Movie Recommendation System

Milestone 3 - 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 3 (Maximum is 5).

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

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

M: all rated movie with tag t

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

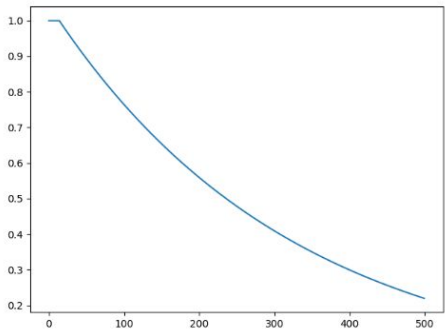

Figure 2.1 Time Variance

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.

Default List
Almost every recommended method may face the cold start question, when a new user comes, how can the system recommend projects to that user. In our project, we create a default recommended list. This list consists of movies with the top 20 highest popularity. The popularity value is stored in the original dataset. Before a user makes 2 ratings higher than 3 stars, the user may keep seeing the default list.

Movie Collections
We also consider movie collections in this project. When a user rates a movie which is involved in a collection, we will add another movie belonging to this collection into the recommended list, and the rest movies will be excluded from the recommended list. The reason why we exclude the rest movies is that movies in the same collection usually have the same tags, in other words, in our algorithm, they will be very easy to achieve highly estimated ratings. In the end, the recommended list will be filled with movies from the same collection, and this isn't an ideal recommendation, so we exclude rest movies from the recommendation.

Comparison

Let's check the top three rated recommenders for our dataset, The Movies Dataset.

First kernel is the Film recommendation engine. In this project, the developer uses 2 methods to recommend movies. First is to calculate euclidean distance between different movies and use this to recommend similar moves to users. Second movie is to use movies' popularity to recommend movies. The key attributions here are IMDB score, number of vote movies received, and year released.

Second one is Getting Started with a Movie Recommendation System. Their method is using collaborative filtering to recommend movies. They use user's rating to do Item Based Collaborative Filtering, and they use SVD to decrease computation load for prediction. However even with these methods, they still use the small rating dataset.

Third method is Movie Recommender Systems. This project creates a hybrid method to recommend movies, one is to Collaborative Filtering, another one is content based recommender(metadata based recommender), which is also used in our project. Again in this project, developers use small rating data.

In our project, we build our algorithm based on entire rating data, and consider one extra variant which no project considers, that is time variance. We introduce time variance to the algorithm, which can recommend movies to users according to users' current taste, compared with other projects, all user's ratings may have the same effect on recommendation. Our project uses the whole rating dataset, but can generate a recommended list in a few seconds, and do not need to train any model.

Work Distribution

Wenhao Tan: data preprocessing and metadata-based recommendation system

Shiyan Cai: data preprocessing and TF-IDF method based on user prefered tag, UI frontend

Yifan Liu: Database and Rest API

System Construction

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.

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

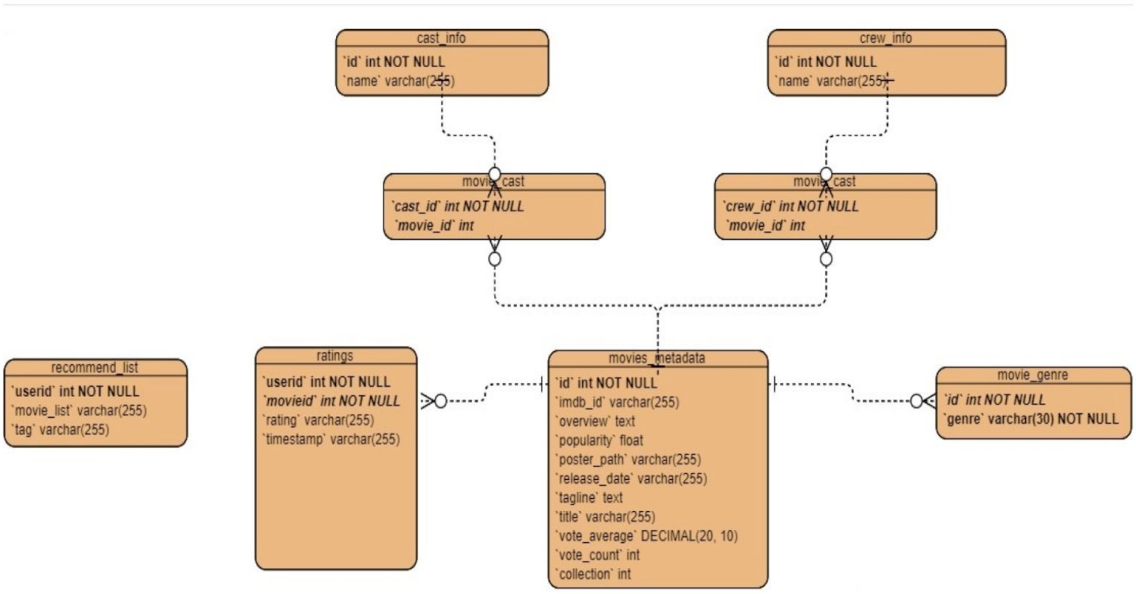

Figure 5.1 ERD diagram for database

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. In our Rest API part, all the front-end requirements are written in the readme on Github. The API mainly implements the above 5 functions. The first is that the customer can search for the movie, and then we need to return the customer's movie id, title, and movie picture from the database. we put these data in the movie_metadata table (look at Figure 5.1). The second function is to extract the required data from the ratings and movies_metadata table by entering the user id. In the same way, the third function is to return the required data by searching for a movie id and user ID. The fourth function is to get the recommended movie, it is the data generated by the recommended movie part from the back end. The last function is the Update Rating Record, so it can update data.

Front end

In our project, we use React.js to build our front end. Front end's task is to display movies' detail, search new movies, update or create ratings, check rating records, and provide a recommend list generated by the backend system.

Main page

Main page includes the following functions: search, display recommend list, and display rating history.

First is the search function, we leave a input text box here to allow users to enter movie names in order to find the movie. This function supports partial movie names.

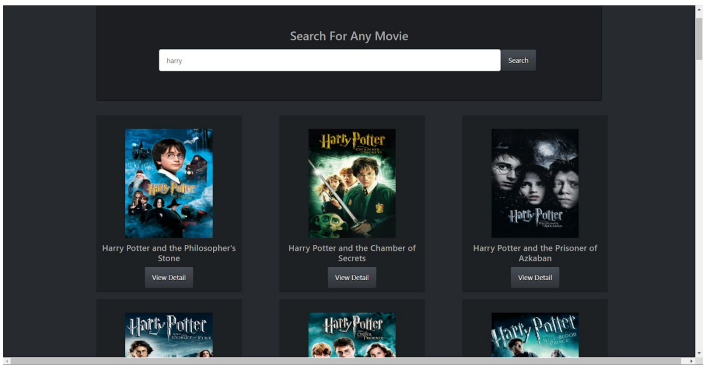

Figure 5.2 Search Function

Second is the recommend list, we use a slick window here to display the recommended movies. Users can use arrows on both sides to check different movies, and in the movie card, it may also display common tags between the recommended movie and the user preference. If a user has no rating, and system provides a default recommend list. There is no tag displayed.

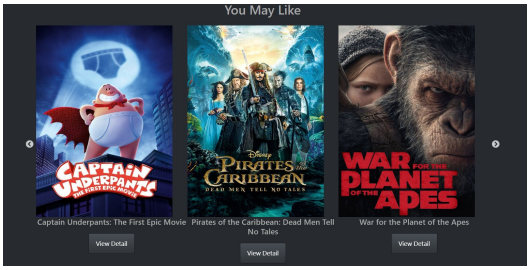

Figure 5.3 Recommend list

Third is the rating history, the function of this part is simple, display all user's rating history, including movie's poster, title, detail page link, and user's rating.

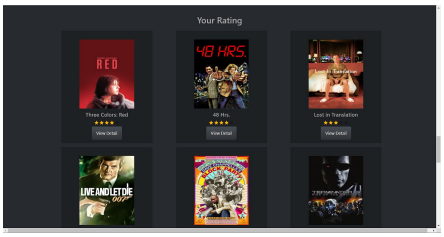

Figure 5.4 Rating history

Movie Detail page

The movie detail page is used to show movie's detail information, like genre, released date, director, cast, TMDb Rating. It can also show common tags in user preference and the movie.

Another important function in this page is to create or update a movie's rating from 1 star to 5 star. Once the rating was created or updated, the front end will send the new rating to the backend. Then, the user's rating history will be updated, and a new recommend list will be generated in the next few seconds.

This page may also show similar movies recommended by metadata based recommendation.

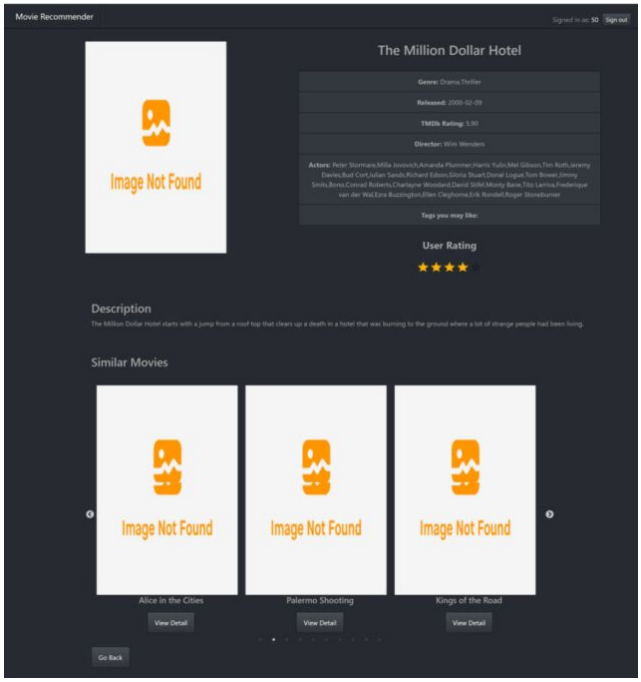

Figure 5.5 Movie Detail page

Nav Bar

In this project, we have thousands of users' records, and for different users, they need sign in and sign out functions to use their own account. And nav bar's purpose is to help users to finish previous actions.

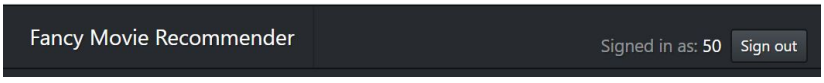

Figure 5.6 Nav Bar

In this project, users can directly login with their user id, and if you want to create a new account, what you need to do is just enter an id never used before.

Result

Metadata-based recommendation

In this method, we provide similar movies to users according to the current movie shown in the page. In our test, we use the movie Ultimate Avengers, whose genre is Action, Adventure, Animation, and Science Fiction. And recommendation movies are Ultimate Avengers 2, Batman Beyond: Return of the Joker, Superman: Brainiac Attacks, Batman: Mystery of the Batwoman,

Armitage: Dual Matrix, Batman Beyond: The Movie, Toward the Terra, Justice League: Throne of Atlantis, Kingsglaive: Final Fantasy XV, Batman Unlimited: Mechs vs. Mutants.
All of these movies have similar genres, casts or directors as the original movie.

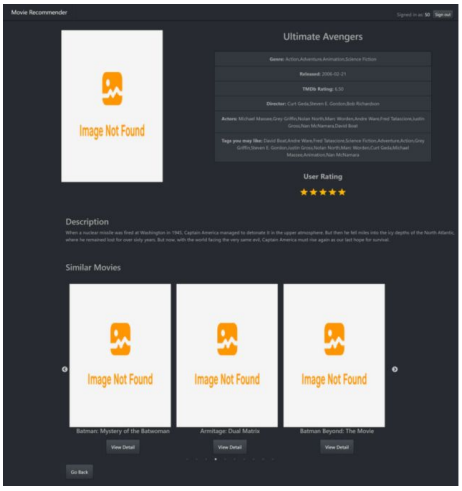

Figure 6.1 Metadata-based recommendation

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

After detecting no errors on running our frontend and backend, we tested our recommendation list in two different scenarios: rating as a new user and rating as an existing user with personalized recommendation list.

Starting as a new user by logging in with a brand new user ID which was not recorded in the database. We expected the new user to have no movies listed in the rated section and a default recommendation list in the recommendation section.

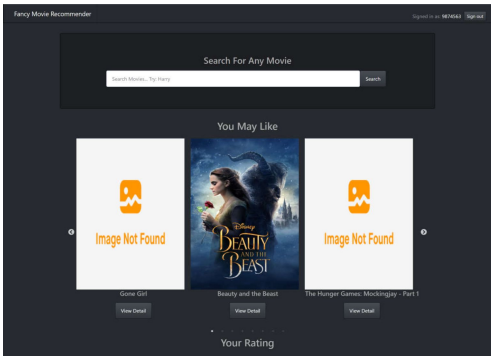

Figure 6.2 New User Created

According to the figure above, the new user has a default recommendation list regarding the movie popularity while having no movies rated in the rated-movie section. Then, we purposely searched for two movies which were both in action genres, having Tom Cruise as the main actor. After rating them with 5 stars, the recommendation list was updated with all movies in action and possibly having Tom Cruise as an actor. In the following images, Minority Report is shown in the recommended list, which Tom Cruise played a role in that.

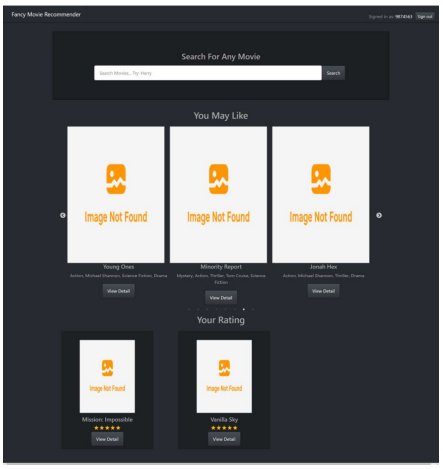

Figure 6.3 New Recommended List

Then we tested as an old user who already had rating records in the database. The rated movie list had all movies previously rated by this user, and the recommendation list was different from the default list as what we originally expected. At this moment, this user was interested in Drama and Thriller. Since all data was from the year of 2015, we can rate new movies to test the time variance algorithm.

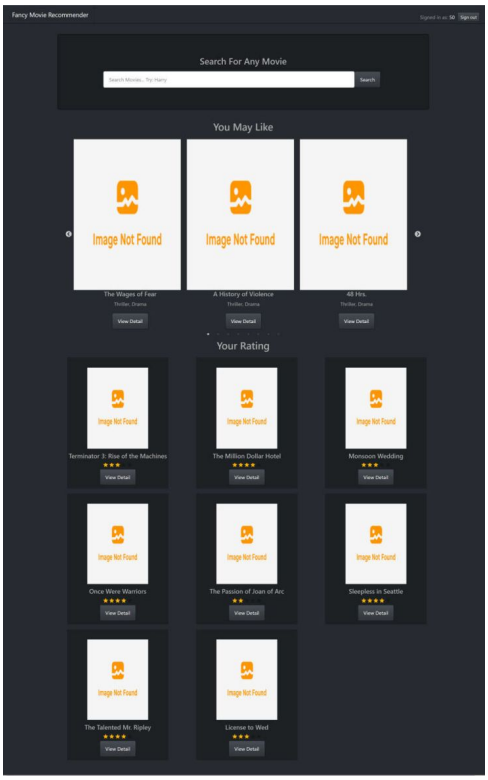

Figure 6.4 Old User Recommended List

After we rated a new movie with new genres and actors from previous favourites, we could see that the list was again updated since the old ratings did not have a relative high weight in the recommendation, and the new tastes would cover the current list according to the recommendation algorithm. At this moment, Drama and Thriller are no longer considered as the user's taste.

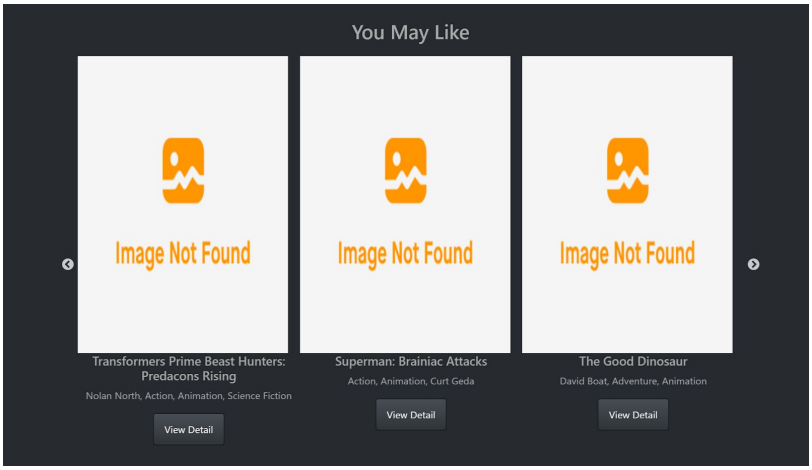

Figure 6.5 New User Recommended List

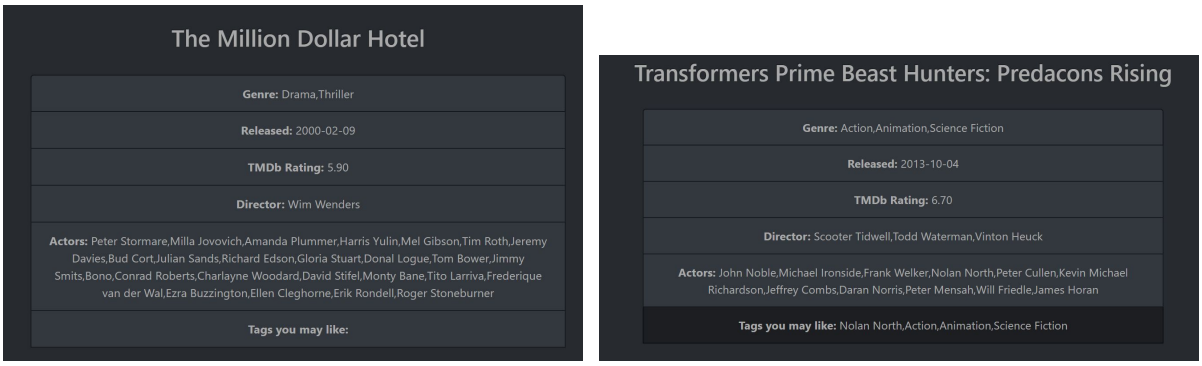

Figure 6.6 New User Taste

According to the tests, all results were following our expectations and the responding time was getting shorter for old users because the recommendation lists were stored in the database instead of being generated in real time.

Conclusion

Our project successfully introduced two recommendation algorithms to the movie dataset. And both algorithms can do a great and real time recommendation to users.

TF-IDF method based on user prefered tag inherents the advantages of a content-based system which is able to recommend movies based on the user's unique tastes, and also avoids cold start by generating a default recommendation list based on the movie's popularity.

Along with these two algorithms, we also built a web UI front end, a flask backend and a mysql database to help visualize our result.

Github Repository

https://github.com/robert4213/Movie_Recommendation_System

Youtube Link

https://youtu.be/4gl48FXEPNI