Steam Recommendation System: Based on Game Features and User Reviews

Group 16: Li Kangfu, Li Zhuoyi, Yu Ting, Zeng Xuran, Zheng Qingchuan

GitHub Link: https://github.com/lzhy1101/bt5153_group_project

Abstract

With the development of video game industry and the diversification of video games, the players-games-matching process has become crucial and challenging for both game players and platforms. Therefore, in our project, we aim at building a Steam recommendation system to recommend 10 games to users, based on their historical reviews and game features. To achieve this purpose, we use three methods (content-based filtering, user-based collaborative filtering and model-based collaborative filtering methods), together with several machine-learning techniques (such as NLP, neural network, and clustering).

As last, we propose two hit ratio metrics and NDCG (normalized discounted cumulative gain) to evaluate the accuracy of our recommendation system. According to the result, the user-based collaborative filtering method outperforms in the three models. Additionally, we suggest that the final selection of recommendation algorithms should also consider the business purpose of the company.

1.1 Background and Purpose

Since the first video game was introduced to consumers in 1971 (Zackariasson & Wilson, 2012), more than a million video games have been created over the forty years of development. Nowadays, playing video games is a popular entertainment option for people, and the video game industry has become one of the most valuable industries in the world (Cheuque et al., 2019), with the revenue surging in recent years (see Figure 1).

One of the reasons behind the success of the industry, is that these video games are highly diversified, and can provide enormous number of options for users. Matching a game to its most suitable player has become a crucial and challenging task for both players and game platforms.

Thus, building a proper recommendation system serves to the following purposes. For **game players**, they could transform from the traditional ways (e.g., advertisements, forums, or friends' recommendations) to a more effective way to find games they like. For **game platforms** (such as Steam), they could help game publishers to find potential suitable players. This would increase the efficiency in players-games-matching process, and therefore create business value and bring in more revenue to the company, as well as the entire video game industry.

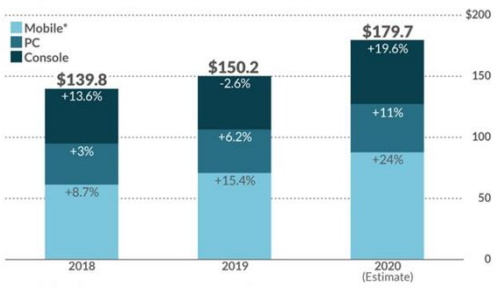

Figure 1. Video game industry revenue from 2018 to 2020 by gaming platform, not including ad. revenue. Numbers in billions. Source: International Data Corporation (IDC).

1.2 Problem Description

In this project, we will build a Steam recommendation system. This system we plan to use will give top 10 recommendations to users, based on their past reviews (i.e., the user's previous records), together with game features.

First, we take a review at relevant literature and summarize common approaches for recommendation systems. Then, we discover the basic facts of our review and game datasets and conduct pre-processing steps for the datasets. Next, we introduce content-based filtering, user-based collaborative filtering and model-based collaborative filtering methods to build recommender systems. At last, we evaluate and compare the three methods, and provide further insights to improve the system.

1.3 Relevant Literature

1.3.1 Recommendation System

In recent years, recommendation systems have changed the way of interaction between websites and their users. They are widely used in online retail platform such as Amazon, Wal-Mart and Netflix, and have a strong impact on customer choices (Fleder et al., 2010). It is a win-win game for both customers and companies. For customers, it could help them recognize new products and find desirable products from bunch of choices (Pham & Healey, 2005). For companies, recommendation systems enable them to

realize potential profit by transforming browsers into buyers, allowing cross-selling of products, and increasing the loyalty of customers (Schafer et al., 2001).

1.3.2 Recommendation Algorithms

Traditional personalized recommender system employs the collaborative filtering (CF) algorithm. The CF algorithm recommends to users by making full use of the information provided by other users who are similar to them (Herlocker, Konstan, Borchers, & Riedl, 1999). A survey shows that recommender systems could be classified into three types: content-based, collaborative filtering and hybrid approaches (Adomavicius & Tuzhilin, 2005).

1.3.3 Game Platform: Steam

The video game industry grows sturdily in decades, but still face problems. According to STEAM registries of 2014, about 37% of games purchased have never been played by the users who bought them. The problem arises from the mismatching between game information and users' interest, which creates an urgent need for the presence of brilliant recommender systems. Cheuque et al. (2019) proposed recommender models based on factorization machines, deep neural networks and the mixture of both for Steam, and all the algorithms outperformed the baseline based on alternating least squares model.

1.4 Innovation Points and Interesting Aspects

The content of our project is novel and meaningful in the following three aspects. **First**, under the context of COVID-19, with the time spent at home increasing, users allocate more time for the online Steam platform. This has led to new highs for monthly active users (120.4 million), daily active users (62.6 million), peak concurrent users (24.8 million), first-time purchasers (2.6 million per month), hours of playtime (31.3 billion hours), and the number of games purchased (21.4% increase over 2019) in Steam[1]. Since we are using the new review dataset from 2021, we may come up with a better recommendation system which incorporate more valuable and brand-new characteristics in such an unusual situation. **Second**, we combine two datasets for reviews and games to get more features. This can provide additional information which may not be used in previous literature. **Third**, except for the traditional recommendation algorithms, we apply NLP techniques to make full use of the reviews posted by users and game descriptions provided by Steam, to make more personalized recommendations for customers.

2. Datasets

To build the recommendation system, we use two datasets on game reviews and games from Kaggle as follows.

2.1 Review Dataset

2.1.1 Source

The dataset on user reviews we are using is *Steam Reviews Dataset 2021*[2], a relatively new dataset updated on Jan 2021, by Marko M. It is obtained using Steam's provided API outlined in the Steamworks documentation[3]. The full dataset is 7.61 GB, with around 21 million user reviews of about 300 different games on Steam.

2.1.2 Description

For the original review dataset, after deleting the non-English reviews with the column *language* (this column will also be deleted after that), the dataset has 9,635,437 rows and 21 columns as follows. Other pre-processing steps will be introduced in Section 3.

Table 1. Review dataset description.

Column	Description	Type
app_id	App (game)[4] ID	int
app_name	App name	str
review_id	Review ID	int
review	Review text of the game	str
timestamp_created	Review creation time	int
timestamp_updated	Review updated time	int
recommended	Whether the review recommends the app	bool
votes_helpful	Number of helpful votes of the review	int
votes_funny	Number of funny votes of the review	int
weighted_vote_score	Score based on number of helpful votes	float
comment_count	Number of comments for the review	int
steam_purchase	Whether the author purchased the app	bool
received_for_free	Whether the author received the app for free	bool
written_during_early_access	Whether review was written on early access	bool
author.steamid	Review author's ID	int

[1] *Steam - 2020 Year in Review* could be retrieved from https://store.steampowered.com/news/group/4145017/view/2961646623386540826.

[2] *Steam Reviews Dataset 2021* can be downloaded from https://www.kaggle.com/najzeko/steam-reviews-2021.

[3] Steamworks documentation on reviews can be viewed at https://partner.steamgames.com/doc/store/getreviews.

[4] The definition of Steam app and app ID can be viewed at https://steamdb.info/apps/. In our project, all the games (i.e., products) are apps, not bundles.

AUTHOR.NUM_GAMES_OWNED	Number of games the author owns	int
AUTHOR.NUM_REVIEWS	Number of lifetime app reviews by the author	int
AUTHOR.PLAYTIME_FOREVER	Author's total playtime of the reviewed app	float
AUTHOR.PLAYTIME_LAST_TWO_WEEKS	Author's playtime of the app in the last two weeks	float
AUTHOR.PLAYTIME_AT_REVIEW	Author's playtime of the app before the review	float
AUTHOR.LAST_PLAYED	Author's playtime of the app last time	float

2.2 Game Dataset

2.2.1 SOURCE

The game dataset we are using is *Steam games complete dataset*[5], updated by Alexander Antonov 2 years ago. It contains more than 40,000 games from Steam shop with detailed data. The size is 77.95 MB.

2.2.2 DESCRIPTION

Firstly, to merge this game dataset with the review dataset later, we extract the unique app ID from column *url*[6]. Secondly, because all the games in the review dataset are apps, not bundles, so we drop the column *types*. After these steps, the game dataset has 40,833 rows and 20 columns.

Table 2. Game dataset description.

COLUMN	DESCRIPTION	TYPE
APP_ID	App (game) ID	int
NAME	App name	str
URL	Game URL	str
DESC_SNIPPET	Short description	str
GAME_DESCRIPTION	Detailed description	str
RELEASE_DATE	Release date	date
DEVELOPER	Developer	str
PUBLISHER	Publisher or publishers	str
GENRE	Genre(s) of a game	str
POPULAR_TAGS	Popular tags	str
GAME_DETAILS	Detailed tags	str
LANGUAGES	Supported languages	str
ACHIEVEMENTS	Number of achievements	int
MATURE_CONTENT	Mature content description	str
MINIMUM_REQUIREMENTS	Minimum specifications	str
RECOMMENDED_REQUIREMENTS	Recommended specifications	str
ORIGINAL_PRICE	Price without discount	float
DISCOUNT_PRICE	Price with discount	float
RECENT_REVIEWS	Recent reviews	str
ALL_REVIEWS	All reviews	str

2.3 Exploratory Data Analysis

Recommendation: Figure 2 shows the count of different review sentiments (extracted from features *recent_reviews* and *all_reviews*), with much more positive reviews than negative ones, especially for the 'very positive' reviews. Figure 3 is the recommendation percentage histogram for each game (calculated from column *recommended*), and most games have relatively satisfying recommendation percentage.

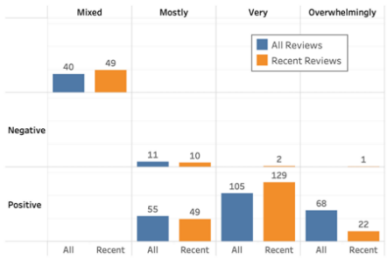

Figure 2. Count of different review sentiments.

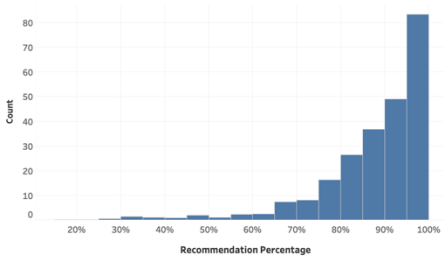

Figure 3. Percentage of recommendation histogram.

Review length: Figure 4 shows the length of reviews. Most reviews are short and have a few words (with the medium of 88 words), while the dataset also has some long reviews, with the maximum to 8,000 words.

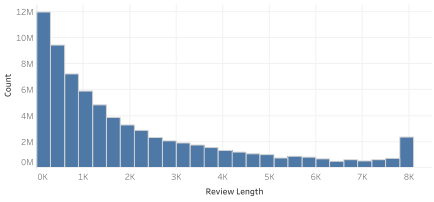

Figure 4. Review length histogram.

[5] *Steam games complete dataset* can be downloaded from https://www.kaggle.com/trolukovich/steam-games-complete-dataset.

[6] The urls of the Steam apps are in the format of http://store.steampowered.com/app/**appID**/appName/.

Review created time: Figure 5 shows the time trend of the count of reviews. The earliest review was created on Nov 20, 2010, and the latest review was created on Jan 13, 2021. The time series has a basically upward trend. Later, we will use the created time of reviews in splitting the dataset.

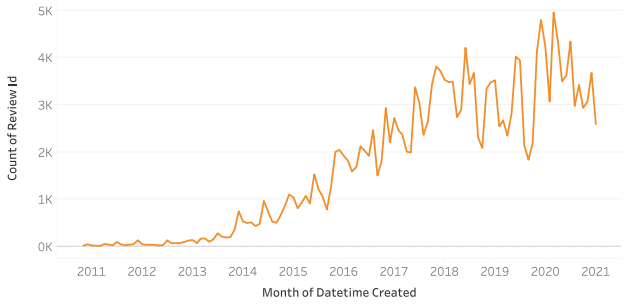

Figure 5. Time series of the count of reviews.

3. Pre-Processing Steps

3.1 Clean

The cleaning procedures for the **review dataset**:

- Drop null values and duplicated rows.
- Drop non-English rows (as mentioned before).
- Change timestamp to actual datetime format.

The cleaning procedures for the **game dataset**:

- Check duplicated rows: no duplicated rows.
- Extract app ID from *url* (as mentioned before).
- Check game types: apps (as mentioned before).
- Because *recent_reviews* and *all_reviews* columns are strings, we split these columns to review types, number of the types, and percentage of the types, respectively (i.e., expand 2 to 6 columns).

3.2 Define Threshold

To prepare enough data for the recommendation system, we need enough reviews for each user, and enough users for each game. With these purposes, we set two thresholds for the filtered dataset:

- **Each user** should have at least 15 game reviews.
- **Each game** should have at least 500 users who have reviewed it.

3.3 Merge

At last, we merge the review and game datasets, using inner join on *app_id*. Because two datasets all have app names, we keep the one from the game dataset.

After these steps, the final dataset has 206,997 rows (i.e., unique reviews), with 43 features mentioned above. In this dataset, we have 136 unique games and 10,525 unique authors for further exploration.

Table 3 Descriptive statistics of reviews, users and games.

Metrics	Number of Reviews for Each User	Number of Users for Each Game
Mean	19.667	1522.037
Std	5.789	1002.749
Min	15.000	500.000
20%	16.000	815.750
50%	18.000	1236.000
75%	21.000	1787.500
Max	103.000	5094.000

3.4 Split

To evaluate the performances of different recommender systems, we split the dataset into training set and testing set.

Training set: For each of the users in dataset, 80% of his/her earlier records would be assigned to training set according to review date.

Testing set: The **latest 20%** of records for each user would be assigned to testing set.

To stress, the split of training and testing data for each user is based on his/her review date for games, as we assume when the recommendation engine makes recommendations for a specific customer, it only possesses the past review data of the user to help it make final decisions.

To sum up, we have 10,525 unique users and 136 unique games in both training set and testing set (as mentioned before). And after splitting the data, we have 165,990 records (i.e., rows) for training set and 41,007 records for testing data.

4. Recommendation Algorithms

In this report, we apply the Top-N recommendation method to help Steam recommend the right products to users, based on their historical customer behavior. We set N to be 10, which means for each user, we would recommend 10 games according to his/her previous records. Here we require the recommended games for each user could not intersect with the existing games in the training set for that user.

4.1 Content-Based Filtering

Content-based filtering is a widely used recommendation method. It is a technique that uses similarities in content features to make decisions.

4.1.1 Item Representations and Cosine Similarities

For our content-based recommendation system, it is crucial to represent the games with certain vectors. In our case, we will transfer the three features (game publisher, genre and game description) into vectors using NLP technique and neural network method.

Genre vector: Game genre is a column in the dataset that contains the categories such as action, adventure, indie etc. It roughly defines which category the games belong to. We use one-hot encoding with zero and one to represent this feature as a vector.

Publisher vector: Game publisher performs an important role in a game's life cycle. Good publisher can boost the sales of games, and many consumers will purchase the games with certain publishers. Thus, we take this feature into account. The first step to vectorize the publisher feature is to use label encoder to encode the publisher strings into values. Then we set these values as input and set up the game ratings as target variables to train a neural network. We set up the embedding size is 30. Finally, we extract the embedding layers with respect to different publishers as the vector representations. The neural network structure is as follows:

Table 4. Neural network structure.

Layer (Type)	Output Shape	#Param
Embedding	(None,1,30)	3030
Flatten	(None,30)	0
Dense	(None,50)	1550
Dense_1	(None,15)	765
Dense_2	(None,1)	16

Game description vector: Game description contains hidden information that summarize the game's uniqueness. The first step to vectorize the game descriptions is to train a word2vec model. We set up the parameters as follows: size=250, window=5, min_count=2, workers=4. Finally, in order to solve the issue that the lengths of descriptions are not the same, we sum up the weights for words in a description and calculate its average. After this step all the inputs will have the same dimensionality.

After connecting the three vectors, we get a long vector of size **295 x 1** to represent each game. With these vectors representations we can now compute and save the cosine similarities between each game.

4.1.2 Recommendation Algorithm Based on Voting

In our case, every user in the training set has records playing more than one game. Therefore, we design the algorithm: as the user has played many games, for each game we search its top 10 related games based on the similarities scores. Finally, we rank the result and select the top 10 games with majority of votes. In case the votes for the games are the same, we compute the average similarities score for games to ensure that every game has different rankings. There is an example following:

Table 5. Recommendation ranking example

Ranking	game ID	#votes	similarities score
1	638970	5	0.711
2	814380	4	0.753
3	424840	4	0.706

At last, the system will recommend top-10 games to a user.

4.2 User-Based Collaborative Filtering

4.2.1 Algorithm

User-based collaborative filtering method recommends items (i.e., games) by finding similar users to the target user.

The algorithm will be conducted with the two tasks:

- Find k similar users (here we set k=3) to the target user by using a similarity function to measure the distance between each pair of users.

  To achieve this goal, first, we compute two user **distance matrixes** based on **user-item matrix** and **user-attribute matrix**, respectively. Next, by combining the two user distance matrices with weightage assignment, we get our final **distance matrix** – from where we find the K similar users.

- Predict the ratings that the target user will give to a set of items, which the K similar users rated but target user does not, and then recommend the top N items to the target users.

4.2.2 Find Similar Users

User-item matrix: To create user-item matrix, we create pivot table by taking user id as columns, app name as index, and rating score as values. The rating score will be set to be 1 if recommended, and -1 if not recommended. Finally, we get a sparse user-item matrix with size of (136, 10525).

Pearson correlation coefficient is used as the distance measurement, as Jaccard similarity ignores the specific value of the rating and cosine similarity treats missing values improperly.

User-attribute matrix: The raw user attributes we selected in this matrix are: *review*, *steam_purchase*, *received_for_free*, *author.num_games_owned*, *author. num_reviews*, *author.playtime_forever*, *author.playtime_ last_two_weeks*, *author.playtime_at_review*, *author.last_ played*.

We use VADAR-sentiment analyzer[7] to do user review sentiment analysis. The VADAR-sentiment analyzer is a lexicon and rule-based sentiment analysis tool that consist of a large library of words which are classed as negative, neutral, or positive. Here we use the negative words percentage of user review to reflect general attitude of user to games.

Then we use label encoder to pre-process categorial variables and standardize numerical variables.

Similar as the user-item matrix, we summarize user attributes using pivot table, by taking user id as columns, attributes as index, and the average of user attribute values as the values. For their attribute value, we also use Pearson correlation coefficient to compute distances between users.

Combined distance matrix: The combined distance matrix is computed by summing up of **user-item** Pearson correlation coefficient matrix and **user-attribute** Pearson correlation coefficient matrix, with assigned weightage of 0.8 and 0.2, respectively. We assign a smaller weightage to user-attribute distance matrix as they are not significant user attributes. If we can get better user attributes such as gender and age, we will assign a higher weightage.

We set 3 most similar users for each target user based on largest combined correlation coefficient distances.

4.2.3 Compute Weighted Rating

After getting the similarity distances, we compute weighted rating from ratings score of each similar users and combined distance matrix to 3 similar users for each target users.

$$\boldsymbol{r_{xi}} = \frac{\sum_{y \in N} \boldsymbol{S_{xy}} \cdot \boldsymbol{r_{yi}}}{\sum_{y \in N} \boldsymbol{S_{xy}}}$$

$$\boldsymbol{S_{xy}} = \boldsymbol{sim}(\boldsymbol{x}, \boldsymbol{y})$$

where r_{xi} is the weighted rating of *user x* to game *i*, r_{yi} is the actual rating of *user y* to game *i*, *y* is the similar user to *user x*, S_{xy} is the Pearson correlation coefficient between *user x* and *y*.

Based on weighted rating, we filter and recommend 10 games that is not played before.

4.3 Model-Based Collaborative Filtering

4.3.1 Algorithm

We used Latent Factor Model (LFM) to conduct model-based collaborative filtering. LFM is based on matrix factorization. By inferring from the game-user rating matrix, LFM tries to discover k number of the hidden factors and establish relationship between these factors with games and with users.

Data used in this model is only *app_id* (game ID), *author.steamid* (user's steam ID) and *recommend* (the Boolean value that whether the user recommends a game).

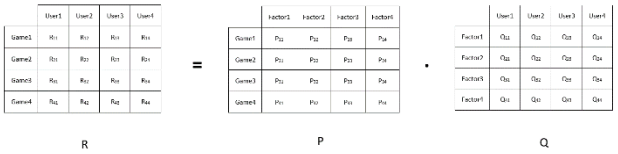

Figure 6. Game-user rating matrix factorization.

The matrix R represents the game-user rating matrix, which is obtained from original dataset. The model training is done by breaking down the matrix R, as illustrated in the Figure 6, into matrix P (the game-factor ownership matrix) and matrix Q (the user-factor preference matrix), with given k factors.

The training is done iteratively. Matrix P and Q will be updated in each iteration using Stochastic Gradient Descend (SGD). The dot product of updated matrix P and Q will then produce a predicted matrix $\hat{\boldsymbol{R}}$. Root mean square error (RMSE) is calculated using difference between the original matrix R and predicted matrix $\hat{\boldsymbol{R}}$. The convergence of the model and finding of optimum hyperparameters will be determined by a lowest RSME.

4.3.2 Procedures

4.3.2.1 Game-User Rating Matrix

The "True" value of the recommendation is considered as a positive rating and replaced with 1, whereas the "False" value is treated as negative rating and replaced with 0. Game ID and User ID require transformation too, as they are very large and not in sequence. These IDs are sorted and remapped to their index. These remapped game Id and user's Steam Id will be used subsequently in the model.

After these treatment on data, a sparse matrix will be created. The row index will be all the remapped game Ids, and the column index will be the remapped user's Steam Ids. The values of the matrix will be within the range of **[**0, 1**]**. If game g is recommended by user u, then the game-user rating matrix R will have R[g][u] = 1, otherwise R[g][u] = 0.

4.3.2.2 Matrix Factorization

The operation in LFM is to find the matrix P and matrix Q that can fit best in Matrix R over iterations, as shown in the equation.

$$\boldsymbol{R_{GU}} = \boldsymbol{P_G Q_U^T} = \sum_{\boldsymbol{k=1}}^{\boldsymbol{K}} \boldsymbol{P_{G,k} Q_{k,U}^T}$$

K is the total number of factors we assume the system has. R_{GU} is the game-user rating matrix. $P_{G,k}$ is the game-factor

[7] VADAR sentiment analyzer description and user manual: https://github.com/cjhutto/vaderSentiment.

ownership vector for k[th] factor. $Q_{k,U}$ is the transposed user-factor preference vector for k[th] factor.

In order to learn the P and Q matrix, the following function will be used as the loss function:

$$L = \sum (R - \hat{R})^2 = \sum (R - \sum_{k=1}^{K} P_{G,k} Q_{k,U}^T)^2 + \lambda \|P\|^2 + \lambda \|Q\|^2$$

The expression $\lambda \|P\|^2 + \lambda \|Q\|^2$ is used as a regularization to avoid overfitting. The value of λ is one of the hyperparameters of the model and has to be tuned.

4.3.2.3 UPDATE MATRIX P & Q

Stochastic Gradient Descend (SGD) is a commonly used method to update parameters. In LFM, SGD is used to update matrix P and Q. It is performed using the following equations:

$$\Delta P_{G,k} = \frac{\partial L}{\partial P_{G,k}} = -2(R - \sum_{k=1}^{K} P_{G,k} Q_{k,U}^T) Q_{k,U}^T + 2\lambda P_{G,k}$$

$$\Delta Q_{k,U} = \frac{\partial L}{\partial Q_{k,U}} = -2(R - \sum_{k=1}^{K} P_{G,k} Q_{k,U}^T) P_{G,k} + 2\lambda Q_{k,U}^T$$

$$P_{G,k} = P_{G,k} + \alpha \Delta P_{G,k}$$

$$Q_{k,U}^T = Q_{k,U}^T + \alpha \Delta Q_{k,U}$$

The value of α is also a hyperparameter of the model and has to tuned. α is also called the learning rate. It determines how fast P and Q converges.

4.3.2.4 FIND THE BEST MATRIX P & Q

The LFM model is trained over epochs. A prediction matrix $\hat{R}$ is calculated using the dot product of updated matrix P and Q. RMSE is calculated for each epoch using the formula:

$$RMSE = \frac{1}{C} \sum_{g=1}^{G} \sum_{u=1}^{U} (R_{gu} - \hat{R}_{gu} + \mu_u)^2$$

Where G is total number of games, U is total number of users, R is the original game-user rating matrix, $\hat{R}$ is the predicted game-user preference matrix, μ is the user average rating vector for each game and C is the total number of none-zero value in the game-user rating matrix.

Training stops on the epoch produced the lowest RMSE value. The model will run several times with different hyperparameters to find the optimum hyper parameter set (shown in the table below).

Table 6. LFM Hyperparameters

HYPERPA-RAMETER	DESCRIPTION	OPTIMUM
K	Number of factors	20
EPOCH	Number of epochs	10
α	Learning rate that determines how fast P and Q converge.	0.001
λ	Degree of regularization that prevents overfitting.	0.01

4.3.2.5 RECOMMENDATION

Recommendation is performed with prediction Matrix $\hat{R}$. Matrix $\hat{R}$ will be a matrix with values representing the degree of preference of a user to a game. We will take the top 10 preference from a user's game preference vector as the final recommendation using the formula below:

$$\hat{R}^* = argsort_{u \in U}^{10} \hat{R}_{G,u}^T$$

where U represents all users, and G represents all games.

5. Results and Evaluation

5.1 Evaluation Metrics

To evaluate the performances of different recommendation algorithms, we conduct the offline test to compare their performance.

In this report, we select **Hit Ratio** (HR) and **Normalized Discounted Cumulative Gain** (NDCG), which have been commonly used in Top-N recommendation as the evaluation metrics for the algorithms.

5.1.1 HIT RATIO

Given a user, each recommendation algorithm would produce a list of ten games for the user. If a test game appears in the recommended list, we call it a **hit**.

To compute HR, we apply two different methods:

- **Method 1:** As long as one of the true games appears in the recommended list for a user, we define the user as a hit user. This metric guarantees at least one game is recommended right for the user. HR (method 1) is calculated as below:

$$HR(method\ 1) = \frac{Number\ of\ Hit\ Users}{Number\ of\ Users}$$

- **Method 2:** If a true game appears in the recommended list for a user, we call it a hit game for that user. The hit ratio for each user is defined as the proportion of hit games in true games for the user. The overall hit ratio is the average of hit ratios for all the users. This metric helps us to evaluate the quality of recommendations for each user. HR (method 2) is calculated as below:

$$HR(method\ 2) = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{Number\ of\ hit\ games_i}{Number\ of\ true\ games_i} \right)$$

5.1.2 NORMALIZED DISCOUNTED CUMULATIVE GAIN

As Hit Ratio is a recall-based matric, it fails to consider the importance of getting top ranks correct, which is crucial in real-world applications.

Hence, we introduce Normalized Discounted Cumulative Gain (NDCG), which helps to measure the ranking quality of the recommended lists. It assigns higher importance to recommendations in top ranks and low importance to results in lower positions. NDCG is calculated as below:

$$NDCG = \frac{1}{N}\sum_{i=1}^{N}(Z_K \sum_{j=1}^{K} \frac{2^{r_j-1}-1}{log_2(j+1)})$$

where Z_K is the normalizer to ensure the perfect ranking has a value of 1; r_i is the graded relevance of games at position *i*. In our report, we set *K* to 10. To simplify the relevance, we set it as a binary relevance in this report: if the recommended game is in the testing set, the relevance r_i would be 1 and 0 otherwise.

To compute the overall NDCG for algorithms, we only consider users that we have recommended right for at least once, and then take averages of all users, as NDCG is employed to evaluate the rank of right recommendations in the recommendation lists.

For all three metrics we mentioned, lager values would lead to a better recommendation result.

5.2 Recommendation Results and Evaluations

For all the recommendation algorithms, three evaluation metrics have been computed to compare their performance. The final results are presented as below.

Table 7. Evaluation metrics for recommender algorithms.

METHOD	HR1	HR2	NDCG
CONTENT-BASED	0.28	0.07	0.45
USER-BASED CF	0.46	0.12	0.44
MODEL-BASED CF	0.26	0.07	0.46

HR1: As shown in Table 6, we could find that the user-based collaborative filtering algorithm achieve highest Hit Ratio1, which means the user-based collaborative filtering (CF) algorithm performs best in recommending at least one right game to the users. In other words, 46% of the customers could be satisfied with at least one ideal game by user-based CF recommendation system.

HR2: However, when it comes to Hit Ratio2, which is used to evaluate the recall of recommendation list for each of the content-based, user-based and model-based CF algorithms win the higher hit ratio at around 7%, while the result of user-based CF is relatively low.

NDCG: For Normalized Discounted Cumulative Gain, the results for three algorithms are quite similar (at around 45%), showing that there is no strong difference between the quality of ranked recommendation list for three different methods.

Above all, the final decision of recommendation algorithm should depend on **the purpose** of the recommender system. If Steam aims to hit more customers, regardless the numbers of correct recommended games (i.e., provide at least one ideal game to as many customers as possible), Hit Ratio (method 1) should be selected as the final evaluation metric. However, if Steam attaches more importance to the accuracy of recommendations for users, Hit Ratio (method 2) should be the key metric, as it focuses on recommending as many correct games as possible to suitable customers. In our case, **user-based collaborative filtering algorithm** outperform the other two methods based on both 2 kinds of hit ratios, therefore is selected as our final recommendation algorithms for Steam.

6. Insights and Discussions

6.1 Limitations

6.1.1 CONTENT-BASED FILTERING

- The range of the games recommended by the system is quite narrow. The consumers may have multiple preferences, but this system cannot bring surprising recommendations to them, since we barely recommend games based on the similar contents.
- If there is a new user coming in, we don't know what the item is to recommend. Therefore, the cold-start problem exists here.

6.1.2 USER BASED COLLABORATIVE FILTERING

- It is a memory-based model, that is to say, it uses the entire database every time when it makes a prediction, so it needs to be in memory. Therefore, it takes a lot of computational power.
- The algorithm relies on user's features to find similar users and recommend the items they interacted with in a positive way, therefore being robust to the new user case. However, if the user base is small, there will not be many users to select as similar users. Cold start problem takes place.

6.1.3 MODEL BASED COLLABORATIVE FILTERING

- Although the model is able to recommend some games for new users, it also has cold-start difficulties to recommend new games based on their popularity efficiently.

- The model has to be retrained when new data is captured, which also leads to the runtime and memory issue.

6.1.4 UNIVERSAL LIMITATIONS

- The models were built on existing user-game recommendation data, and one of the assumptions is that a user will vote for recommendation if the user likes the game. However, there are many users who do not comment or vote when they have played the game. These users are, therefore, not represented in the model.
- The input data used might be significantly biased towards recommendation, as the vast majority of users vote for recommendation, while only a few users voted not recommended. This is a common user behavior that a user tends to not comment or vote rather than vote negatively.

6.2 Improvements

As mentioned in the limitation part, the memory-based model is very expensive to compute at run time. To mitigate the issue, we separate the computation of three distance matrices with the rest parts. To make further improvements, we can use GPU instead of CPU to do parallel computing.

It is worth mentioning that we can come up with a hybrid recommendation system combining multiple methods. The hybrid system has different strategies for different users. It can solve the cold start issue to some extent. For example, if there is a new user coming in, we can recommend the most popular games sold this week. If the user only leaves a few operations, we might use the content-based filtering. And if the user has purchased or commented enough number of games, we can use collaborative filtering. We may also introduce some randomness in the recommendations to give some surprising suggestions.

References

Adomaviciu, G. and Tuzhilin, A. *Incorporating contextual Information in Recommender Systems Using a Multidimensional Approach.* ACM Trans. Information Systems, vol. 23, no. 1, Jan. 2005.

Cheuque, G., Guzmán, J., and Parra, D. *Recommender systems for Online video game platforms: The case of STEAM.* In Companion Proceedings of The 2019 World Wide Web Conference, pp. 763–771, San Francisco, CA, 2019.

Feitas, Scott and Clayton, Benjamin. 2018. Anime Recommender System Exploration: Final Report. s.l. : Github, 2018.

Fleder, Daniel, Kartik Hosanagar. *Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. Manage. Sci.* 55(5) 697–712, 2009.

Herlocker, J., Konstan, J.A. and Riedl, J. *An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms.* Information Retrieval 5, 287–310, 2002.

Pham, A., Healey, J. *Tell You What You Like.* Los Angeles Times, 2005.

Schafer, B., Konstan, J. and Riedl, J. *E-Commerce Recommendation Applications.* Data Mining and Knowledge Discovery 5, 115–153, 2001.

Yu, Hanqiao. *Latent Factor Model for Book Recommendation System ---Taking Douban as an Example.* Proceedings of the 2019 International Conference on Education Science and Economic Development (ICESED 2019), pp. 221–226, Atlantis Press, 2020.

Zackariasson, P. and Wilson, T. L. (eds.). *The video game industry: Formation, present state, and future.* Routledge, 1st edition, 2012.