



ISS Project (Graduate Certificate in Intelligent Reasoning Systems)

Steam Game Recommendation System

Peng Shaoze (A0261840J)

Tian Yuyang (A0261848U)

Xu Xindi (A0261833E)

Zhang Xingyu (A0261781B)



National University of Singapore | Institute of Systems Science

<https://github.com/zxypro1/SteamGamesRec>

Executive Summary

In the era of information overload, users can obtain massive game information through multiple channels. If users have a clear purpose, search is a very convenient and quick way to retrieve information. However, with the popularity of smartphones and mobile networks, users get more and more fragmented game time, and "killing time" has become the core goal of many products such as Steam.

People are lazy, if the game recommendation system can filter out irrelevant content for users in a timely manner, realize the game distribution function of thousands of people, help users filter and recommend games of interest, and let the right information find the right person, then this recommendation capability will significantly improve the user experience.

The Steam platform is a game and software platform that Valve Corporation hired BitTorrent (BT download) developer Bram Cohen to develop and design himself. The Steam platform is one of the world's largest comprehensive digital distribution platforms. Players can buy, download, discuss, upload and share games and software on the platform. On October 23, 2022, according to Steam statistics, the number of online players on the Steam platform in China exceeded the 30 million mark on the evening of the 23rd, reaching 30,013,151, setting a new historical record.

This project implements game recommendation and distribution algorithms based on a dataset containing more than 8,000 popular games on the steam game platform and 20,000 user information, including user-based recommendation, content (game)-based recommendation, and description-based recommendation. At the same time, we use JavaScript, Css, Html, React framework and Flask lightweight framework to achieve front-end and back-end development of the project.

CONTENTS

1. Demand Analysis	5
1.1. Development Background	5
1.2. Market / User Analysis.....	5
1.2.1. Target Users	5
1.2.2. Market / User Analysis.....	6
1.3. Demand Analysis	7
1.3.1. Functional Requirements	7
1.3.2. Non-functional Requirements	7
2. Product Positioning.....	7
2.1. Recommended System Description	7
2.1.1. The Main Function.....	7
2.1.2. Advantages and Innovations	7
2.2. User Scenario Requirements Analysis	8
2.3. Practical Problem Solved.....	8
3. Data Loading and Data Preprocessing.....	9
3.1. Data Loading.....	9
3.1.1. Obtain data	9
3.1.2. User-items data	9
3.1.3. Items detail data	9
3.2. Data Preprocessing.....	10
3.2.1. Games data.....	10
3.2.2. User items Data.....	10
4. System Design and models	11
4.1. System Design	11
4.1.1. User-based recommendation.....	12
4.1.2. Item-based recommendation.....	12
4.1.3. Text-based recommendation	13
4.2. Database Design.....	13
4.2.1. Table Structure	13
4.3. Model	14
4.3.1. Recommendation model built by LightFM.....	14
4.3.2. NLP model built by spaCy	17
4.4. Backend.....	19
4.4.1. Overview	19
4.4.2. APIs.....	20
4.5. Frontend	22
4.5.1. Overview	22
4.5.2. Deployment.....	23
5. Conclusion	23
5.1. Summary of Achievements	23

5.2. Future Improvements	23
6. Appendix	24
6.1. Proposal.....	24
6.1.1. Problem.....	24
6.1.2. Solution.....	24
6.1.3. Objective	24
6.2. System Functionalities Mapping.....	24
6.3. Installation and User Guide	25
6.4. Individual Reports.....	29

1. Demand Analysis

1.1. Development Background

In 2021, the number of mobile game users in China alone will exceed 730 million, a year-on-year increase of 2.8%. According to the "48th China Internet Development Statistical Report" released by CNNIC, as of June 2021, the number of mobile Internet users in China has reached 1.007 billion, and the penetration rate of mobile game users among mobile Internet users has reached 72.3%. Mobile games have become one of the most common online entertainment methods for netizens. With the continuous increase in the scale of game players, the vigorous development of game recommendation systems has been promoted.

For a long time, as the Steam platform is the world's largest comprehensive digital game software distribution platform, it is difficult for players to filter out their favorite games due to the complicated and complicated game labels and game versions, as well as the fuzzy and inconsistent perception of individual games. , especially for new users, because they do not understand the game content of the game system or even the game attributes reflected by the game tags, it is difficult to use traditional tags to filter out the games they try for the first time. In addition, the current traditional game recommendation system often recommends popular games or filters according to game tags, and does not involve a comprehensive recommendation algorithm. This project mainly uses a graph neural network to construct embeddings vectors for game-based recommendation. The NLP word segmentation algorithm further implements description-based game recommendation.

The game system based on the web service built on the Flask lightweight framework as the carrier, the user/game-based recommendation algorithm and the traditional filtering and screening mechanism implemented by the graph neural network and NLP algorithm, can avoid the traditional game recommendation system as much as possible. Therefore, by analyzing the game data information and user data information on the Steam platform, we understand the business logic and technical implementation of traditional game recommendation, and summarize the existing problems at this stage. Based on this, we have implemented this project, namely game recommendation. system. Provide Steam game players with a game recommendation system that integrates game-based recommendation, user-based recommendation, and description-based recommendation, and provides a one-stop game recommendation service and platform for the majority of Steam players.

1.2. Market / User Analysis

1.2.1. Target Users

In the past five years, the number of global game users on the Steam platform has continued to rise from 90 million to 130 million, and there are tens of millions of new

users growing every year. The core target users of the game recommendation system of this project are all users of the Steam platform, and provide new users on the Steam platform. A game recommendation mechanism based on friends/content is introduced. Due to the limited team members, it only targets users on the Steam platform and chooses to deeply cultivate related users. This model can also empower various game platforms, such as WeChat game applet users. Strong scalability, so the target users in the future will be positioned as players on various game platforms around the world to achieve personalized game recommendations.

1.2.2. Market / User Analysis

In order to help the team clearly and clearly understand the needs and pain points of Steam game players for the product concept of the game recommendation system Web service proposed by us, we have done preliminary user research and market research, through user interview research and user data analysis methods , and conducted research and promotion through personal social media, and conducted in-depth user demand analysis based on the results of data analysis and user interviews. In addition, the second most popular word related to game recommendation in the Baidu Index is the Steam platform game recommendation, which feedback the target users. high-frequency focus needs. After the above research, the team clarified the product positioning and the needs of Steam players and users, and completed the priority grading of needs by weighing multiple dimensions.

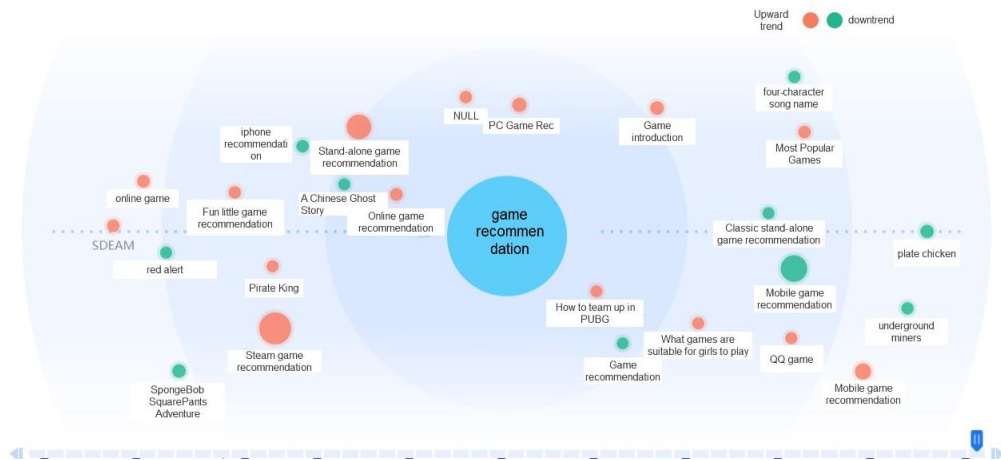


Figure 1.1: Game recommendation related words

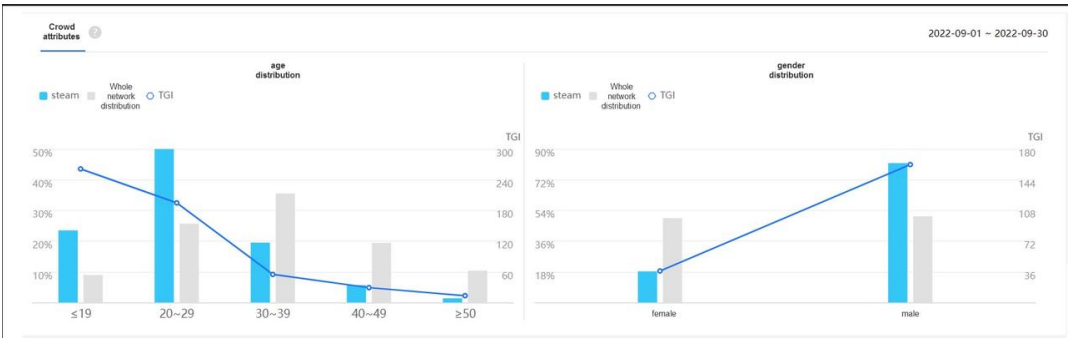


Figure 1.2: (Baidu Index) Steam crowd portrait

1.3. Demand Analysis

According to the feedback data from previous data analysis and user interviews, as well as through competitive product analysis (the traditional recommendation mechanism for games on the Steam platform), the user needs are clearly defined as follows.

1.3.1. Functional Requirements

- Game-based recommendation, the user selects the game he is interested in after filtering according to the game tag, and makes a relevant recommendation according to the game selected by the user;
- Based on the user's recommendation, the user selects other user IDs or friend IDs in the database, and makes recommendations based on the game data owned by the selected user;
- Content-based recommendation, the user enters personalized description content, including personal character, preferred game tags, etc., and recommends according to the relevance of the tag after word segmentation;

1.3.2. Non-functional Requirements

- Reliability: Steam games and user data information are true, and false information is eliminated;
- Interactivity: The functional logic jump conforms to the user's daily Web operations;
- Robustness: ensure the security and stability of information storage through Alibaba Cloud;

2. Product Positioning

2.1. Recommended System Description

2.1.1. The Main Function

- Game-based recommendations;
- User-based recommendations;
- Recommendation based on content (description);

2.1.2. Advantages and Innovations

Advantages: Compared with similar products in the market (such as Steam's traditional game recommendation system), the advantages of this game recommendation system mainly include the following three points:

- Support game-based recommendation, not just the traditional recommendation system filtering mechanism based on tags;

- Support content-based recommendation, recommend games based on the relevance of user description content and tags;

- Support user-based recommendation to solve the pain point of opaque information caused by new users' lack of understanding of game tags;

Innovation:

- Build embedding vector based on graph neural network learned from 8000+ game data to support game-based recommendation;

- Innovate the recommendation mechanism of Internet games with traditional forms, and transform the product form;

2.2. User Scenario Requirements Analysis

This recommendation system mainly divides scenarios and needs into the following two categories according to users:

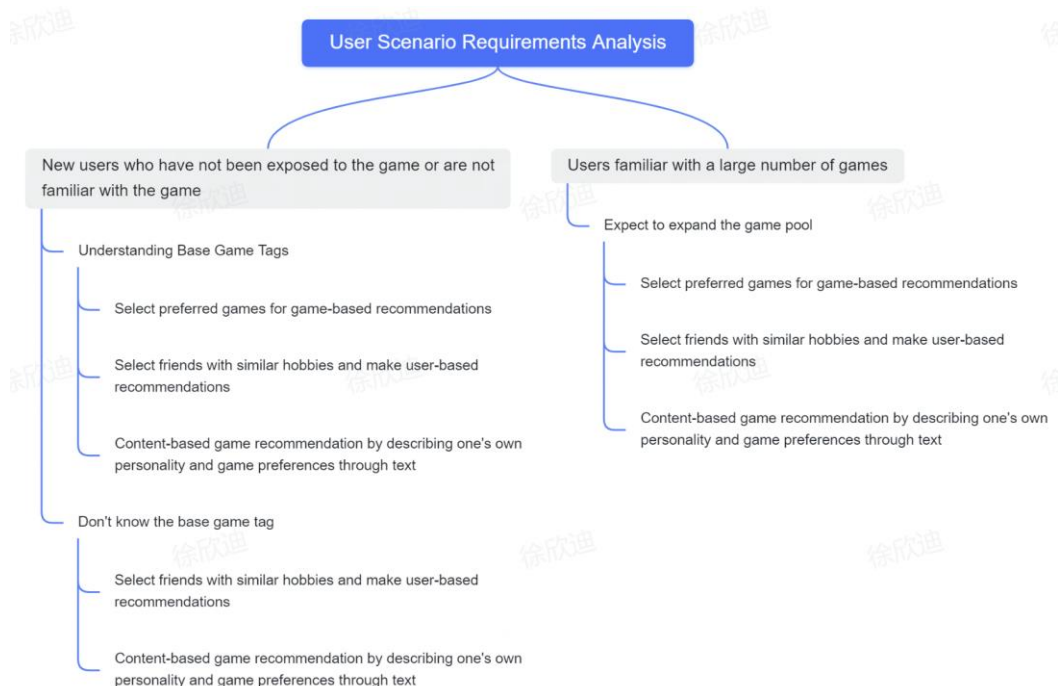


Figure 2.1: User Scenario Requirements Analysis

2.3. Practical Problem Solved

- Help users efficiently obtain game information that meets their preferences and avoid invalid recommendations;

- Provide users with multi-dimensional game information data to help users make game screening and decision-making;

- The Steam game community is highly targeted, helping users achieve extensive and

effective user recommendations;

- For all kinds of users (new and old users), provide personalized gamification recommendations to solve the user's "game shortage" needs;
- Wide range of usage scenarios. Web services cover the needs of users in multi-dimensional scenarios. For details, see the user scenario requirement analysis diagram;
- Get close to real Steam platform users and improve the user experience on the platform;
- Later version iterations have certain commercial value and space;

3. Data Loading and Data Preprocessing

3.1. Data Loading

3.1.1. Obtain data

In this section, we retrieve the data and save it as a JSON file.

Whilst the data was said to be in JSON format, when running it through a JSON linter, we noticed that it was not proper JSON due to the use of single quotes instead of double quotes.

To remedy this, we explored options such as replacing the single quotes with double quotes. However this led to issues where the name of a game included an apostrophe, e.g. Assassin's Creed.

3.1.2. User-items data

The dataset was downloaded from

https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data file Version 1: User and Item Data.

A workable solution to correct the json format was to read the data carefully using the `ast.literal_eval()` method and saving the outputs as .json files.

We now have a .json file that we can easily view as a Pandas DataFrame.

	user_id	items_count	steam_id	user_url	items
0	76561197970982479	277	76561197970982479	http://steamcommunity.com/profiles/76561197970...	[{"item_id": "10", "item_name": "Counter-Strik...
1	js41637	888	76561198035864385	http://steamcommunity.com/id/js41637	[{"item_id": "10", "item_name": "Counter-Strik...
2	evcentric	137	76561198007712555	http://steamcommunity.com/id/evcentric	[{"item_id": "1200", "item_name": "Red Orchest...
3	Riot-Punch	328	76561197963445855	http://steamcommunity.com/id/Riot-Punch	[{"item_id": "10", "item_name": "Counter-Strik...
4	doctr	541	76561198002099482	http://steamcommunity.com/id/doctr	[{"item_id": "300", "item_name": "Day of Defea...

Figure 3.1: .json file

3.1.3. Items detail data

This dataset was downloaded from

https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data file Version 2: item metadata

Similar to above, we will use `ast.literal_eval()` to read the data and then save as a .json file.

	publisher	genres	app_name	title	url	release_date	tags	discount_price	re
0	Kotoshiro	[Action, Casual, Indie, Simulation, Strategy]	Lost Summoner Kitty	Lost Summoner Kitty	http://store.steampowered.com/app/761140/Lost_...	2018-01-04	[Strategy, Action, Indie, Casual, Simulation]	4.49	http://steamcommunity.com/app/761140/r
1	Making Fun, Inc.	[Free to Play, Indie, RPG, Strategy]	Ironbound	Ironbound	http://store.steampowered.com/app/643980/Ironb...	2018-01-04	[Free to Play, Strategy, Indie, RPG, Card Game...]	NaN	http://steamcommunity.com/app/643980/r
2	Poolians.com	[Casual, Free to Play, Indie, Simulation, Sports]	Real Pool 3D - Poolians	Real Pool 3D - Poolians	http://store.steampowered.com/app/670290/Real_...	2017-07-24	[Free to Play, Simulation, Sports, Casual, Ind...]	NaN	http://steamcommunity.com/app/670290/r
3	彼岸领域	[Action, Adventure, Casual]	弹珠人 2222	弹珠人 2222	http://store.steampowered.com/app/767400/2222/	2017-12-07	[Action, Adventure, Casual]	0.83	http://steamcommunity.com/app/767400/r
4	NaN	NaN	Log Challenge	NaN	http://store.steampowered.com/app/773570/Log_C...	NaN	[Action, Indie, Casual, Sports]	1.79	http://steamcommunity.com/app/773570/r

Figure 3.2: .json file

3.2. Data Preprocessing

In this section, we will conduct initial preprocessing of the two JSON files we obtained in the previous section. We will aim to create relevant Pandas DataFrames and save files as csv for subsequent exploration and modelling. In particular, we will extract user ids and game ids to create a user-item interactions DataFrame, with each row being a particular user-item relationship (namely owned item).

3.2.1. Games data

We will first load the gamesdata file, which has a row for each game and various descriptive features as columns. We note that there are certain features which are lists, namely genres, tags and specs. These will be investigated further in the Data Exploration phase.

3.2.2. User items Data

We now load the user items data, which has users as rows and details regarding items owned as columns.

Each game is represented by a dictionary with keys the game's `item_id`, `item_name`, `playtime_forever` and `playtime_2weeks`. The dictionaries are then stored in a list.

We will look to extract the `item_ids` into a separate column. For now we will leave the playtime data but look to incorporate it later.

We will now generalize the above and create a column extracting the `item_id` from the dictionaries for each user.

4. System Design and models

Before we focus on our solutions for game recommendation, we should understand several different types of recommendation system. By definition, a recommendation system is a system that can identify and provide recommended content for users by using users' interest. It usually has three different types: Collaborative Filtering, Content-Based Filtering and Hybrid recommender system.

- Collaborative Filtering

Collaborative filtering is a method of making automatic predictions about the interests of a user by collecting preferences from many users. The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. But one major problem for collaborative filtering is “cold start”. Though collaborative filtering is a powerful recommendation system, it can't work without user history. As for those new users who don't have user history, the system can't give good recommendations.

- Content-Based Filtering

In a content-based recommendation system, keywords are used to describe the items. Besides, a user profile is built to state the type of item this user likes, which means the system try to recommend products which are similar to the ones that a user has liked in the past. The idea of content-based filtering is that if you like an item, you will also like a 'similar' item. For example, the game recommendation system will recommend games to users according to the games they used to play.

- Hybrid recommender system

Hybrid recommender system is a special type of recommender system that combines both content and collaborative filtering method. The combination of two methods is proved to be more effective in many cases. Besides using two methods separately and then combining the results, hybrid recommender system can also add content-based capabilities to a collaborative-based approach, which is proved to have better recommendation performance compared with pure method.

4.1. System Design

The system architecture diagram shown below illustrates the whole structure of the web application of Steam Games Recommendation System.

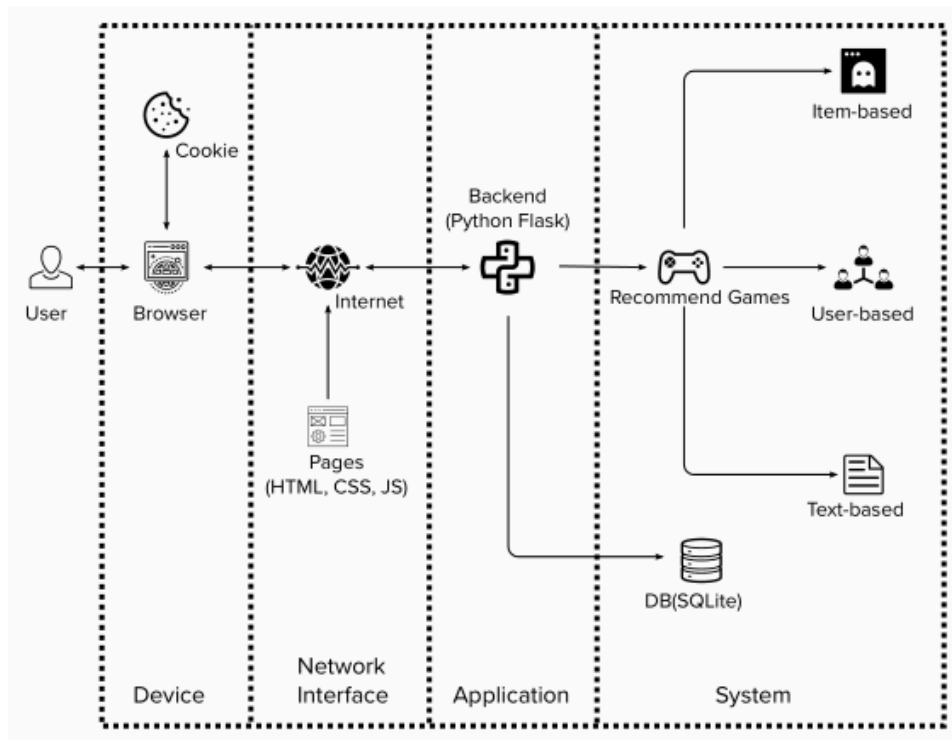


Figure 4.1: System Architecture

By comparing three different types of recommendation system, we decided to use hybrid recommender system as our recommendation model. Since the dataset we use contains historical data of users, and every game in the data set is labeled and classified in detail, both collaborative filtering and content-based filtering can work well. Through combining these two methods together will make our recommendation system more powerful and accurate.

We decided to subdivide our recommendation system into three recommendation methods according to specific use functions. As is shown in the system architecture above, we have three different recommendation models.

4.1.1. User-based recommendation

In this part, we will train a model that can recommend games for users who have played games and have games in their inventory. This is a typical collaborative filtering method. We can provide more accurate and targeted recommendations based on the user history information stored in the database.

4.1.2. Item-based recommendation

In this part, we will explore a different recommendation model. Maybe the users want to try some new types of games which are not similar to the games they have owned. Given this situation, we will build a recommendation system that based on categories and characteristics of games. Users can choose some new games, and then we will recommend similar games as what they chose. This is a type of content-based recommendation system.

4.1.3. Text-based recommendation

Our system also has a solution for new users who have no experience with games at all. Because these users don't have historical data and don't know much about games, it's hard to recommend games for them accurately with user-based recommendation and item-based recommendation system. Therefore, in this part, we need users to answer some questions to describe their personality, and then we use the word segmentation method to extract the keywords answered by users. Finally, we match the extracted keywords with the game tags in the database, and recommend some games that are in line with the user's personality and preferences.

4.2. Database Design

4.2.1. Table Structure

The database contains a total of four tables, including the game table, the user table, and two embedding tables. Two of the embedding tables store binary vectors and are meaningless for visualization. Therefore, the table structure is not shown below. The game table is described in the previous section. Multiple cvs files left join generated after data preprocessing

(1) games_dws table structure

we merged the games data and built a wide table of user data in the dws layer. The table fields are as follows:

```
-- gamerec.gamesnew2_dws definition

CREATE TABLE `gamesnew2_dws` (
  `Column1` int(11) DEFAULT NULL,
  `publisher` mediumtext,
  `genres` mediumtext,
  `app_name` mediumtext,
  `title` mediumtext,
  `url` mediumtext,
  `release_date` mediumtext,
  `tags` mediumtext,
  `discount_price` double DEFAULT NULL,
  `reviews_url` mediumtext,
  `specs` mediumtext,
  `price` mediumtext,
  `early_access` mediumtext,
  `id` double DEFAULT NULL,
  `developer` mediumtext,
  `sentiment` mediumtext,
  `metascore` mediumtext,
  `steam_appid` int(11) DEFAULT NULL,
  `detailed_description` mediumtext,
  `about_the_game` mediumtext,
  `short_description` mediumtext,
  `steam_appid1` int(11) DEFAULT NULL,
  `header_image` mediumtext,
  `screenshots` mediumtext,
  `background` mediumtext,
  `movies` mediumtext
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figure 4.2: games_dws table structure

4.3. Model

In this chapter, we will talk in more detail about how to train the model and use the model to make recommendations. We mainly use LightFM Hybrid model and Spacy to fulfil our recommendation methods.

4.3.1. Recommendation model built by LightFM

LightFM is a hybrid matrix factorisation model representing users and items as linear combinations of their content features' latent factors. The model outperforms both collaborative and content-based models in cold-start or sparse interaction data scenarios, and performs at least as well as a pure collaborative matrix factorisation model where interaction data is abundant.

The model learns embeddings vectors for users and items in a way that encodes user preferences over items. When multiplied together, these representations produce scores for every item for a given user; items scored highly are more likely to be interesting to the user. The user and item representations are expressed in terms of representations of their features: an embedding is estimated for every feature, and these features are then summed together to arrive at representations for users and items.

The latent representation of user u is given by the sum of its features' latent vectors:

$$q_u = \sum_{j \in f_u} j$$

The latent representation of item i is given by the sum of its features' latent vectors:

$$p_i = \sum_{j \in f_i} j$$

The model's prediction for user u and item i is then given by the dot product of user and item representations, adjusted by user and item feature biases:

$$r_{ui}^{\wedge} = f(q_u \cdot p_i + b_u + b_i)$$

Above is the general idea of the model, next we will train our own game recommendation model with the data we have prepared on user and game information. Fortunately, there is a library that makes easy to build a lightFM model. LightFM model is developed by Lyst. They also created a library for building lightFM model. We will use the library when building our own lightFM model.

(1) Create interaction matrix

Before we applying the lightFM algorithm to build our own recommendation model, we need to build an interaction matrix first. The interaction matrix should contain the relationship between each user and game, which will be calculated when training the model.

Given that we already had information about all our users and the games they owned in our database, we decided to construct a two-dimensional interaction matrix, in which the rows represent user information and the columns represent game information. So, the shape of interaction matrix should be 17543 x 8556, which means the interaction matrix contains the relationship between 17543 users and 8556 games. And in order to represent the relationship between users and games, we use whether a user owns a game to judge their relationship. In this way, the value stored in the interaction matrix will be either 0 or 1. Just as the table shown below:

Table 4.1: Interaction Matrix

id	10	20	30	40	50	60	70	80	130	220	...	526790	527340	527440	527510	527520	527810	527890	527900	528660	530720
uid																					
0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

(2) Selection and comparison of model parameters

In this part, we will use the interaction matrix to train our own recommendation model. During training, there are still many relevant parameters to consider. We split the interaction matrix into two parts, which represent training set and test set for evaluation purposes. We choose to have roughly 80% of our data as training and 20% as test.

Then we need to decide which loss function to use. LightFM offers us two choices:

- BPR: Bayesian Personalized Ranking pairwise loss. Maximizes the prediction difference between a positive example and a randomly chosen negative example. Useful when only positive interactions are present and optimizing ROC AUC is desired.
- WARP: Weighted Approximate-Rank Pairwise loss. Maximizes the rank of positive examples by repeatedly sampling negative examples until rank violating one is found. Useful when only positive interactions are present and optimizing the top of the recommendation list is desired.

Both loss functions are suitable for our recommendation system, so we decided to train these two models separately to see which one has better performance.

The results are shown in the tables below:

Table 4.2: Performance

Loss Function	Training Set	Test set	AUC (train)	AUC (test)
BPR	0.80	0.35	0.96	0.70
WARP	0.73	0.42	0.98	0.92

According to the results, we found that the precision score is low on test set, but this is expected due to the sheer volume of games available and the scarcity of interactions.

But the AUC has a high score when applying WARP as loss function. So, finally we decide to use WARP as our loss function to train our model.

Moreover, we also need to decide the “no_components” parameter when training. This parameter controls the number of “embeddings”, which represents the dimensionality of the features in latent space. We vary this number from 10 to 40 to see how this affects model performance. But we found this feature has little impact on overall model performance.

(3) Final model and test

Finally, we train our model with WARP loss function on the full interaction matrix and set the “no_components” parameter as 30. Then we got a lightFM model which can recommend games for different users, which is a user-based recommendation system we mentioned before. Moreover, we also got two embeddings which we will use later to build our item-based recommendation system.

Now let’s check how the model works. When we input the user’s id 2022 to our model, the model will return top 5 games which best meet the user's preferences. In this way, we have successfully implemented a user-based game recommendation system. As the results shown below:

```
Known Likes:
1- Emily is Away
2- No More Room in Hell
3- Age of Empires II HD
4- PlanetSide 2
5- Sanctum 2
6- Crysis 2 - Maximum Edition
7- Age of Empires® III: Complete Collection
8- Saints Row: The Third
9- Red Faction®: Armageddon™
10- Darksiders™
11- LIMBO
12- Medal of Honor™
13- Defence Alliance 2
14- Command & Conquer: Red Alert 3 - Uprising
15- Burnout Paradise: The Ultimate Box
16- Company of Heroes: Tales of Valor
17- Dead Space
18- Mirror's Edge™
19- Sid Meier's Civilization® V
20- Warhammer® 40,000: Dawn of War® - Game of the Year Edition
21- Company of Heroes - Legacy Edition
22- Garry's Mod
23- Killing Floor
24- Counter-Strike: Global Offensive
25- Left 4 Dead 2
26- Portal
27- Half-Life 2: Lost Coast
28- Half-Life 2: Deathmatch
29- Counter-Strike: Source
30- Half-Life 2

Recommended Items:
1- Terraria
2- Arma 2: Operation Arrowhead
3- Unturned
4- Portal 2
5- Just Cause 2
```

Figure 4.3: Model result

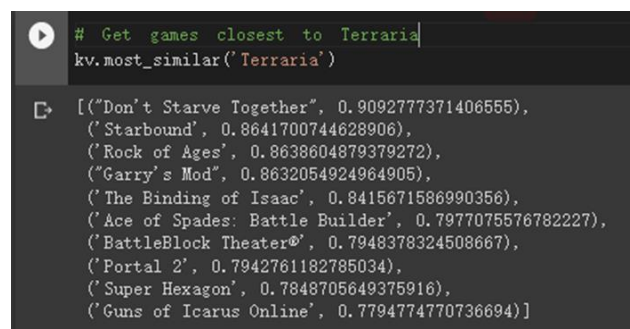
(4) Embeddings

In this part, we need to build an item-based system with the help of embeddings we got from the lightFM model. According to the definition of embeddings in lightFM model, embeddings are latent representations of users and items in a high-dimensional space, which are learned through stochastic gradient descent methods. In other words, the embeddings represent the characteristics of the games and users. After training, the model will offer two embeddings. One is called item-based embedding, and the other is user-based embedding. Targets with similar features will have a closer embedding vector.

As for the features defined by the embeddings, users with similar games are more similar to each other, and these games will have similar features. In this way, we found a way to judge the similarities between games, which is of great importance in item-based game recommendation. Based on the results of the previous model training, we can get an item-based embedding array which has the shape of 8556x30. Each row represents the feature of a game, which corresponds to 8,556 games. The embedding space can be interpreted as a distance metric and so we would expect similar games to have similar vectors.

In this way, we use cosine similarity to calculate the similarity between two games. Cosine similarity ranges between -1 and 1 based on the angle between the vectors. From this figure, cosine distance is defined as 1 minus the cosine similarity and therefore a value between 0 and 2. When a user wants to find games similar to the game he has chosen, it's convenient to calculate cosine similarity with other games in the database.

As shown below, when we want to find games similar to “Terraria”, we just input the name of the game, and we can easily get the games similar, such as “Don't Starve Together”. In this way, we can easily build an item-based recommendation system.



```
# Get games closest to Terraria
kv.most_similar('Terraria')

[("Don't Starve Together", 0.9092777371406555),
 ('Starbound', 0.8641700744628906),
 ('Rock of Ages', 0.8638604879379272),
 ("Garry's Mod", 0.8632054924964905),
 ('The Binding of Isaac', 0.8415671586990356),
 ('Ace of Spades: Battle Builder', 0.7977075576782227),
 ('BattleBlock Theater', 0.7948378324508667),
 ('Portal 2', 0.7942761182785034),
 ('Super Hexagon', 0.7848705649375916),
 ('Guns of Icarus Online', 0.7794774770736694)]
```

Figure 4.4: Model result

4.3.2. NLP model built by spaCy

We have implemented user-based recommendation and item-based recommendation system in the previous section. Next, we will try to solve the problem of making recommendations to people who have never played games before. To solve this problem, we want users to describe themselves about their personalities and preferences. Then the system can infer from their description what type of game they are suitable for, thereby avoiding the cold start problem of missing user history data.

This is an NLP problem, and we want divide this problem into two steps. The first step is to understand the key words according to the input of users. The second step is to match the key words with the tags of games, and then recommend specific types of games.

We decide to use spaCy to fulfil these two steps. SpaCy is an open-source software library for advanced natural language processing. Unlike NLTK, which is widely used for teaching and research, spaCy focuses on providing software for production usage. Using Thinc as its backend, spaCy features convolutional neural network models for part-of-speech tagging, dependency parsing, text categorization and named entity recognition (NER).

(1) Process the input sentences

In this part, we need to use spaCy's pre-trained model called 'en_core_web_lg'. The spaCy offers three types of pre-trained model for NLP, which are different in accuracy. We chose the largest model which has the best performance in NLP.

To get the key words according to the sentences input by users, we need to segmenting text into words, punctuations marks etc. The model we used has the accuracy of 99.93% in segmenting text. Moreover, it labels each word separated from the sentence according to its part of speech in the sentence, such as "ADJ", "ADV" and so on.

Because our question asks users about their interests and preferences, we can select adjectives of the answer as our keywords. For example, when the answer is "I am curious. And I like scary and exciting things.", we can choose "curious, scary, exciting" as our keywords to represent new user's preference. It will easy to implement with the large pre-trained model offered by spaCy.

(2) Calculate the similarity

After the system get the keywords that describe the user's preferences, we need to associate the keywords with games in order to make appropriate recommendation. According to the database we built, each game has multiple labels that describe its category. So we need to calculate the similarity between keywords and tags.

Usually in NLP, we can use vectors to represent words, and then through extensive text training, we can finally use a unique vector to represent each word. Finally, words with similar semantics will be more similar in vector representation. Based on this theory, we still use spaCy's pre-trained model 'en_core_web_lg'. Because it has 514157 unique vectors (300 dimensions), which contains all 199 tags of our games.

In this way, we can quickly find the closest tag to the keyword. For example, as the result shown below, if user input "I like horrible games", the keyword will be "horrible". After computation of the model, most similar tag to the "horrible" will be "horror", which is reasonable.

```

D:\Anaconda\Anaconda\envs\pytorch\python.exe "F:/PycharmProject/Game_Recomond/Get the Tag.py"
Please input your answer:
I like horrible games
Target tags are: ['Horror']

```

Figure 4.5: Model result

4.4. Backend

4.4.1. Overview

The responsibility of backend development is to develop data access services. The backend in our project is mainly responsible for receiving requests sent by the frontend(user), and then, according to the parameters in requests, operating on data, such as searching relevant games by name, genre, or id, and searching recommended games by certain chosen game by user. So as to realize the frontend's response to user's requests.

In this part, we use Python as programming language, because it is widely used in the implementation of AI algorithms, which are the core of our recommendation system. In addition, Python has great portability since Python is open-sourced, and it has been ported to most platforms. If the program avoids the laziness of the system, then it can be used on any platform.

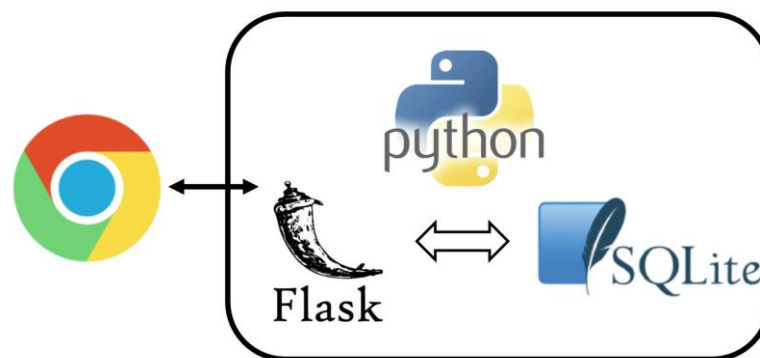


Figure 4.6: Backend structure

Backend Framework

In web application development, in order to improve the development efficiency, we need to use Python framework. The framework is a semi-finished product. It has encapsulated basic codes and provides corresponding APIs. When developers use the framework, they can directly call the encapsulated APIs to save a lot of code writing, thereby improving work efficiency and development speed.

The most popular three Python frameworks are Django, Flask and Tornado. Here we choose Flask, a lightweight web application framework written in Python, which is suitable for our project. Flask does not require particular tools or libraries, but it can

support extensions than can add application features, which makes it able to scale up to complex applications. Therefore, such features of Flask can help our game recommendation system set up quickly as well as keep its potential to extend to a large application.

DB

SQLite is a lightweight database, a powerful embedded relational database management system, using a compact C library. It is fast, efficient, and scalable, providing support for a large subset of SQL92, multiple tables and indexes, transactions, views, triggers, and a large number of client interfaces and drivers. The SQLite3 core engine does not depend on third-party software, no installation, no configuration, no need to start, shut down or configure a database instance. When the system crashes, there is no need to do any recovery operations, and it will automatically recover the next time the database is used. In addition, SQLite makes data query simpler: data can be loaded into SQLite in memory database, and data can be extracted at any time, or in the way you want.

Since SQLite doesn't need complex configuration before using it, no installation, no configuration, and no need to start, shut down or configure a database instance, we choose SQLite as a database. Moreover, if we need a more powerful DB software, SQLite is easy to be replaced.

4.4.2. APIs

API(Application Program Interface) is a set of definitions, programs and protocols that communicate with each other between computer software through an API interface. In backend, APIs are designed to provide services for the frontend. Through calling APIs, the frontend can send requests and parameters, so then achieve complex functions after obtaining the data returned by the backend.

In our game recommendation program, we designed five main APIs, two search modes and three recommendation modes, whose functions are searching games by a similar name, getting user's information by searching user's name, recommending games by certain game, recommending games by certain user, and recommending games by descriptions entered by the user.

“/searchGameByName”

This API is used for search box, which is the first mode of searching. When user wants to find a game but could not remember the whole name, he can enter an uncompleted name of this game in the search box. And then, information of all the relevant games would be extracted from database and then shown on the screen. The realization principle of this method is based on SQL statement, extracting all games whose names contain user input information, and return them to the frontend.

"/searchUserByName"

This function is used when user wants to find information of his friend, including his

favorite games. Similar to the previous API, the backend would return information of all the relevant users according to the input based on SQL statement. As a result, our system could realize the function of finding friends.

"/getSimilarGamesByItem"

This is the first recommendation mode in our system, which is an item-based method. In general, this function is to recommend similar games based on one game chosen by the user. In this function, the trained item-based model introduced above would be applied.

Data preparation:

- an item embedding matrix, representing features of each items(games), which is calculated by the trained model and dataset used for training the model;
- an item id, which is the game id chosen by the user;
- an item dictionary, which is a one-to-one correspondence table between game ids and names

Implementation process:

Define a function, whose inputs are item embedding matrix, item id, item dictionary, and the number of recommended items, whose outputs are recommended games. In this function, we need to calculate cosine distance between the chosen game and others, and then compare the values, the smaller the value, the more similar the two games. After ranking, information of top 100 similar games would be picked and returned to the frontend.

"/getSimilarGamesByUser"

This is the second recommendation mode in our system, which is a user-based method. When calling this API, our system could recommend games based on the user's friend's favorite games. In this function, the trained user-based model introduced above would be applied.

Data preparation:

- a trained model, which is based on user-item method using LightFM;
- user-item interactions, which represent games played by each user;
- a user id, which is the id of the chosen friend of the user;
- a user dictionary, which is a one-to-one correspondence table between user ids and interaction indexes, generated by user-item interactions.

Implementation process:

Define a function, whose inputs are the trained model, user-item interactions, user id, user dictionary, whose outputs are names of recommended games. After getting the names, another function would extract other needed information of recommended games according to the generated names. As a result, recommended games would be

returned to the screen and the user.

"/getTagsFromText"

This is the third recommendation mode in our system, which is a text-based method. User needs to enter a description of his ideal game, and then, our system would analyze and extract the keywords in this description. Finally, return related types of games as recommendation.

Data preparation: a text, which is the description of user's ideal game entered by himself.

Implementation process: utilize NLP(Natural Language Processing) algorithm to train a model to recognize and extract the keywords of that description, so then obtain several genres the user may like.

4.5. Frontend

4.5.1. Overview

For this project, we have used a separate front and back end for development. For front end development, we used React.js as a framework for development, for its wide applicability, good compatibility and developed community. We also use Javascript as the development language and Ant Design as UI and components library.

As this project did not involve overly complex web interaction, we decided to use only one page to display the content. The page layout is as follows.

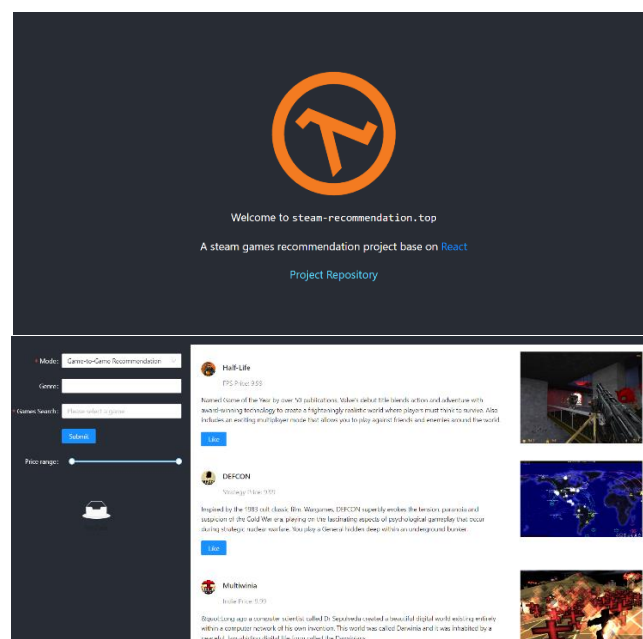


Figure 4.7: Web page

Framework

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets developers compose complex UIs from small and isolated pieces of

code called “components”. As a front-end MVVM framework, React is very simple and easy to use, and with its internal implementation of state management tools, it significantly reduces the mental burden on developers when managing variables.

However, React is only a framework and does not contain implementations of the various UI components. Therefore, we chose Ant Design as the UI component library because it is aesthetically pleasing and easy to use, and is also based on the React implementation. For engineering, we used Webpack as a packaging tool to package the front-end resource files. In addition, we also used some additional tools for development such as axios and react-helmet.

4.5.2. Deployment

This project is currently deployed on a cloud server, so you can access it via this url:<http://steam-recommendation.top/>

5. Conclusion

5.1. Summary of Achievements

Our team has successfully constructed a Steam Games Recommendation web application, and deployed it online in this project. Our system provides three recommendation modes for people with different needs. For those who often play games, our system could recommend games similar to his favorites. For those who don't have any idea of which type of games they like, our system could provide their friends' favorites for consideration. For those who have ideal games, they can make a description of their ideal games, and then, the system would recommend games that meet the requirements. To sum up, the project has achieved its objectives and delivered a minimum viable product.

5.2. Future Improvements

Despite a successful implementation for a minimum viable product, there are areas for future improvements.

- The user-item data we use is very limited. Due to a lack of computing resources as well as considerations of how fast it would run after deployment, we only used data from over 20,000 users and ended up only being able to extract information from around 8,000 games, compared to the 20,000+ games in the steam game library. This means that both our user and game libraries are incomplete, which can lead to bias when making recommendations.
- The user data we use is all from Australia, which means that when recommending games the system will default to you being an Australian and will only push games that Australians like. We will be supplementing the user and game data later to compensate for any bias caused by insufficient and not extensive data.

- Due to lack of server performance and inexperience in development, we chose to save part of the matrix as a csv file and read the file on each request to get the information. This resulted in a significant drain on server resources and excessively long response times (1-2 minutes) and also made it impossible for the server to achieve high concurrency. We will then put this part of the matrix into the database to ensure server performance.

6. Appendix

6.1. Proposal

6.1.1. Problem

For a long time, as the Steam platform is the world's largest comprehensive digital game software distribution platform, it is difficult for players to filter out their favorite games due to the complicated and complicated game labels and game versions, as well as the fuzzy and inconsistent perception of individual games. , especially for new users, because they do not understand the game content of the game system or even the game attributes reflected by the game tags, it is difficult to use traditional tags to filter out the games they try for the first time. In addition, the current traditional game recommendation system often recommends popular games or filters according to game tags, and does not involve a comprehensive recommendation algorithm.

6.1.2. Solution

- Game-based recommendations;
- User-based recommendations;
- Recommendation based on content (description);

For details of the recommended algorithm, please refer to Sections 4.1.1-4.1.3

6.1.3. Objective

We expect to complete the market research, demand analysis, function design, algorithm development and front-end and back-end implementation of the MVP version of the game recommendation system in one month, and based on our three types of recommendation functions, we can empower Steam platform users to efficiently find their favorites new game.

6.2. System Functionalities Mapping

- Machine Reasoning

In this course, we learned how to infer from existing knowledge and data to obtain implicit relationships, which is of great importance in building our recommendation system. When building item-based and user-based systems, we trained a

recommendation model with existing relationships between users and games. Then we can infer the similarity between users and similarity between games through the model, which is a process of machine reasoning.

- Reasoning Systems

We learned how to build a reasoning system on this course. Our game recommendation system is a kind of reasoning system. So we use the methods we learnt from the courses to build three different recommendation models and make efforts to improve accuracy.

- Cognitive Systems

In this course, we learned many methods to recognize the data we obtained from the real world like sound, image, and text. And the methods of natural language processing helped us build the text-based recommendation system. This system needs to process the sentences input by users and extract key words in the sentence. Moreover, we also need to calculate the similarity between different words. All of these problems can be solved using the text processing methods learned in class.

6.3. Installation and User Guide

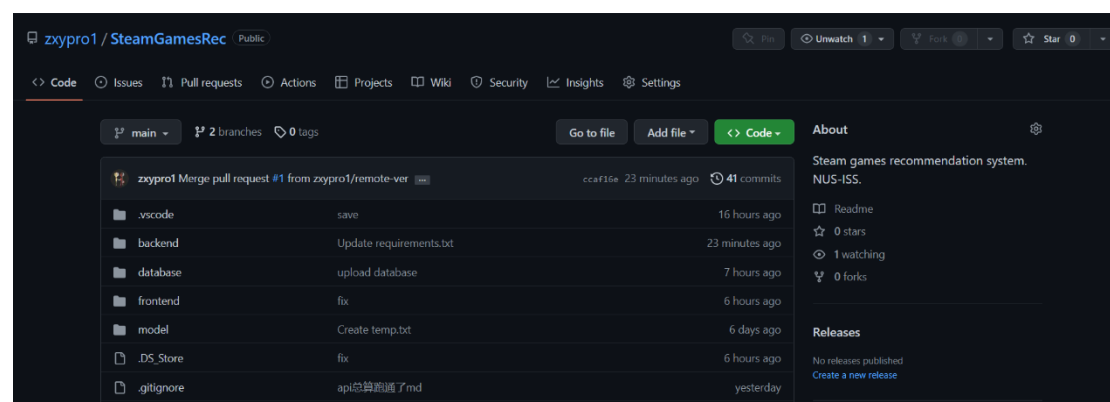
Local Installation Guide

Step1. Install Python

<https://www.python.org>

Step2. Clone our project from github

<https://github.com/zxypro1/SteamGamesRec>



Step3. Install environment via requirement.txt

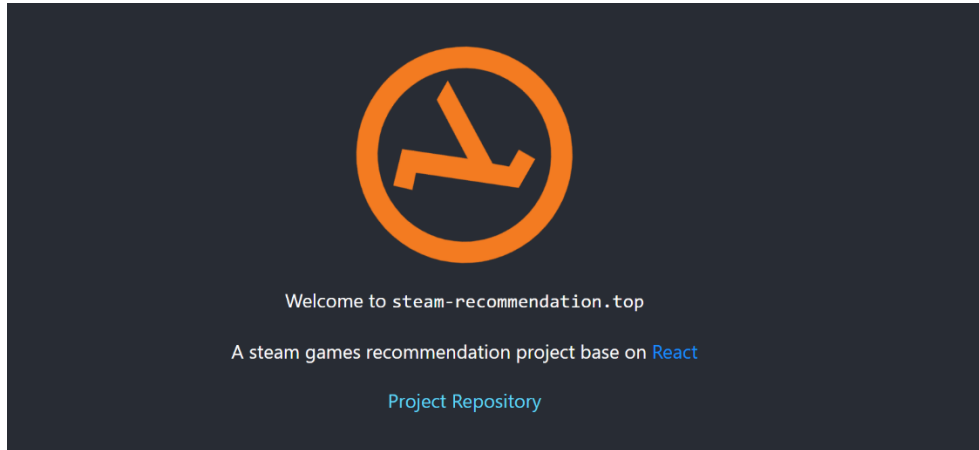
```
1 cd backend
2 pip install -r requirements.txt
```

Step4. Run the project

```
1 python __init__.py
```

Make sure you run the file in backend directory.

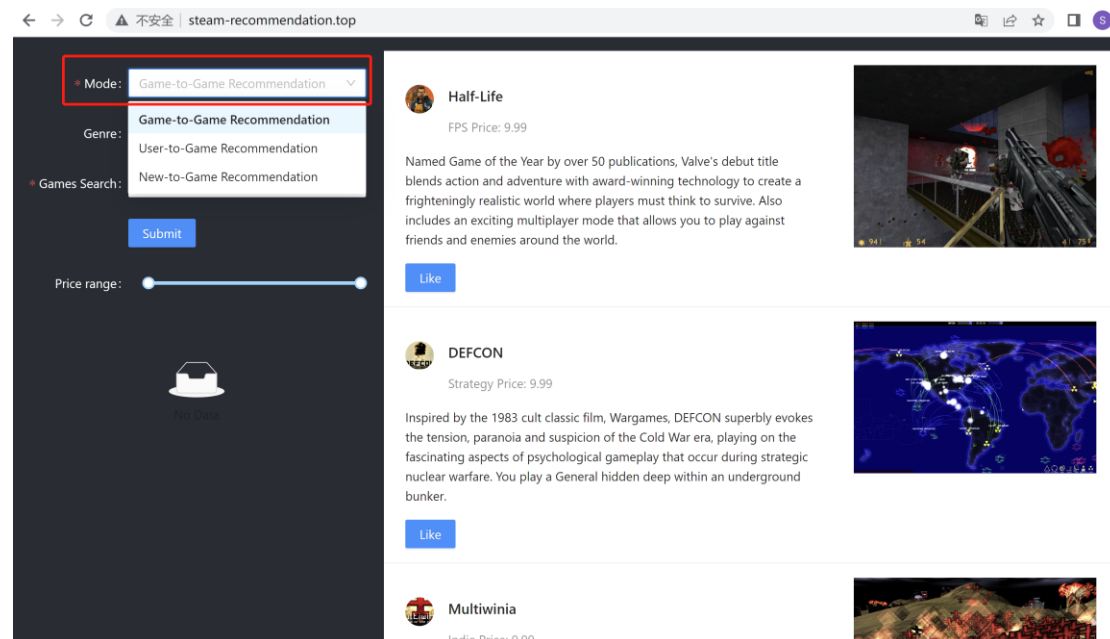
Open the url: <http://127.0.0.1:8080/> and you will see the website.



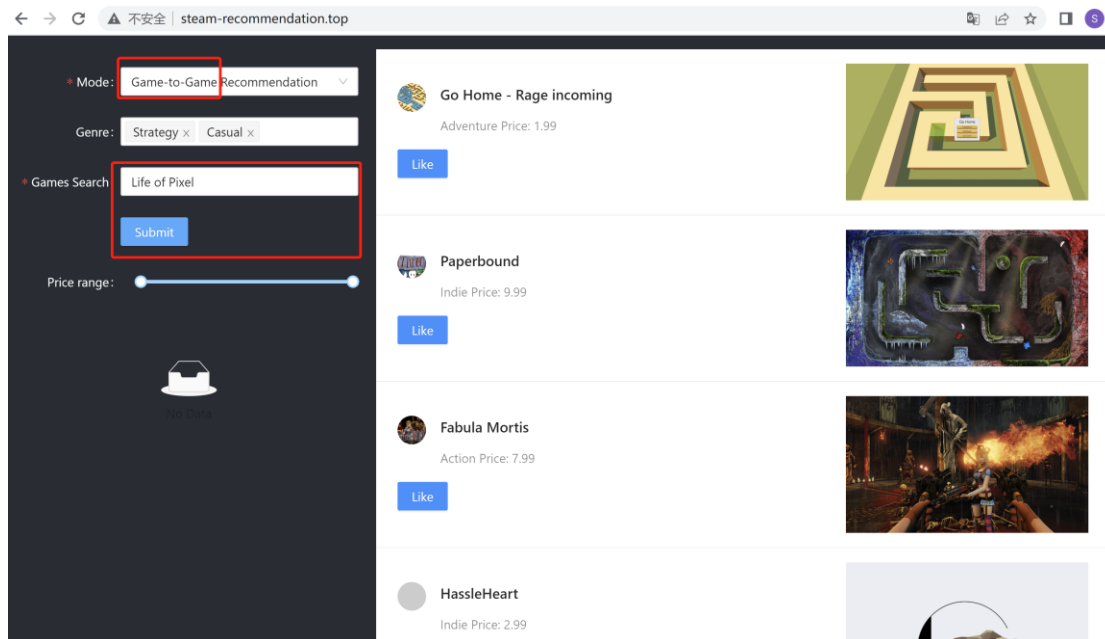
User Guide

Please click this link <http://steam-recommendation.top/>

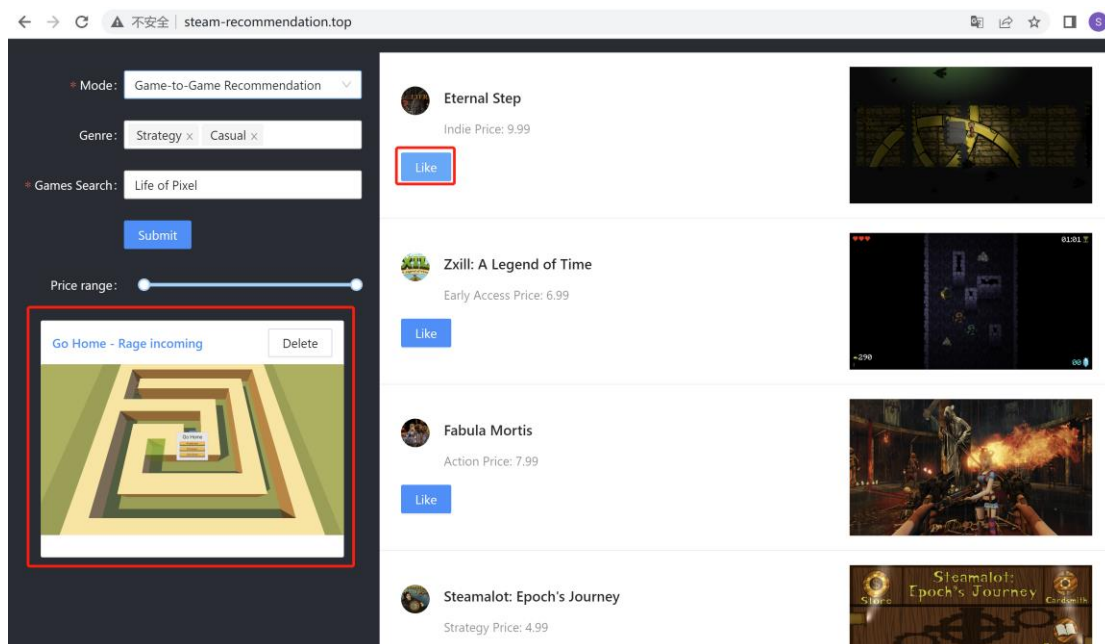
You can click the context in the red block to choose the mode of recommendation



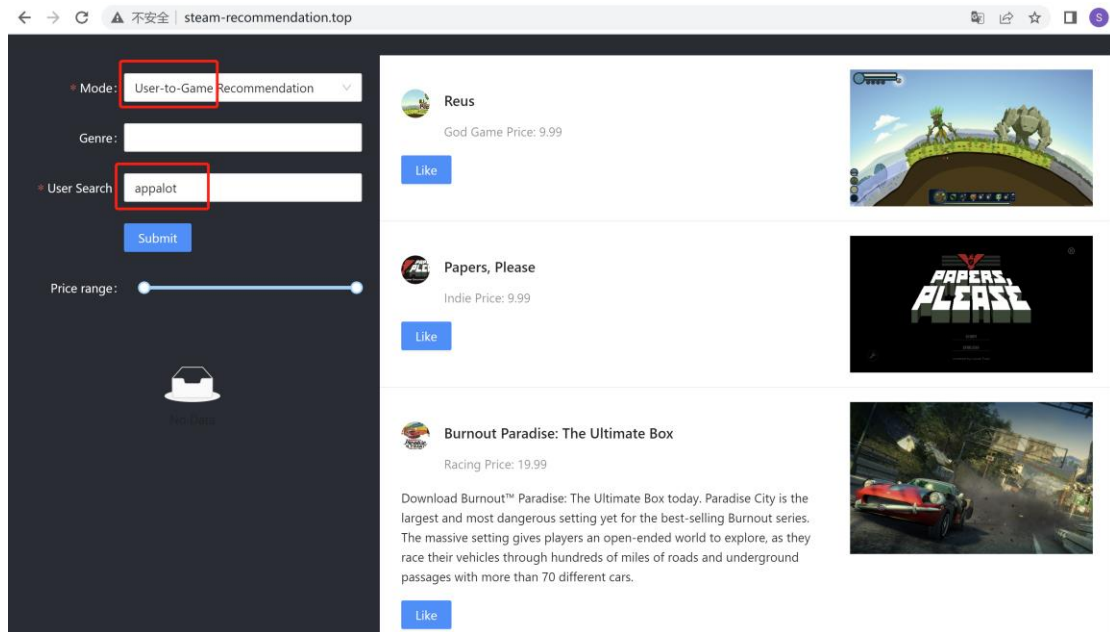
In game-to-game mode, you can input your interested game's name, and then, the system would recommend similar games, shown on the right.



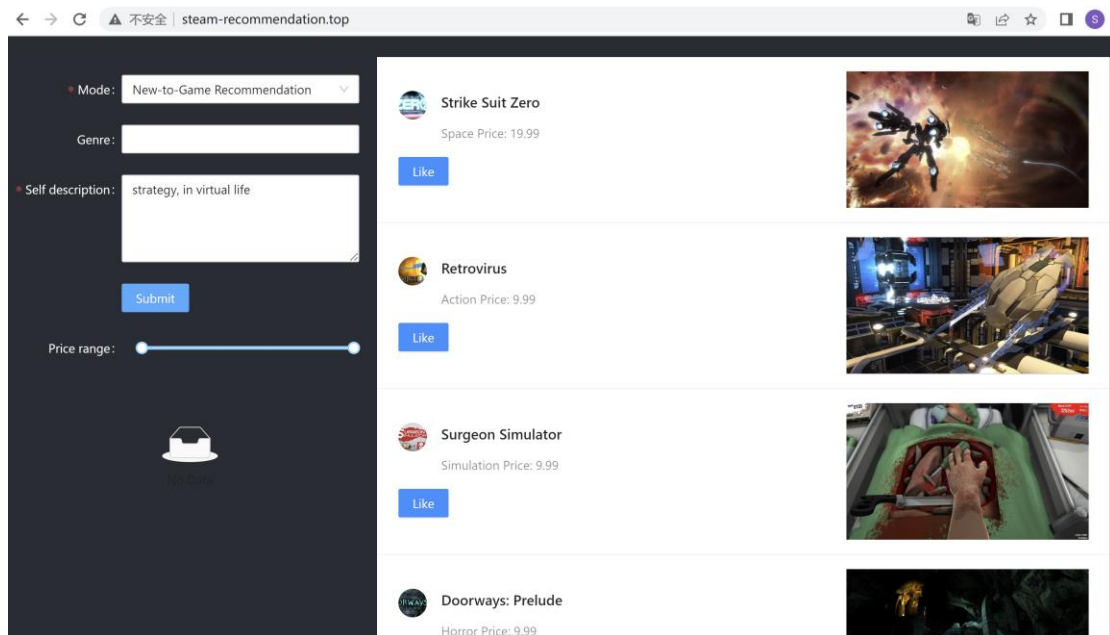
If you click “like”, this game would be shown on the left, and then, recommended games would still show on the right.



In user-to-game mode, you can input your friend’s name, and then, the games he played would show on the right. You can click like for further recommendation.



In new-to-game mode, you can input a description of your ideal game, then, the recommended games will shown on the right.



6.4. Individual Reports

Tian Yuyang

1. Your personal contribution to the project.

- Data Loading and Data Preprocessing;
- Database Structure Design and Database Development;
- Participate in product concept proposal and functional design;
- Responsible for the first chapter to the third chapter of the project report and the fourth chapter database design part;
- Make a video of the business value of this game recommendation system;

2. What learnt is most useful for you.

Through this project, I implemented an MVP version recommendation system from 0-1, which allowed me to have a transition from theory to practice for intelligent reasoning systems. And I was in charge of data loading and preprocessing, and I also learned more complex recommendation algorithms for NLP and graph neural networks. At the same time, I also participated in the database development part of the Web service, which gave me a deep grasp of the database structure of the engineering project, and was able to complete the development of this part by myself.

3. How you can apply this in other situations or workplaces.

- I deeply understand and master the ability of data loading and data preprocessing. I am familiar with various methods of data processing in numpy and pandas libraries. I have participated in the flow of the entire data process. The processing of data can be applied to other projects or work.
- I have a deep understanding and mastery of database development and design capabilities, familiar with various MySQL statements and Dbeaver for visual development, and can be independently responsible for database development in other projects or work.
- Familiar with the algorithm design of intelligent reasoning system, and can transfer related algorithm capabilities to other projects or work scenarios.
- Mastered animation production based on Animaker, and at the same time familiar with the writing method of commercial project documents.

Peng Shaoze

1. Your personal contribution to the project.

- Design recommendation system architecture: Hybrid recommender system.

- Choose algorithm for recommendation.
- Train and optimize the lightFM recommendation model to build a user-based recommendation system.
- Data processing and exploration.
- Use embedding to build an item-based recommendation system.
- Use spaCy to build a text-based recommendation system.
- Participate in the overall design of the project.
- Responsible for chapter 4.1 and 4.3 of the project report.

2.What learnt is most useful for you.

In this project, I learned how to build a recommendation system from scratch, especially the algorithmic part of the recommendation system. I learned how to train a recommendation model with lightFM, and how to recommend the most suitable products for users according to the relationship between users and items. Although the process of training the model is complex and difficult, I learned which parameters are more influential in building a recommendation system through repeated adjustment and optimization of the model. Moreover, I also verified what I had learned in class in practice and realized the significance of these algorithms in practical application. Like when dealing with recommendations for new users, using spaCy for natural language processing is a great way to analyze user input and make recommendations based on user input. To sum up, through this project, I learned how to combine the AI technology I learned with real life, and how to apply AI in product development.

3.How you can apply this in other situations or workplaces.

- In this project, I learned how to design and implement a game recommendation system. However, what I can do is not only game recommendation, I can apply this recommendation system to a variety of other scenarios, like food recommendation, music recommendation, book recommendation and so on. Although the recommended goals are different, the system is essentially the same.
- In addition, I used a lot of natural language processing methods when building the text-based recommendation system. These methods are very effective in processing language semantic analysis. If I need to build a chatbot, or extract keywords from an article, these methods are of great importance.
- When designing the recommendation system, I did lots of data analysis and processing by Pandas library. A good command and use of Pandas library will be of great help to me when I need to deal with data in the future.

Xu Xindi

1.Your personal contribution to the project.

- Construct this system's backend with python, flask and SQLite
- Design this recommendation system architecture
- Design system functions
- Analyze and choose suitable algorithm for recommendation
- Make technical presentation video
- Write project report and user guide

2.What learnt is most useful for you.

- Learn to use Python framework Flask and SQLite to construct a web application's backend
- Learn to design recommendation system architecture
- Know recommendation algorithms and how to train models

3.How you can apply this in other situations or workplaces.

- Through this project, I learnt how to build a complete web application, and known how to write backend, which is needed in any industries. Therefore, this skill could benefit me in subsequent other projects and help me find jobs in many companies.
- During this project, I was exposed to many machine learning algorithms, and had chance to train the model, which can help me solve similar problems in practical applications.
- Familiar with the database design and learnt how to develop database, which are necessary in many projects.

Zhang Xingyu

1.Your personal contribution to the project.

- Designed the overall system architecture and the way the front-end, back-end and algorithms are implemented.
- Determine the division of labour and decide on the technology stack, framework and language to be used for the system.
- Acquisition of data, data pre-processing and data mining.
- Independent front-end development and back-end development, using React.js, Flask, Ant Design, pandas, etc.
- Cloud server build and environment configuration. Deployment of project on the

cloud server.

- Responsible for the content of chapters 4.5, 5.2 and the installation guide.

2.What learnt is most useful for you.

Through this project, I learnt how to design a recommendation system from scratch (including website, interface, algorithms and data) and successfully implemented a complete system, deepening my understanding of the system construction and machine learning algorithm knowledge learnt in class. In algorithm building, I learned about NLP, recommendation systems and how to combine these two fields. In front and back-end development, I learned various data structures, system design methods and best practices through extensive coding. More importantly, I mastered the overall process and way of building a web application and my engineering skills have grown tremendously.

3.How you can apply this in other situations or workplaces.

- During this project, I learnt a great deal about the field of machine learning algorithms, including data acquisition, data cleaning, data pre-processing, model building and more, as well as how to use the relevant libraries in python to implement them. This knowledge will undoubtedly benefit me in my subsequent work on algorithms.
- In addition to the knowledge of algorithms, I also learned a great deal about engineering during this project. In a time when models are difficult to implement, having sufficient engineering experience will certainly come in handy in the workplace.
- Through extensive coding and documentation, I have also gained knowledge of code standardisation. Writing code in a more standardised way will come in handy in the future.