Movie Recommendation System

Milestone 2 - Group 5

Shiyan Cai 011415651
Wenhao Tan 010428158
Yifan Liu 011410984

Introduction

The dataset we selected was “The Movies Dataset” from kaggle. It is a huge dataset containing metadata and ratings for about 45,000 movies from 270,000 users. It has been used by quite a few people to build movie recommendation systems. We went through the posted kernels and concluded three popular approaches: simple popularity based, user-based collaborative filtering, and movie-based collaborative filtering. Popularity based system is the simplest approach which simply recommends movies based on the rank of movie popularities. User-based collaborative filtering uses user rating to find similar users and uses rating from those users to predict ratings. Movie-based collaborative filtering uses user ratings to find movies with similar ratings.

From all these posted kernels, we noticed the absence of metadata-based recommendation systems. The reason might be the difficulty of comparing metadata and the varying size of metadata. Our first goal of this project is to build a movie recommendation system based on as much meta information as possible.

In addition to the recommendation approach, we also noticed that none of the posted kernels uses the full ratings dataset but the including smaller dataset obviously due to the huge size of the ratings dataset. Our second goal of the project is to build a rating-based recommendation system that uses the entire or at least half of the huge dataset. The method we used here is TF-IDF method based on user prefered tag and movies tag.[1]

Compared with the original method, we also introduce a time variance weight to the system. In many projects, developers do not consider time variance in their analysis, they believe that users may make their interests unchanged. But in the real world, most people’s taste may change over time. Newer ratings should have higher importance than older ones.

Reference:

[1] Szomszor, M., Cattuto, C., Alani, H., O’Hara, K., Baldassarri, A., Loreto, V., & Servedio, V. D. (2007). Folksonomies, the semantic web, and movie recommendation.

Methods

Metadata-based recommendation systems

The metadata we selected to build our recommendation system on are: genres, cast, director, and collection. Our first approach was to build a vector for all possible genres with 1 meaning present and 0 meaning absence. With a vector representing the grenes, it is possible to find the similarity between any two movies. After data prepossessing, we found out that there are 32 unique genres in the dataset. It means each movie would require a vector of size 32 to store the genre information. Even though it is doable, the computation of similarity matrix for input size 45,000x32 still takes a long time. Besides, this method is impossible to be applied on cast information and there is no guarantee that whatever measure we found for other metadata would be in the same scale. While researching how to compare text, we came across the count vectorizer from sklearn library. It could convert a collection of text into a matrix to token counts. It compares two texts by counting the frequency of unique words inside the text and output the count vector. In theory, we could compare the metadata of two movies by extracting the information into strings and use this count vectorizer to convert the strings into numeric vectors and find the similarity. This could work pretty well for genres alone due the limited size of unique words for genres. However, as we add more meta information into the string, the string would become more complex and performance might drop significantly when the string becomes too complex. Therefore, we have to limit the size of this metadata string. Besides genres, we decided to use only the cast information for top 3 characters since main characters usually make much more differences. In terms of director, it is usually as important as the cast. Therefore, we repeat the name of the director 3 times to give it the same weight as cast. The metadata strings are then formed by concatenating all these information. In terms of collection, we made a multiplier matrix to multiply the similarities between movies in the same collection by 1.2 and others by 1 to give movies in the same collection higher weight than those not. With this approach, the computation time is still quite long and the similarity matrix might be too large to store. The solution we came up with is to pre-calculate the top 10 movies for each movie and store that instead. Unlike ratings data, movie metadata is less likely to be updated and this computation could be repeated periodically if it was to be used in an actual application.

TF-IDF method based on user prefered tag and movies tag

First step of this approach is to analyze the user's interest. We retrieve the user's rating record from the database. Since we only care about the user's interest, we only check movies with rating higher than 4 (Maximum is 5).

Next, we will count tags' importance, Tag including genre, cast and director. Importance function is shown below:

$$Importance(t) = \Sigma_{m \in M} ((Rating(m) - 3) * TimeVariance)$$

M: all rated movie with tag t

Time Variance: $Min(1, e^{\frac{-day+14}{320}})$ (day: day before last rating made)

We assume user's interest on a certain tag may start to decrease after 2 weeks, and only remain one third after one year.

After getting user's preferred tags, we search movies' data to get all movies which contain at least 2 tags, and use the following equations to calculate similarity and predict rating.

Given a new (in the sense of unrated) movie m∗, we consider the set of keywords Km∗ and introduce a notion of "similarity" between Km∗ and a user's tag set Tu,r. We define such a measure of similarity as:

$$\sigma(u, m, r) = \sum_{\{(k, n_k) \in T_{u,r} \mid k \in K_m\}} \frac{n_k}{\log(N_k)}$$

We sum over all tags which Km∗ and the tag set Tu,r have in common, and we weight each keyword k proportionally to its importance nk in the tag set, and inversely proportional to the logarithm of its global frequency.

We subsequently define the weighted average rating as:

$$\bar{\sigma}(u, m) = \frac{1}{S(u, m)} \sum_{r \in R} r\, \sigma(u, m, r)$$

$$S(u, m) = \sum_{r \in R} \sigma(u, m, r)$$

Where S(u,m) is a normalization factor, r is movie rating. $\bar{\sigma}$ is an estimate of the user's rating.

If the movie's estimated rating is the same, we may sort these movies by the sum of their tag importance.

Work Distribution

Wenhao Tan: data preprocessing and metadata-based recommendation system

Shiyan Cai: data preprocessing and user-based collaborative filtering system, UI frontend

Yifan Liu: Database and Rest API

Current Progress

UI Frontend:

Frontend is already finished, by using React. It contains two pages, one is the main page, another is the movie detail page.

In the main page, it contains the search part, recommendation part, and rating history part. Figure shows below, is the page with the test back end. All data used in following pictures are not real data from the database, only for the test purpose. For some unknown reason, poster paths from TMDB are no longer available, this is why many pictures cannot be shown here.

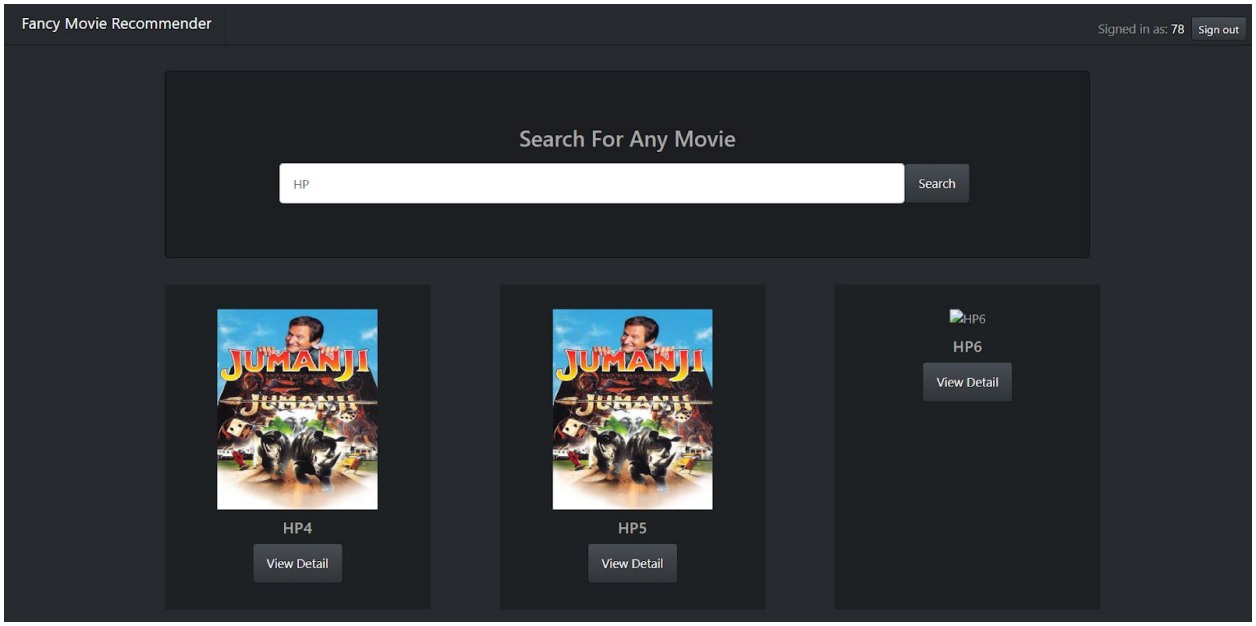

Figure 4.1 Search Part

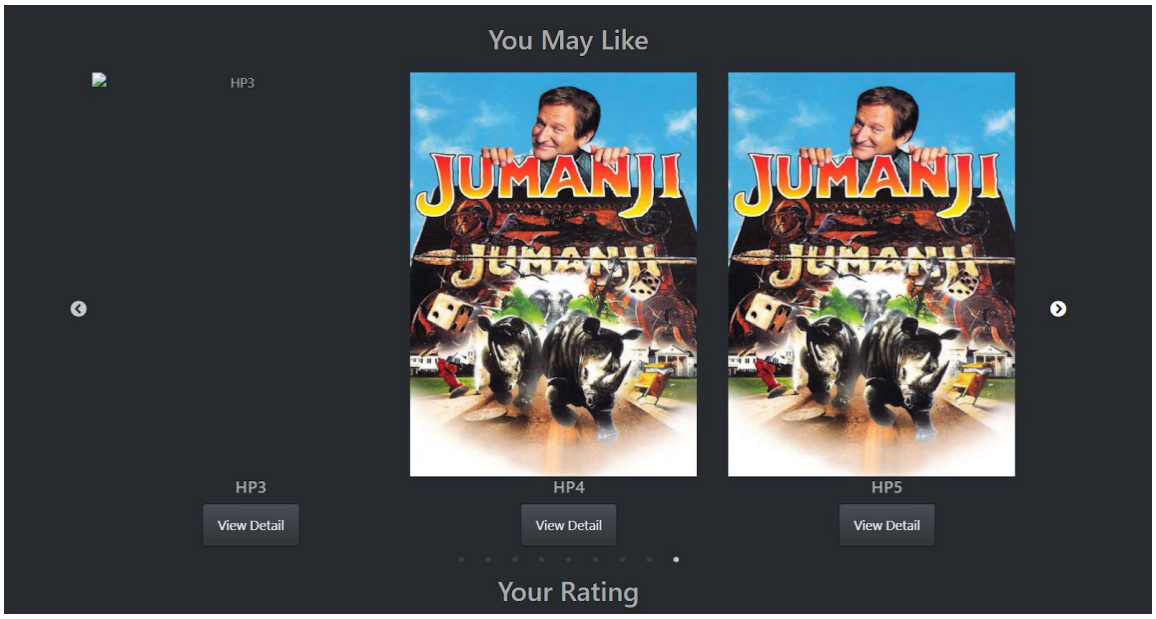

Figure 4.3 Recommend Part

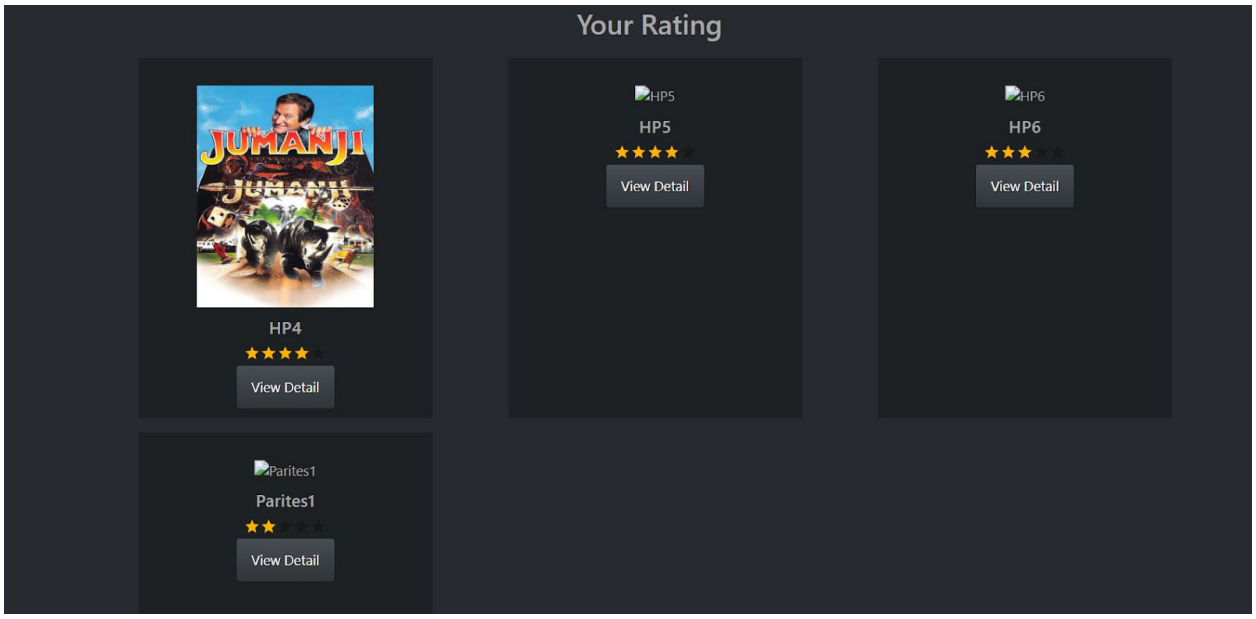

Figure 4.3 Rating History Part

In the main page, it also contains a nav bar which allows users to sign in and sign out with their id.

Another page is the movie detail page, this page is to show movies detail including title, genre, release date, IMDb rating, director, and actors. It contains a common tag set between movie tag and user preferred tag. It also allows users to create or update the rating on the movie.

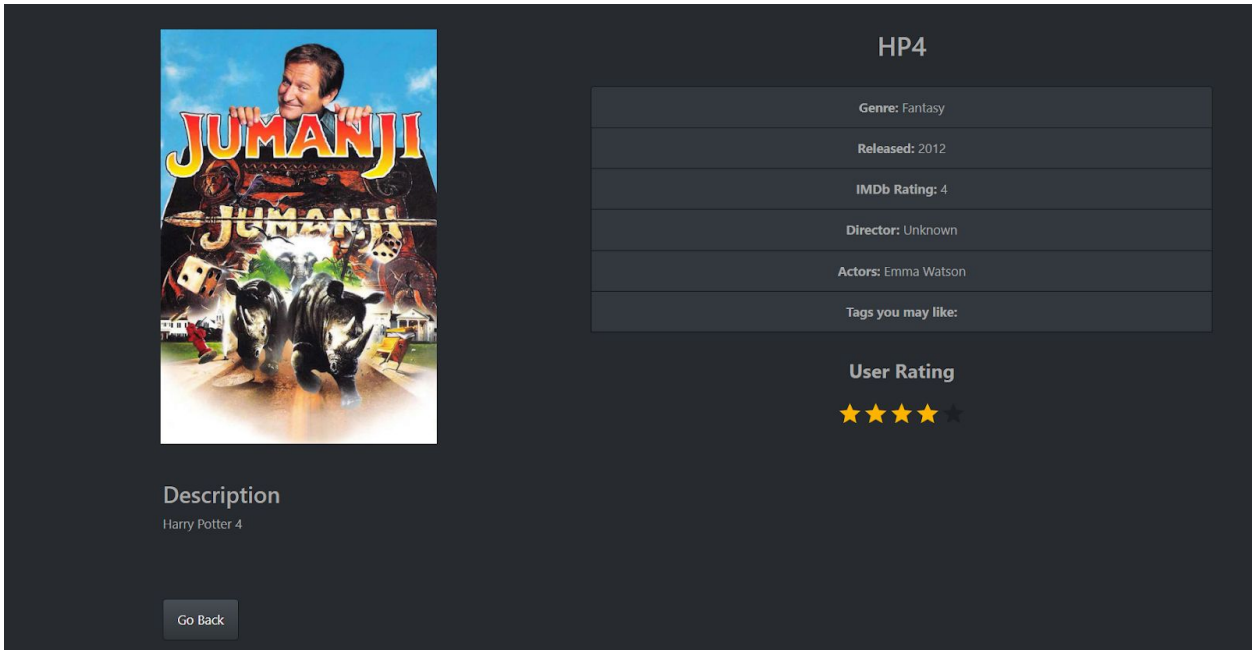

Figure 4.4 Movie Detail Page

Data Preprocessing

Data preprocessing is divided into three parts, data for frontend, data for user based collaborative filtering, and metadata based recommendation. All of these data are processed with operations like drop duplicates, drop none.

1. **Data for frontend**

   Our data from kaggle includes metadata of a movie like genre, release date, IMDb rating, cast, and crew information. However most of these data are stored in a JSON format string. One of the most difficult jobs is to parse these strings into a new table which contains data that can be inserted into mysql.

2. **TF-IDF method based on user prefered tag and movies tag**

   Data used in this method is the same as one used in the front end. But it may also need user's rating records to create a preferred tag set.

3. **Data for metadata based recommendation**

   The data used in this part is the same one as the front end. However, the preprocessing approach is different due to different requirements. The JSON format genres strings are transformed into lists after the genre texts were extracted. The names of directors were extracted by comparing the list of cast and crew. For cast, only the first three actors were

kept to not give the cast too much weight than it should be given. Finally, all of this information was concatenated into one string for each movie.

TF-IDF method based on user prefered tag and movies tag

The core code of this part is finished, but it still needs to test and be integrated into the backend.

Code can be found in the github backend folder.

Metadata-based recommendation

This part is finished. A csv file is generated with index being IDs of movies and each row contains ten top most recommended movies for that movie. Both the file and source code can be found in the github repo.

Database

We sorted out the data and extracted useful data into a table and stored it in the database. The database has been successfully connected**.** The following is the data we sorted into the database

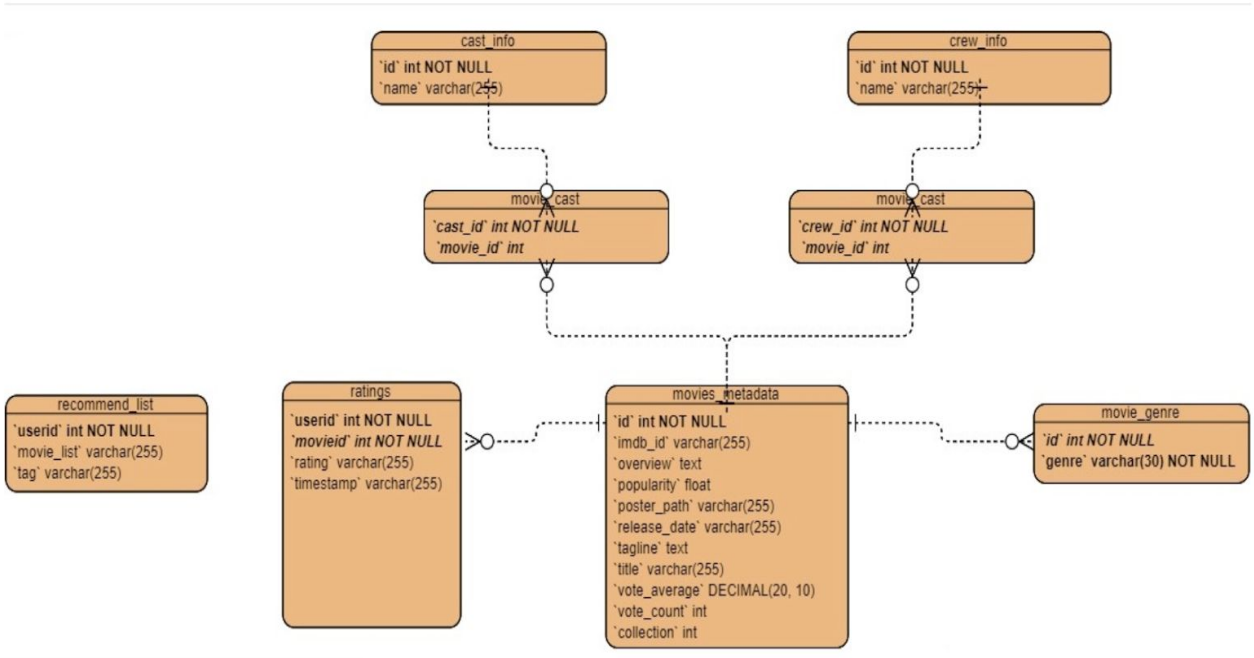

Figure 4.6 ERD diagram for database

	id	imdb_id	overview	popularity	poster_path
1	2	tt0094675	Taisto Kasurinen is a Finnish coal miner whose father has ju…	3.860491000000001	/gZCJZOn4l0Zj5hAxsMbxoS6CL0u.jpg
2	3	tt0092149	An episode in the life of Nikander, a garbage man, involving…	2.29211	/7ad4iku8cYBuB08g9yAU7tHJik5.jpg
3	5	tt0113101	It's Ted the Bellhop's first night on the job...and the hote…	9.026586	/eQs5hh9rxrk1m4xHsIz1w11Ngqb.jpg
4	6	tt0107286	While racing to a boxing match, Frank, Mike, John and Rey ge…	5.538671	/lNXmgUrP6h1nD53gkFh4WDzT6RZ.jpg
5	11	tt0076759	Princess Leia is captured and held hostage by the evil Imper…	42.149697	/btTdmkgIvOi0FFip1sPuZI2oQG6.jpg
6	12	tt0266543	Nemo, an adventurous young clownfish, is unexpectedly taken …	25.497794	/syPWyeeqzTQIxjIUaIFI7d0TyEY.jpg
7	13	tt0109830	A man with a low IQ has accomplished great things in his lif…	48.307194	/yE5d3BUhE8hCnkMUJOo1QDoOGNz.jpg
8	14	tt0169547	Lester Burnham, a depressed suburban father in a mid-life cr…	20.726578	/or1MP8BZIAjqWYxPdPX724ydKar.jpg
9	15	tt0033467	Newspaper magnate, Charles Foster Kane is taken from his mot…	15.811921	/oFWvF7OJfT2ydAAatlnsgChV4FP.jpg
10	16	tt0168629	Selma, a Czech immigrant on the verge of blindness, struggle…	10.684806	/7xizDTz4Yj4IYm2ud4f6EfEXe5H.jpg
11	17	tt0411267	Adèle and her daughter Sarah are traveling on the Welsh coas…	5.691508	/8fzjzQhLXl1afshhsE5Y3MGuco4.jpg
12	18	tt0119116	In 2257, a taxi driver is unintentionally given the task of …	24.30526	/zaFa1NRZEnFgRTv5OVXkNIZO78O.jpg
13	19	tt0017136	In a futuristic city sharply divided between the working cla…	14.487867	/qriaeUUwdmlgethK3aSAx68mG05.jpg
14	20	tt0314412	A Pedro Almodovar production in which a fatally ill mother w…	10.310508	/vVj92VBFUBzjqzLvBiPVVlJDp78.jpg
15	21	tt0060371	The Endless Summer, by Bruce Brown, is one of the first and …	1.378819	/jqsZQPxeMjcvVN5aFGgAk7qQodr.jpg
16	22	tt0325980	Jack Sparrow, a freewheeling 17th-century pirate who roams t…	47.32666500000001	/tkt9xR1kNX5R9rCebASKck44si2.jpg
17	24	tt0266697	An assassin is shot at the altar by her ruthless employer, B…	25.261865	/97fNAi62HawGjWru7PvVmF7RAbU.jpg
18	25	tt0418763	Jarhead is a film about a US Marine Anthony Swofford’s exper…	9.997032	/kmbyO0XUHRHcMyxVSZAWDdrpxIu.jpg
19	26	tt0352994	Eyal, an Israeli Mossad agent, is given the mission to track…	1.7099959999999998	/5BH0k5pauAqywpr4NL9nKYcKC6O.jpg
20	27	tt0411705	Matt, a young glaciologist, soars across the vast, silent, i…	18.847707	/bvGyYmmnQG5NPP7m2nuLhaiC8LH.jpg
21	30	tt1530535	Koji Morimato’s animated science fiction short story about h…	0.987992	/1A59xquD5rZeQRgjMfI4qu1ZvcQ.jpg

Figure 4.7 Table 1 movies_metadata_processed

release_date	tagline	title
1988-10-21		Ariel
1986-10-16		Shadows in Paradise
1995-12-09	Twelve outrageous guests. Four scandalous requests. And one …	Four Rooms
1993-10-15	Don't move. Don't whisper. Don't even breathe.	Judgment Night
1977-05-25	A long time ago in a galaxy far, far away...	Star Wars
2003-05-30	There are 3.7 trillion fish in the ocean. They're looking fo…	Finding Nemo
1994-07-06	The world will never be the same, once you've seen it throug…	Forrest Gump
1999-09-15	Look closer.	American Beauty
1941-04-30	It's Terrific!	Citizen Kane
2000-05-17	You don't need eyes to see.	Dancer in the Dark
2006-01-26	One of the living for one of the dead.	The Dark
1997-05-07	There is no future without it.	The Fifth Element
1927-01-10	There can be no understanding between the hands and the brai…	Metropolis
2003-03-07		My Life Without Me
1966-06-15		The Endless Summer
2003-07-09	Prepare to be blown out of the water.	Pirates of the Caribbean: The Curse of the Black Pearl
2003-10-10	Go for the kill.	Kill Bill: Vol. 1
2005-11-04	Welcome to the suck.	Jarhead
2004-02-05	He was trained to hate until he met the enemy.	Walk on Water
2004-07-16	2 lovers, one summer, and the 9 songs that defined them.	9 Songs
1995-12-23		Magnetic Rose

Figure 4.8 Table 1 movies_metadata_processed

vote_average	vote_count
7.1	44
7.1	35
6.5	539
6.4	79
8.1	6778
7.6	6292
8.2	8147
7.9	3438
8	1244
7.7	392
5.6	76
7.3	3962
8	666
7.2	78
7.8	23
7.5	7191
7.7	5091
6.6	776
6.5	20
5.1	103
7.9	11

Figure 4.9 Table 1 movies_metadata_processed

	id	name
1	31	Tom Hanks
2	12898	Tim Allen
3	7167	Don Rickles
4	12899	Jim Varney
5	12900	Wallace Shawn
6	7907	John Ratzenberger
7	8873	Annie Potts
8	1116442	John Morris
9	12901	Erik von Detten
10	12133	Laurie Metcalf
11	8655	R. Lee Ermey
12	12903	Sarah Freeman
13	37221	Penn Jillette
14	2157	Robin Williams
15	8537	Jonathan Hyde
16	205	Kirsten Dunst
17	145151	Bradley Pierce
18	5149	Bonnie Hunt
19	10739	Bebe Neuwirth
20	58563	David Alan Grier
21	1276	Patricia Clarkson
22	46530	Adam Hann-Byrd

Figure 5.0 Table 2: cast_info

	id	name
1	7879	John Lasseter
2	7879	John Lasseter
3	12891	Jill King
4	12891	Jill King
5	104330	Andrew Stanton
6	12892	Joel Cohen
7	12892	Joel Cohen
8	12893	Alec Sokolow
9	12894	Garrett Wang
10	74071	Bonnie Arnold
11	12895	Cirroc Lofton
12	1272806	Ed Catmull
13	12896	Herbert Jefferson Jr.
14	152660	Ralph Guggenheim
15	12897	Arlene Martel
16	1631574	Steve Jobs
17	8	Jeffery Quinn
18	1631585	Lee Unkrich
19	7883	Tanya Lemani
20	1278755	Ralph Eggleston
21	1168870	Annie Rose Buckley
22	1278758	Robert Gordon

Figure 5.1 Table 3 crew_info

	id	genre
1	862	Animation
2	21032	Animation
3	10530	Animation
4	15789	Animation
5	43475	Animation
6	22586	Animation
7	8587	Animation
8	18242	Animation
9	9479	Animation
10	15139	Animation
11	812	Animation
12	408	Animation
13	10020	Animation
14	10895	Animation
15	11827	Animation
16	10112	Animation
17	19042	Animation
18	10539	Animation
19	2300	Animation
20	12233	Animation
21	40651	Animation
22	9323	Animation

Figure 5.2 Table4 Movie_genre

	crew_id	movie_id	job
1	7879	862	Director
2	7879	862	Original Story
3	7879	10020	Executive Producer
4	7879	9487	Director
5	7879	9487	Story
6	7879	863	Director
7	7879	863	Original Story
8	7879	585	Executive Producer
9	7879	12	Executive Producer
10	7879	9806	Executive Producer
11	7879	4935	Executive Producer
12	7879	13925	Director
13	7879	13925	Producer
14	7879	13925	Writer
15	7879	920	Screenplay
16	7879	920	Director
17	7879	920	Original Story
18	7879	2062	Executive Producer
19	7879	10681	Executive Producer
20	7879	13053	Executive Producer
21	7879	14160	Executive Producer
22	7879	24480	Executive Producer

Figure 5.3 Table 5 Movie_crew

	cast_id	movie_id
1	31	862
2	1419935	568
3	1419935	31918
4	18373	9800
5	31	858
6	31	32562
7	18373	396
8	18373	857
9	31	11974
10	31	2619
11	18373	22176
12	18373	29968
13	31	9489
14	31	56235
15	31	863
16	1318751	41670
17	19254	240
18	19254	1816
19	31	19259
20	31	12309
21	1343811	637
22	1343811	8358

Figure 5.4 Table 6 Movie_cost

REST API

REST API we used Flask. The three methods defined by the HTTP protocol: POST, GET, and PUT. These correspond to the three traditional actions performed on data in a database: CREATE, READ, UPDATE. When the user clicks After searching for related movie names, we will return all relevant movie information from the database to the

user. API also supports users to create and update movie's rating. API information is listed in the Github readme file.

Github Repository

https://github.com/robert4213/Movie_Recommendation_System