8月31... -70% $49.99 $14.99

巫师之昆特牌 2020年5月1... 免费开玩

The Witcher 2: Assassins of Kings Enhanced Ed... 2012年4月17... -85% $19.99 $2.99

The Witcher: Enhanced Edition Director's Cut 2008年9月1... -85% $9.99 $1.49

Thronebreaker: The Witcher Tales 2018年11月1... -50% $19.99 $9.99

Figure 1: **Steam search engine**

2 Data processing and analysis

2.1 Description

2.1.1 Steam user data

We discover a large steam data set on the website provided by Julian McAuley, UCSD[2]. This data set provides rich steam data including user review data, user repository data, bundle data, etc. Among all data sets provided we choose two user data set, user review data set and user play behavior data set, construct our user features. All data sets are in JSON format and a sample is shown in Fig 2. The user review data set contains reviews data of each users including if a user recommend a game and if the review from the user is useful. The user play behavior data set contains the game each user owns and how many time users spend on each game.

```
{'user_id': 'LydiaMorley',
 'user_url': 'http://steamcommunity.com/id/LydiaMorley',
 'reviews': [{'funny': '1 person found this review funny',
   'posted': 'Posted July 3.',
   'last_edited': '',
   'item_id': '273110',
   'helpful': '1 of 2 people (50%) found this review helpful',
   'recommend': True,
   'review': 'had so much fun plaing this and collecting resou
rces xD we won on my first try and killed final boss!'},
  {'funny': '',
   'posted': 'Posted July 20.',
   'last_edited': '',
   'item_id': '730',
   'helpful': 'No ratings yet',
   'recommend': True,
   'review': ':D'},
  {'funny': '',
   'posted': 'Posted July 2.',
   'last_edited': '',
   'item_id': '440',
   'helpful': 'No ratings yet',
   'recommend': True,
   'review': 'so much fun :D'}]}
```

Figure 2: **Sample metadata**

2.1.2 Steam game data

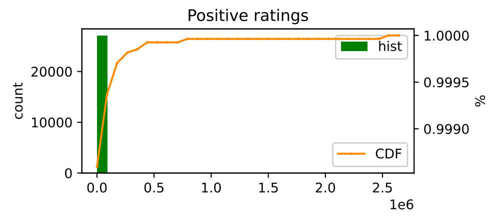
For the features of games, we find a steam game dataset on Kaggle website[3]. The Kaggle dataset contains the information of games on steam platform including the game's genre, required age, language, etc. The dataset is in CSV form thus it can be directly loaded into dataframe for further process and analysis.

The cleaned dataset contains 13 columns in total. including *appid, name, release date, price, genres, steam tags, positive ratings, negative ratings, owners, median playtime, detailed description, about the game*, and *short description*.

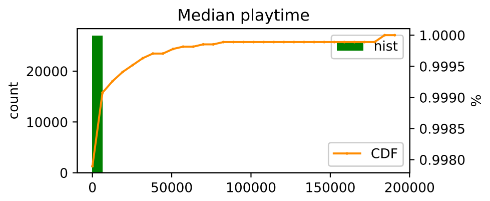
2.2 Exploratory data analysis

2.2.1 Steam games overview

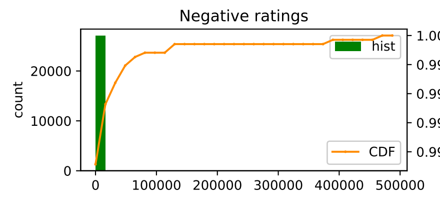
We checked three properties from game dataset, and found clear skewed distributions. This observation is expected because in real world most of the games are receiving relatively low attention, with only a few exceptions (such as Counter-Strike and Dota 2). The 99 percentile of positive ratings (Fig.3a), median playtime (Fig.3b), and negative ratings (Fig.3c) are respectively 15116, 1810, and 2799. To mitigate the difference between popular games and mediocre ones, we will perform log-scale transformation so that mediocre games can be further differentiated regardless of those exceptional games (see Fig.3d). To avoid numerical error by taking logarithm of 0 we implement each value by 0.1.



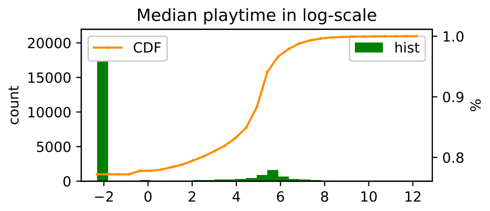
(a) Positive ratings of games dist.



(b) Median playtime dist.



(c) Negative ratings of games dist.



(d) Median playtime by log-scale

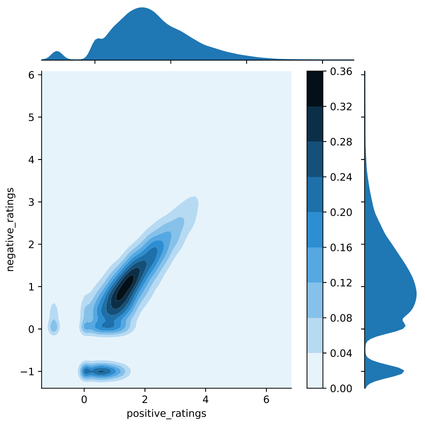
Figure 3: **Distribution of game properties: proitive ratings, median play time, negative ratings, and log-scaled median playtime**

We also found that negative ratings are not convincing enough to directly tell the poorness of a game. It can be demonstrated by the high correlation between positive/negative ratings after log-scale (see Figure 4a). Although the 2-dimensional histogram demonstrates that some games receiving high positive ratings have few negative ratings (the lower cluster), more popular games also tend to received more criticisms (the long cluster on diagonal), so it's unfair to directly use such rating for game comparisons. To overcome this, we applied collaborating filtering, which will be further discussed in the memory-based recommendation section.

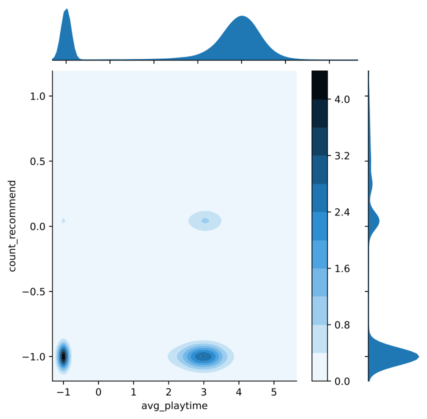
2.2.2 Users dataset overview

The user dataset, as mentioned before, includes multiple natural language information (e.g. review) and link contents. To have a general sense of users we observe the cluster result by K-means evaluated with silhouette score.

The silhouette score shows that the users are approximately 3 clusters (see Figure. 5). We observe that the median of average review length per cluster are 23.98, 194.75, 1431 respectively, which cate-



(a) Correlation between positive and negative ratings given by users in log-scale



(b) Correlation between user playtime and #recommended games in log-scale

Figure 4: **Correlation heatmap in log-scale for the properties of games (a) and users (b)**

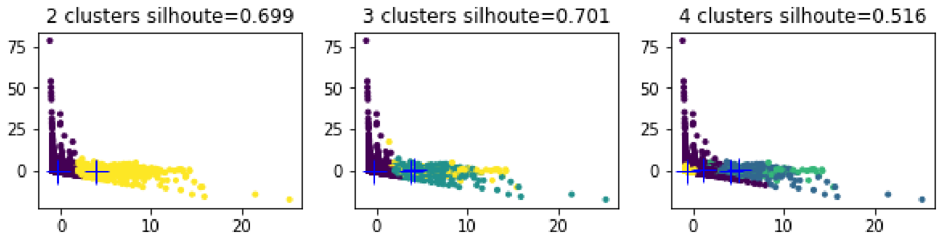


Figure 5: **User profiling by cluster**

gorizes users by their level of activity in gaming ecosystems. We observed a Spearman's correlation coefficient of **0.93778**. Although the high correlation might be resulted from the low participation rate in review and recommendation, we think gamers giving longer reviews have higher chance to recommend games.

Another important user property is shown in Figure. 4b. Ignoring the left bottom cluster, we found that a gamer's average playtime actually has weak relation with the number of games they recommend. It's also validated by Spearman's coefficient (0.157). Therefore, we might need to consider both in our user-oriented algorithm since they represent two distinct facets of a user.

2.3 Pre-processing and cleaning

The original data from the website[1] is in JSON format so we first process the data to extract features we need and save all the features into a dataframe format. In this part we will generate three dataframes. First is the dataframe for the memory-based recommendation model and also the train label for the multi-dimention recommendation model which is extracted from user play behavior dataframe and contains each user's playtime on each game. This database have about 5000000 rows and we sample it in our experiment and model training part. Second is the user-feature dataframe which contains the multi-dimensional features we extracted for each user which contains about 90000 rows of user data. The third dataframe is the item-feature dataframe containing the multi-dimensional features we extracted for each item. We have about 30000 rows each for one item extracted from original data sets.

After extracting three datasets, we simply process our data to prepare for model building part. Since we decide to use users playtime on games as label, we need to process the playtime data to make it more reasonable for training. The original playtime data has a really fluctuating distribution so we decide to normalize the data scale of playtime data. We use log transform and max-min scale method to re-scale playtime as the label in later model training and experiments. Since in the first part of pre-process, extracting data from original dataset to dataframe, we already filter some abnormal data such as null values. So we don't apply further cleaning methods on our data.

3 Design and Implementation

3.1 System Overview

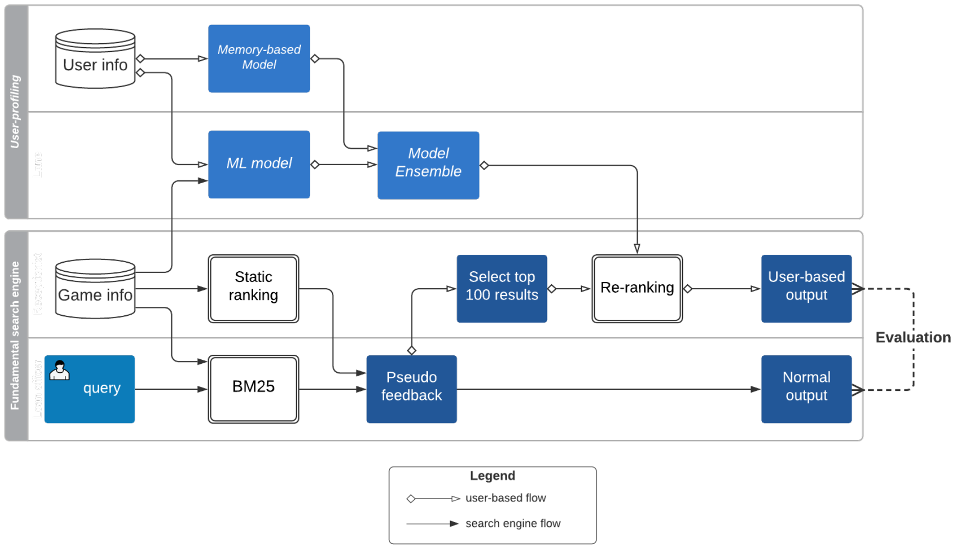


Figure 6: **Overview of the system workflow**

Our system is composed of two parts: fundamental search engine and user profiling improvements. The basic search engine is following a traditional methodology to retrieve best matches with NLP approaches e.g. TF-IDF and BM25. In addition to this, our user-based models include a model ensemble of memory-based model and Machine Learning model. We perform the rank optimization and evaluate our achievements by comparison. The detailed design and reasons are introduced in the following sections.

3.2 Search[10]

It's sometime annoying when game players exploring new games on game stores like Steam, because the search engine only allows searching on names and tags. If the user only have a vague idea of which kinds of games he want to explore, the existing search engine may cause inconvenience.

Therefore, instead of using the original search engine, we utilized a game specific search engine allowing vague search on description as the backbone. The search engine is equipped with other features like popular search engines to achieve higher accuracy and easier use. This search engine primarily take the advantage of steam game dataset [3].

3.2.1 BM25 based Ranking

The basic searching function is based on the BM25. BM25 is a kind of tf-idf-like retrieval functions with probabilistic retrieval framework [11].

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \frac{f(q_i,D) * (k_1 + 1)}{f(q_i,D) + k_1 * (1 - b + b * \frac{|D|}{\text{avgdl}})} \tag{1}$$

where $f(q_i, D)$ is the term frequency, and $\text{IDF}(q_i) = \log \frac{N-n(q_i)+0.5}{n(q_i)+0.5}$ is the inverse document frequency. b and k_1 are constants. In our project, we obtain scores from various documents, such as description, name and tag. The scores are added to form a score resemble with different weights.

3.2.2 Static Ranking

In order to take the quality of games into consideration and press penalty on low quality games, we introduce static ranking method in our platform. Static rank represents the value of each item and independent from the input query.

We calculate the static rank from the size, the popularity, the positive review rate, the total review number, the freshness and the median playtime, along with the grading scores from IGN websites. Static rank is combined with the BM25 score to decide the final rank.

3.2.3 Pseudo Relevance Feedback

Before show the results feed to the user-based re-ranking, we will re-run the progress for two extra feedback rounds. In this period, the top 10 results are considered as relevant while others will be considered as irrelevant. The top 10 results serve as the pseudo feedback of "like", so we will increase the rank of games similar to the top 10. When offering feedback for each of them, the query representation will be modified in the direction of high-ranked documents' representation (also in the reverse direction for irrelevant documents' representation). Moreover, normalization is provided with respect to the document number. The results after 2 extra rounds will be shown to the user as final result.

The idea is from *Rocchio algorithm* [12], i.e. most of documents within a class is near to a central point. By two rounds of pseudo relevance feedback, we can raise the rank of documents similar to the high-relevant results. As a result, modified query representation will be pull towards the center of the target game cluster.

3.3 Recommendation

3.3.1 Memory-based Model

Our first recommendation model applies memory-based method, also known as neighbor-based method. We treat the two properties **playtime** and **recommendation indicator** after user-item aggregation. As mentioned in section 2.2, we applied log-scale to the skewed data to capture minor differences among game playtime.

Memory-based and model-based methods are two main categories of Collaborative Filtering techniques, which aim at harnessing quality judgement from similar users for evaluations on new items for a user. Memory-based method is a widely used strategy in recommendation system as it predicts rating for a user-item pair based on the user history of ratings on different items. Therefore, the key objective of this approach is to design appropriate similarity metrics to user-item pairs.

One of the greatest advantages of Collaborating Filtering is that it does not require feature selection at all (we focus on the feature selection methods in the next section). Most today's Collaborating Filtering algorithms are based on user-item rating matrix with entries as ratings given by users to items.

Single Value Decomposition (SVD) is a famous matrix factorization-based prediction method popularized by Funk et al. during the Netflix Prize[7]. Its core strategy is to obtain a decomposition of matrices with latent factors connected to users and items[8]. The rating estimation is calculated by

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u \tag{2}$$

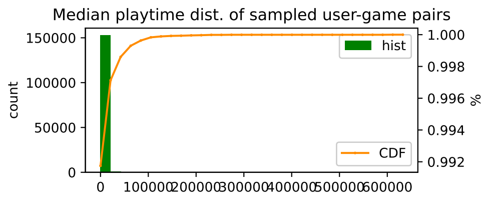
where μ represents the average rating of all items; b_u represents the average rating given by user u minus μ; b_i represents the average rating of item i minus μ; q_i is the item row of item-factor matrix, and p_u is the user row of user-factor matrix.

SVD++, as an extension to SVD, combines the consideration of "implicit ratings" into the rating. A new term $\left(|I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j\right)$ is added to p_u thus capturing the latent value of "rating action" itself.

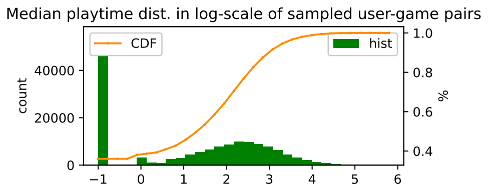
Due to the large size of our sampled data, we adopted SVD as the preferred algorithm instead of SVD++ as it obtains similar performance but drastically save algorithm runtime.

Apart from SVD, we also select KNN with means and Co-clustering algorithms to serve as baselines. Co-clustering is a straightforward collaborative filtering strategy as it assigns users and items to group and calculate the rating based on group averages. We select it as one of the baselines because we would like to experiment if our user clustering makes sense in rating system.

From the sampled dataset we could observe a similar distribution of playtime as the entire game population (see Figure. 7b).



(a) Median playtime dist. of sampled user-game pairs



(b) Median playtime dist. in log-scale of sampled user-game pairs

Figure 7: **Distribution of playtime sampled user-game pairs**

We adopted Mean Absolute Error (MAE) to evaluate the performance of each model, since all the ratings are log-scaled numerical data. We performed 5-fold grid cross-validation[9] on each of the three models and report the MAE scores and best parameters. The benchmark results of playtime and recommendation indicator are listed in Tab. 1 and 2.

Rank	Algorithm	MAE
1	SVD	1.639161
2	KNNWithMeans	1.726102
3	Co-clustering	1.841283

Table 1: Prediction of log-scaled game playtime

Rank	Algorithm	MAE
1	SVD	1.716649
2	KNNWithMeans	1.808506
3	Co-clustering	1.862197

Table 2: Prediction of log-scaled game recommendation

The parameter combination giving the best MAE score are shown below.

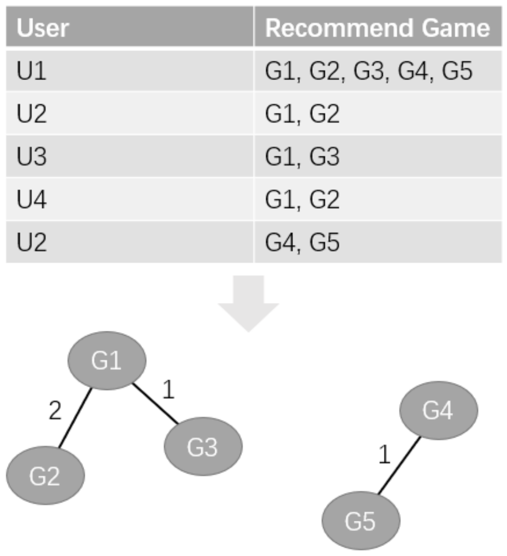
1. SVD: random_state: 671, n_epochs: 50, lr_all: 0.01, reg_all: 0.01
2. KNNWithMeans: random_state: 671, min_k: 3
3. CoClustering: random_state: 671, n_cltr_u: 3, n_epochs: 40

From the prediction results, we observe that SVD has the lowest MAE score and thus we select it as out final memory-based algorithm. Notice that we are not aiming at approaching the MAE lower bound of this rating system. Instead, we are utilizing this metric to help indicate two dimensions of user preferences in selecting games. With this tuned SVD model we are able to add user-oriented customization and proceed to build the model ensemble in order to improve the probability for a user to successfully find a preferred game from the query search results.

3.3.2 Machine learning Model

Besides the memory-based model, we want to build another model based on machine learning skills that can be trained to understand multi-dimensional features for users and games and predict each user's preference for each game. Different from building a complex sequence model to learn user-based behavior data like raise in the paper "Self-Attentive Sequential Recommendation" [4], we already have a memory-based model to deal with some user behavior data so we decide to build a model with basic machine learning methods to learn mixed multi-dimension user and item features in this part. We first decide to build feature set for users and items based on original data and some data mining skills and then apply machine learning model on those features to do model selection.

- **Feature building**: We first build user and item feature sets based on our original data sets. For users, since we already have a memory-based model to understand user behaviors, we only need to use some additional features from users in this part including the users' average playtime, number of games, number of recommendations, etc. And for the game features we get feature though two ways. The first part of item features come from the statistic features from the game dataframe we built in the pre-process section including the achievements number, positive rating number, negative rating number, etc. Besides those statistics features, we also build co-recommend item network for item from the user recommendation data we have. We build a weighted indirect network in which each node is a steam game and there exists an edge between every two games that recommended by the same users. The weights of the edges are counts of times of co-recommendation relationship. A sample building process of the co-recommend netowrk is show in Fig 8.



User	Recommend Game
U1	G1, G2, G3, G4, G5
U2	G1, G2
U3	G1, G3
U4	G1, G2
U2	G4, G5

Figure 8: **Example of building of Item co-recommend network.**

We roughly analyze the network properties and find some existed characteristics for game relationships on steam. First is that the distribution of degrees as shown in Fig 9 in the network shows most games has small degree numbers and few games has really large degree numbers which represent that several popular games are recommended by lots of users while lots of games may only known or recommended by few players. Another phenomenon we find is that the average weight in the graph is around 1.7 and the max weight in the graph is 468 which means most games in the network are co-recommended by small number of players but several popular games can be co-recommended by hundreds of players which is agree with the characteristics we notices in the degree distributions.

- **Model selection**: After applying some simple min-max Scala methods on our prepared features, we can start model implementation and experiment part. First, we need to find proper metrics for the model evaluation and selection. The first metric we decide to use is mean absolute error(MAE) which is a popular metric for regression tasks. The second metric

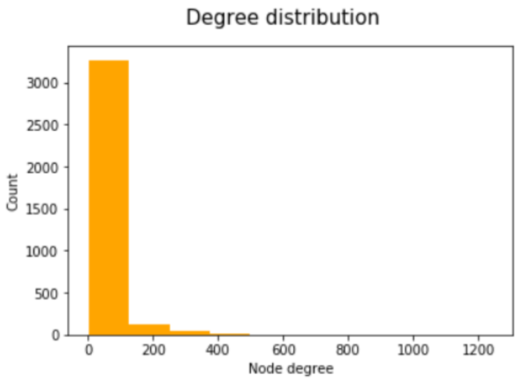


Figure 9: **Degree distribution of the co-recommend network.**

we want to use to evaluate our model is Normalized Discounted Cumulative Gain(NDCG) which is a popular metric for ranking problem since we care about not only the precision of regression scores but also the rank of scores because we will recommend games based on the rank. The NDCG score[5] is a rank model evaluation metric which is popular in measuring the ranking quality of information retrieval tasks. Lots of researches on recommendation models use AUC and NDCG as evaluation metrics these years[6]. The NDCG score is normalized discounted cumulative gain which is a score in range of 0.0 to 1.0 [5],

$$NDCG = \frac{DCG}{IDCG} \tag{3}$$

$$DCG = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{4}$$

$$IDCG = \sum_{i=1}^{|rel_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{5}$$

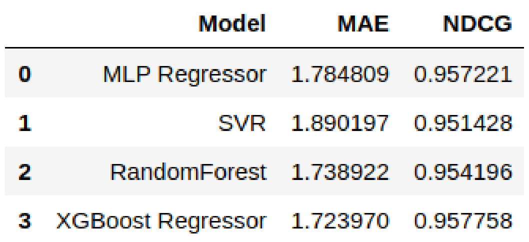
where rel_i is the graded relevance of the result at position i and rel_p represents the list of games in the set up to position p. We use MAE score to evaluate the quality of regression and use NDCG score to evaluate the quality of ranking.

After we decide the metrics for model evaluation, we start model implementation and selection task. We decide to implement four models including MLP Regressor, SVR, RandomForest Regressor, XGboost Regressor. We want to ensure the comprehensiveness of our model set so we choose linear models including MLP Regressor and SVR and we also choose two ensemble models including XGboost and RandomForest. When selecting models, we get five MAE scores for each model with 5-fold cross validation and use the mean of five score to evaluate each model's performance. Besides the MAE score, we also use NDCG score to evaluate models. We split train and test data and use prediction result and test result to calculate NDCG scores. Then we compare the MAE and NDCG scores of four models and choose the model with smallest MAE score and largest NDCG score. The result is shown in Fig 10. We can see from the table that the XGboost regressor has the highest NDCG score and the smallest MAE. So we decide the choose XGboost regressor as our final model.

From the results in Fig 10, we can see that the XGboost regressor has the highest NDCG score and the smallest MAE. So we decide the choose XGboost regressor as our final model. Then we start applying further parameter tuning on XGboost regressor based on grid search and cross validation and find the following parameter set as shown in Fig 11.

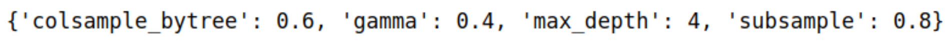
3.3.3 Model ensemble

After we complete the memory-based model and machine learning model, we start the combination of two models. The memory-based model will provide a score predicted by user's past preference



	Model	MAE	NDCG
0	MLP Regressor	1.784809	0.957221
1	SVR	1.890197	0.951428
2	RandomForest	1.738922	0.954196
3	XGBoost Regressor	1.723970	0.957758

Figure 10: **Result of model evaluation.**



```
{'colsample_bytree': 0.6, 'gamma': 0.4, 'max_depth': 4, 'subsample': 0.8}
```

Figure 11: **Best parameter set after grid search.**

on games and the machine learning model will provide a score predicted by additional user and item features. We just use simple weighted sum method to combine the score from two models since its not our main purpose to find a best way to ensemble two recommendation models. One remain problem in this part is that since we want to find as rich features as possible we get data from several different data sets, which will cause a divergence of games used in two model. For example, if some games are not existed in steam game data set, then we will not have the features to predict it with the machine learning model but we can get a predicted value from memory-based model. So we will add a simple try except in the predictor code to make sure we can get prediction results from at least one model if we don't have the feature for an input item.

4 Evaluation and Result analysis

The evaluation is established on comparisons. We used 100 random users with sufficient amount of records in [2]. We randomly kept out the latest purchased game from what each users' game libraries. Then, for each game we kept out, we sample 2 keywords or phrase from the game's short description contained in [3] to form a query. We carefully treated these generated queries using Rapid Automatic Keyword Extraction algorithm [14], because the generated queries should not be too specific nor too general, which will make the searching task either too easy or impossible. Next, we leverage each generated query to perform user-based ensemble steam game retrieval based on the corresponding user's records [2]. The performance of the is evaluated based on the mean rank of the latest purchased game in the retrieval results. We tested the using four models: pure search engine (without the help of user-based recommendation), search engine with memory-based model, search engine with machine learning model and search engine with ensemble model. The results are shown in the Tab. 3.

	Method	Average Rank
1	Pure Retrieval	5.23
2	with Memory-based Model	4.62
3	with Machine Learning Model	4.95
3	with Ensemble Model	4.41

Table 3: **Prediction of log-scaled game playtime**

Fig. 12 is a distribution histogram for the pure retrieval system and the one with ensemble model we designed in this project.

We provide two case studies in for evaluation of our model.

Here in Fig. 13 is an example of comparison between retrieved games from pure search engine and search engine with ensemble model. The red box is indicating the target game, "XCOM: Enemy Unknown", to retrieve. The user id to perform this test search is "Cardinal4766." This user had one record of purchasing "Warhammer 40,000: Dawn of War II Chaos Rising" (appid: 20570), which is actually a similarly type with "XCOM", both of them are strategy games with science fiction background. The query we generated based on the short description of the game is "Slingshot,

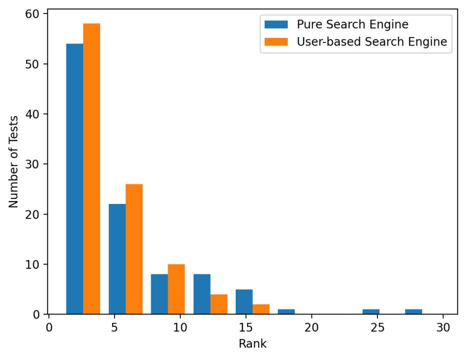
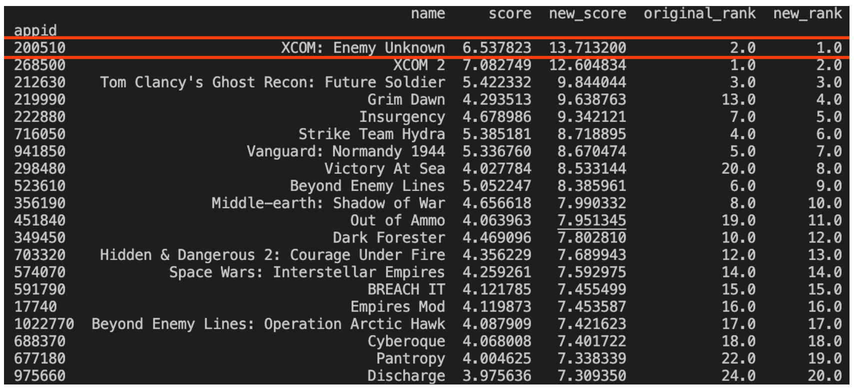


Figure 12: **Ranking distribution of the corresponding system.**

Enemy." In the Fig. 13, "new rank" indicates the rank of the target in the retrieval results with the user-based ensemble model, while "original rank" indicates that rank given by the backbone retrieval system. The result shows that the all "XCOM" series game are scored considerably higher because of the implementation of user-based re-ranking. Also, the first edition "XCOM: Enemy Unknown" stands from the series. Intuitively, with the user-based model, the system understand that "Cardinal4766", a fan of strategy game alien-background with alien background, who also never played "XCOM" series before, acquire high probability to purchase and play the first edition "XCOM: Enemy Unknown."

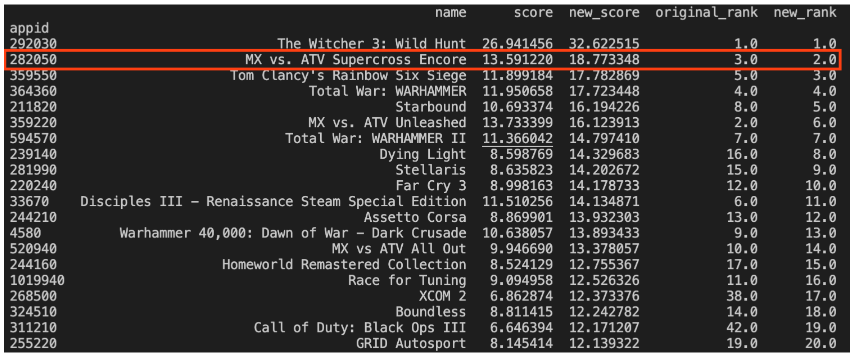

```
                                            name     score  new_score  original_rank  new_rank
appid
200510                       XCOM: Enemy Unknown  6.537823  13.713200            2.0       1.0
268500                                    XCOM 2  7.082749  12.604834            1.0       2.0
212630    Tom Clancy's Ghost Recon: Future Soldier  5.422332   9.844044            3.0       3.0
219990                                 Grim Dawn  4.293513   9.638763           13.0       4.0
222880                                Insurgency  4.678986   9.342121            7.0       5.0
716050                         Strike Team Hydra  5.385181   8.718895            4.0       6.0
941850                    Vanguard: Normandy 1944  5.336760   8.670474            5.0       7.0
298480                            Victory At Sea  4.027784   8.533144           20.0       8.0
523610                        Beyond Enemy Lines  5.052247   8.385961            6.0       9.0
356190             Middle-earth: Shadow of War  4.656618   7.990332            8.0      10.0
451840                               Out of Ammo  4.063963   7.951345           19.0      11.0
349450                             Dark Forester  4.469096   7.802810           10.0      12.0
703320    Hidden & Dangerous 2: Courage Under Fire  4.356229   7.689943           12.0      13.0
574070            Space Wars: Interstellar Empires  4.259261   7.592975           14.0      14.0
591790                                 BREACH IT  4.121785   7.455499           15.0      15.0
17740                                 Empires Mod  4.119873   7.453587           16.0      16.0
1022770  Beyond Enemy Lines: Operation Arctic Hawk  4.087909   7.421623           17.0      17.0
688370                                 Cyberoque  4.068008   7.401722           18.0      18.0
677180                                  Pantropy  4.004625   7.338339           22.0      19.0
975660                                 Discharge  3.975636   7.309350           24.0      20.0
```
Figure 13: **Example 1 of Retrieval Result.**

Similarly to the analysis above, Fig. 14 is another example of the comparison between retrieved games from pure search engine and search engine with ensemble model. In this case, "MX vs. ATV Supercross Encore" (appid: 282050) is target game with generated query "open world maps exclusive waypoint races". The user id to perform this test search is "crayfishh," who is a new user with no past records. The rank of the target game also boosted 1 rank after implementing the user-based model.

5 Conclusion and future work

In our project, we design an ensemble steam game retrieval model. We combine the traditional searching model with user_preference based recommendation model to provide retrieval results that not only close to query but also preferred by users. As the result, our user-based ensemble retrieval system out performed the backbone retrieval system [10]. The ensemble model significantly helps to improve the retrieval results and can discover more user desired games.

For the recommendation model, we build a ensemble model which is the combination of memory-based collaborative model and user-item feature based machine learning model. Unlike most recent researches that trying to discover complex state of the art model such as the attention model in [4],



```
                                                        name      score  new_score  original_rank  new_rank
appid
292030                              The Witcher 3: Wild Hunt  26.941456  32.622515            1.0       1.0
282050                           MX vs. ATV Supercross Encore  13.591220  18.773348            3.0       2.0
359550                           Tom Clancy's Rainbow Six Siege  11.899184  17.782869            5.0       3.0
364360                                   Total War: WARHAMMER  11.950658  17.723448            4.0       4.0
211820                                              Starbound  10.693374  16.194226            8.0       5.0
359220                                   MX vs. ATV Unleashed  13.733399  16.123913            2.0       6.0
594570                                Total War: WARHAMMER II  11.366042  14.797410            7.0       7.0
239140                                           Dying Light   8.598769  14.329683           16.0       8.0
281990                                             Stellaris   8.635823  14.202672           15.0       9.0
220240                                            Far Cry 3   8.998163  14.178733           12.0      10.0
33670     Disciples III - Renaissance Steam Special Edition  11.510256  14.134871            6.0      11.0
244210                                        Assetto Corsa   8.869901  13.932303           13.0      12.0
4580           Warhammer 40,000: Dawn of War - Dark Crusade  10.638057  13.893433            9.0      13.0
520940                                      MX vs ATV All Out   9.946690  13.378057           10.0      14.0
244160                          Homeworld Remastered Collection   8.524129  12.755367           17.0      15.0
1019940                                      Race for Tuning   9.094958  12.526326           11.0      16.0
268500                                               XCOM 2   6.862874  12.373376           38.0      17.0
324510                                            Boundless   8.811415  12.242782           14.0      18.0
311210                             Call of Duty: Black Ops III   6.646394  12.171207           42.0      19.0
255220                                       GRID Autosport   8.145414  12.139322           19.0      20.0
```

Figure 14: **Example 2 of Retrieval Result.**

we focus on using different model to deal with different kinds of features and design a combined recommendation model. We design a light model which combines memory-based model and XGboost machine learning model. And our model is proved to outperform single memory-based method and single machine learning method since it can take advantage of a wide range of user and game features.

Finally, we combine our search model with recommendation model and create evaluation set to analyze the performance of our ensemble model. As the result shown, our ensemble model perform well in providing user-preference based retrieval results as those games preferred by users have a higher rank in our new ensemble model than in the original search model.

Some remaining works that are not done in our project can be the implementation of some state of the art recommendation model. Although our model can be lighter than those complex models and can have a relative good performance, the experiments with some state of the art model can help use modify our model structure to do further improvements on it. Another future improvement we can do on our model is trying more experiments on the ensemble methods such as changing the structure of tuning the combining weights.

Also, we noticed that some successful related work [13] introduced web page click-through statistics for user profiling, and distinguish the query semantics (e.g. Apple the company vs. apple the fruit). In algorithm design, some models adopt the strategy to obtain more accurate negative samples to vote for query results, and feed the user responses back to user database [13], which would be a valuable addition to the game search engine as well.

References

[1] Baptiste R. (2020). Introduction to recommender systems. `https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada`

[2] `https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data`

[3] `https://www.kaggle.com/nikdavis/steam-store-games?select=steam.csv`

[4] Kang, W.K. & McAuley, J.M. (2018) Self-Attentive Sequential Recommendation. `https://cseweb.ucsd.edu/~jmcauley/pdfs/icdm18.pdf`

[5] Wikipedia contributors. (2019). Discounted cumulative gain. Wikipedia. `https://en.wikipedia.org/wiki/Discounted_cumulative_gain`

[6] Wan, M.W. & McAuley, J.M. (2018) Item Recommendation on Monotonic Behavior Chains. `http://cseweb.ucsd.edu/~jmcauley/pdfs/recsys18b.pdf`

[7] Funk, S. (2006) Netflix Update: Try This at Home. `https://sifter.org/~simon/journal/20061211.html`

[8] Kumar V. (2020) Single Value Decomposition (SVD) In Recommender System. `https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/`

[9] Pedregosa F., Varoquaux, G. and Gramfort, A. and Michel, V. andd Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D.

and Brucher, M. and Perrot, M. and Duchesnay, E. (2011) Scikit-learn: Machine Learning in Python `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

[10] Fan S., Yang S., He S.. (2019) "Video game search engine" University of Michigan Fall2019 Information Retrieval Project.

[11] Wikipedia contributors. (2020). Okapi BM25. `https://en.wikipedia.org/wiki/Okapi_BM25`

[12] Wikipedia contributors. (2020). Rocchio algorithm. `https://en.wikipedia.org/wiki/Rocchio_algorithm`

[13] Ng W., Deng L. and Lee D. (2007). Mining User Preference Using Spy Voting for Search Engine Personalization. ACM Transactions on Internet Technologies, Vol. 7, No. 3, August 2007, Pages 1-28

[14] Rose, S. & Engel, D. & Cramer, N. & Cowley, W. (2010). Automatic Keyword Extraction from Individual Documents. 10.1002/9780470689646.ch1.