DSCI-551: Foundations of Data Management

Project Report: Movie Recommend System

Huiyun Peng, huiyunpe@usc.edu

Xiangyuan Chi, xiangyua@usc.edu

Instructor: Wensheng Wu
Nov. 28, 2022

Table of Contents

1. Topic

This project aims to provide a platform for movie lovers, especially cinephiles, to personalize their own movie database. Stepping into a post- pandemic era, people are more relied on streaming services, and hence both our movie search system and UI application not only help users to avoid information glut, but also feed users what they are interested in. There are four parts in our movie distributed system- genres, year, popularity, and language- each of which can be used for filtering movies and proposing a recommendation. Firstly, the decision-making process is done; finally the most pertinent information is shown.

2. Motivation

Living in the era of information overload, people put more focus on the quality of information instead of the quantity. Similar to personalized advertising, movie recommendation system filters movies in advance based on personal preference. In this way, people can save time in the process of picking movie. For users with the need of uploading movie item, the UI page gives the clear instruction on how to upload movie; for users who forget where the movie uploaded into, the UI page also lists the locations of the movie.

3. Firebase Database: Emulated Distributed File System

Emulated distributed file system (EDFS) is organized in Firebase database. Similar to HDFS, our EDFS has a NameNode called information table which stores the metadata, and four DataNodes, including genres, year, popularity, language, each of which partitions a key in information table and stores the partitioned content.

Based on the EDFS, our GUI is achieved using Tkinter, an open-source python package. In addition to the search function through DataNodes, we also add the command functions for users to easily upload a file or position which partitions a file is stored. The command functions not only help users to shorten the time of problem detection, but also optimize the efficiency of our frontend and backend code in python script and Jupyter notebook. The detailed architecture of the whole process is shown below in Figure 1.

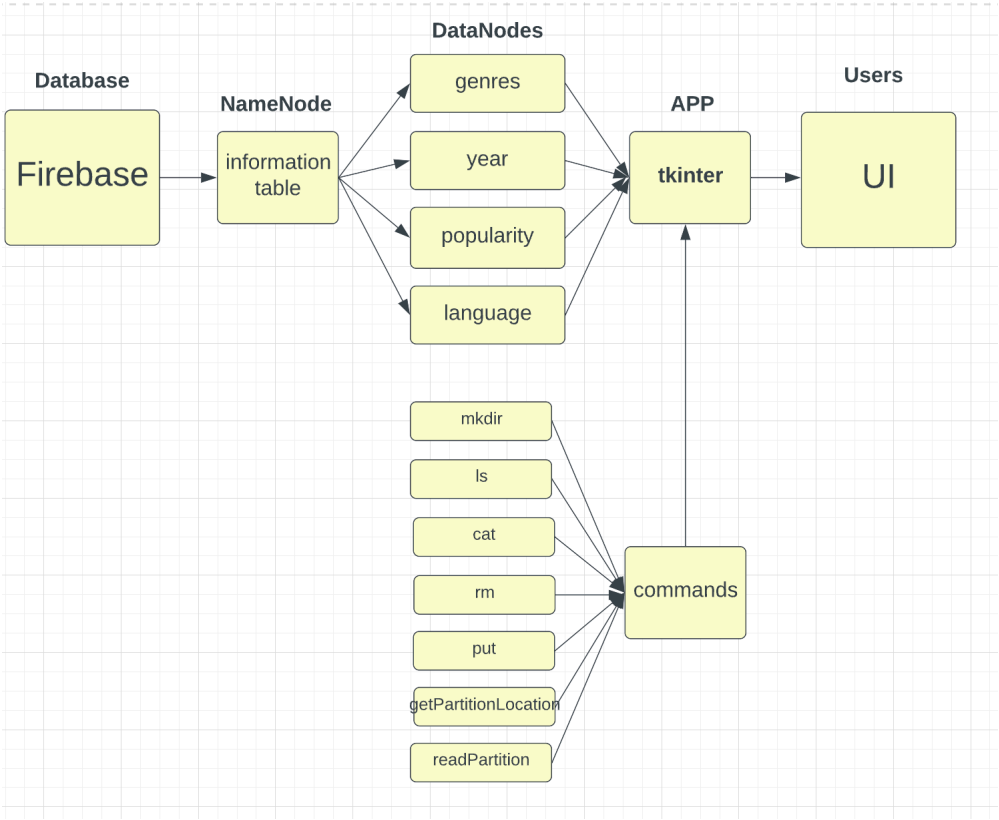

Fig. 1 Firebase Architecture & GUI Architecture

4. Data Flow and Work Flow

Our project includes two datasets: movies_metadata.csv and ratings.csv. The former has more than 45000 movies, which is too time-consuming for us to upload into Firebase, and hence we decide to randomly select a 5% sample set to upload to Firebase, but use the complete dataset for data analysis. The latter has more than three million rows of data, some of which is redundant, and hence we combine them based on the same key, and output a new dataset with smaller size called new_rate.csv.

Given the wrangled datasets, we import them into Jupyter notebook and complete data cleaning process. After uploading and partitioning the data into Firebase, we start to create an user interface application and perform data analysis. Overall, our UI App consists of one main page and 10 sub-pages, in which users can navigate by the button (explained in details in 6. Main Functions of App).

The flowchart of detailed process is shown below in Figure 2.

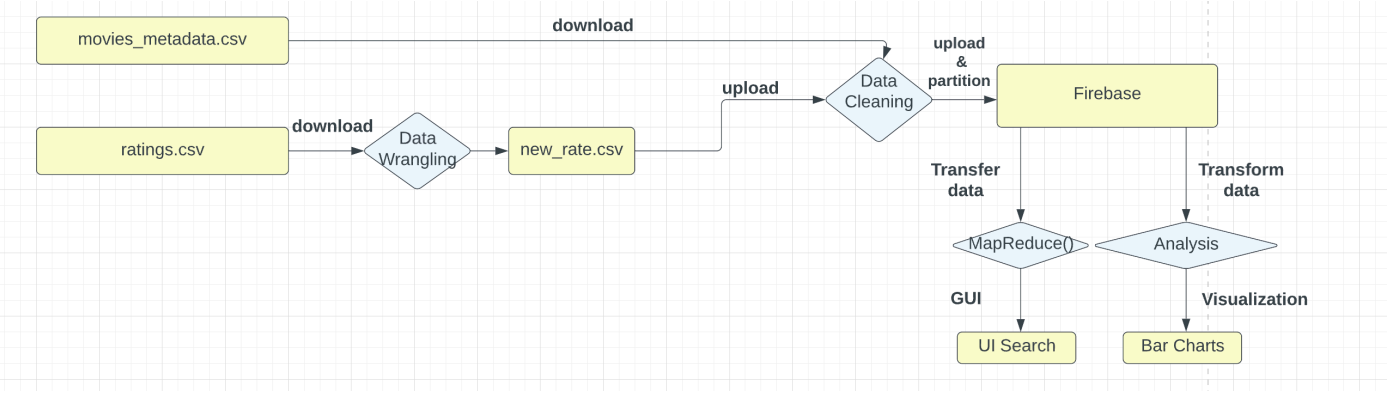

Fig. 2 Flowchart

5. Data Processing

5.1. Data Cleaning and Data Wrangling

Pandas library is used for data cleaning. For the movies_metadata.csv, we initially subset it according to what we need to upload into Firebase, and the number of attributes is halved, from 23 to 12. Then, we deal with the missing values by dropna(), and remove the outliers which are far below or above the average value.

Next, we need to standardize the string format of key-value pairs for Firebase upload. There are three attributes-genres, production_company, language-stored in a complicated JSON format (e.g.:[{'iso_639_1': 'en', 'name': 'English'}, {'iso_639_1': 'la', 'name': 'Latin'}]). In this case, we decide to create a check_dictionary to match the abbreviation with its full spelling. Then we iterate the value and convert it to readable value based on check_dictionary (Figure 3)

```
print(language_dict)
```

```
{'en': 'English', 'es': 'Spanish', 'fr': 'French', 'zh': 'Chinese', 'de': 'German', 'no': 'Norwegian', 'sv': 'Swedis
h', 'it': 'Italian', 'ru': 'Russian', 'la': 'Latin', 'gd': 'Scottish Gaelic', 'ar': 'Arabic', 'ga': 'Irish', 'ja': 'J
apanese', 'pt': 'Portuguese', 'el': 'Modern Greek (1453-)', 'ko': 'Korean', 'da': 'Danish', 'tr': 'Turkish', 'hu': 'H
ungarian', 'km': 'Central Khmer', 'mi': 'Maori', 'cs': 'Czech', 'bo': 'Tibetan', 'th': 'Thai', 'hi': 'Hindi', 'cy':
'Welsh', 'pl': 'Polish', 'nl': 'Dutch', 'sr': 'Serbian', 'nv': 'Navajo', 'uk': 'Ukrainian', 'is': 'Icelandic', 'tl':
'Tagalog', 'fi': 'Finnish', 'hy': 'Armenian', 'mn': 'Mongolian', 'vi': 'Vietnamese', 'bs': 'Bosnian', 'af': 'Afrikaan
s', 'mr': 'Marathi', 'iu': 'Inuktitut', 'lo': 'Lao', 'yi': 'Yiddish', 'gu': 'Gujarati', 'he': 'Hebrew', 'kk': 'Kazak
h', 'ca': 'Catalan', 'cr': 'Cree', 'ml': 'Malayalam', 'ka': 'Georgian', 'ro': 'Romanian', 'qu': 'Quechua', 'bn': 'Ben
gali', 'tg': 'Tajik', 'ta': 'Tamil', 'zu': 'Zulu', 'et': 'Estonian', 'hr': 'Croatian', 'sh': 'Serbo-Croatian', 'pa':
'Panjabi'}
```

Fig. 3 check_dictionary

5.2. Data Analysis and Visualization

Pandas library is used for data analysis, and matplotlib library is used for data visualization. We aim to find out the top 10 highly rated movies. In other words, we need to regularize the scores by weighted formula. Given the IMDB rating formula, we compute the mean value of our movies_metadata dataset, filtering the abnormal value by quantile. After computing the weight score, we sort it in descending order and pick the first 10 movies (Figure

4).

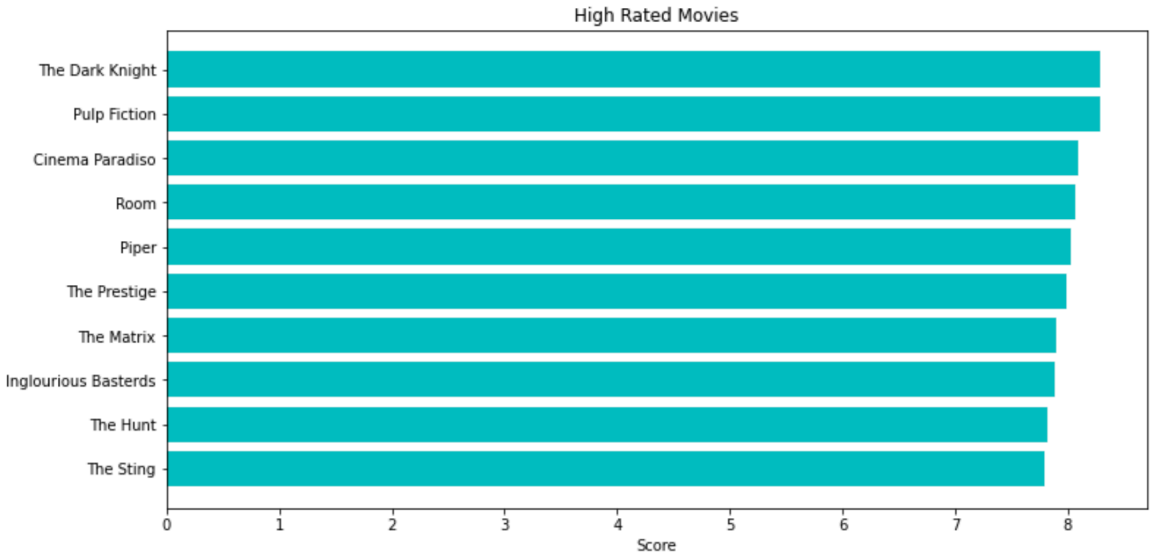

Fig. 4 Top 10 Highly Rated Movies

Furthermore, if we want to catch up with the most popular movies, we can simply sort the movies_metadata based on the popularity attribute, outputting it in descending order (Figure 5).

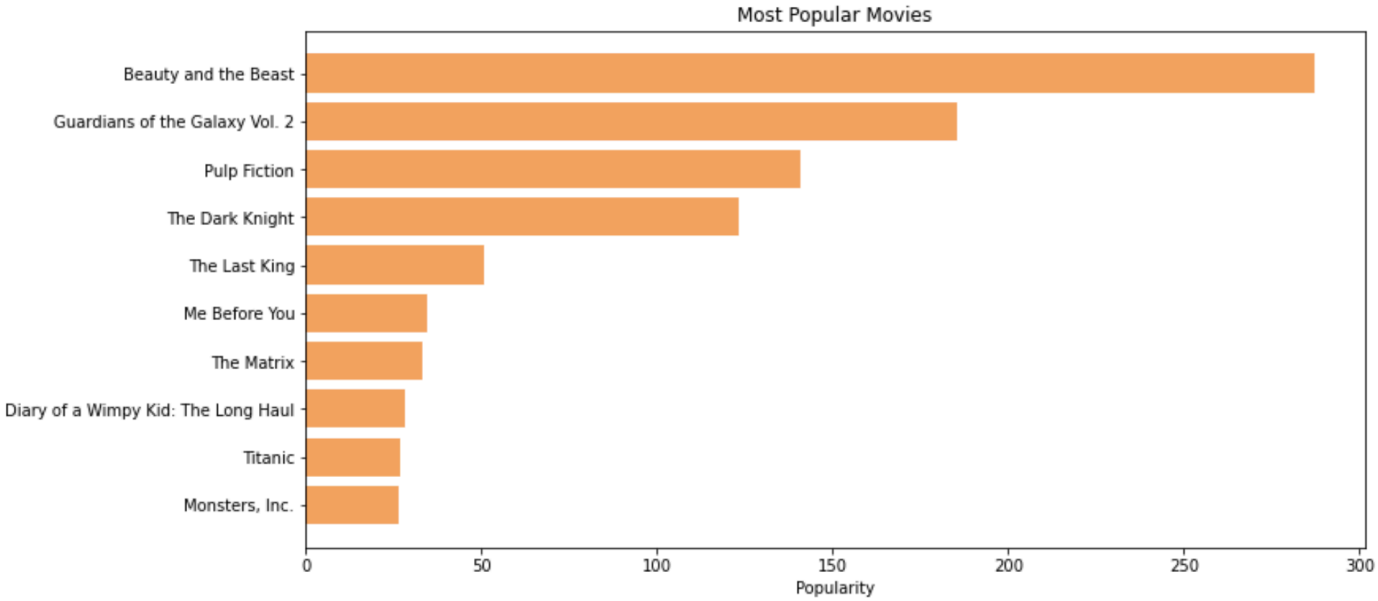

Fig. 5 Top 10 Popular Movies

5.3. Emulated Distributed File System Building

When the above two steps are done, we start to build EDFS. We need to create the

NameNode at first by uploading the processed movies_metadata into Firebase, stored as an information table (Figure 6). Then, we need to create replicas of the data by MapReduce() function. In this project, we partition the information table into 4 DataNodes. For example, if the genres of a movie “Big Fan” is [“Comedy, Drama”], the genres partition table will store as {“Comedy”:[“Big Fan”]} and {“Drama”:[“Big Fan”]}.

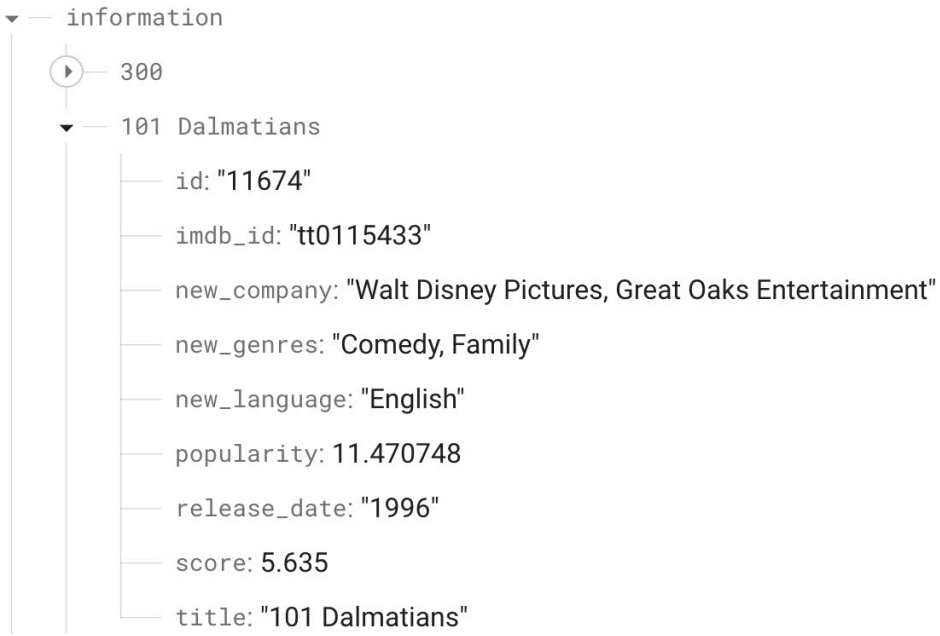

Fig 6.information table

6. Main functions of the app

There are three main functions we provided in our app (Fig. 7): build data, search movie and movie recommendation.

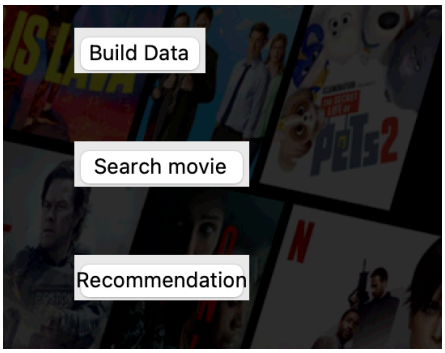

Fig 7. Overview of GUI app functions

6.1. Build data function introduction

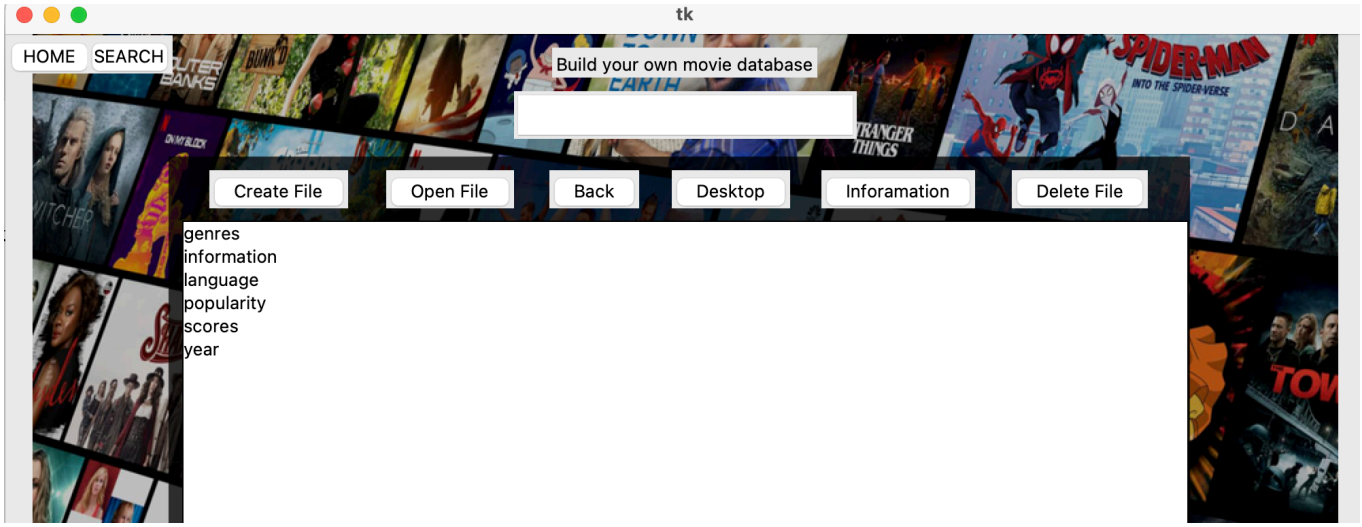

Fig. 8 Overview of the build movie data content

On this page(Fig. 8), users can build their own movie data and record their favorite movie name, genre, language, and release year. Firstly, users can create files under this page. Enter the file name in the entry box, then click create file bottom to create a file. Also, if the user wants to view the contents of the file, click the file name and then click the open file button to view the contents of the file(as shown in Fig. 9).

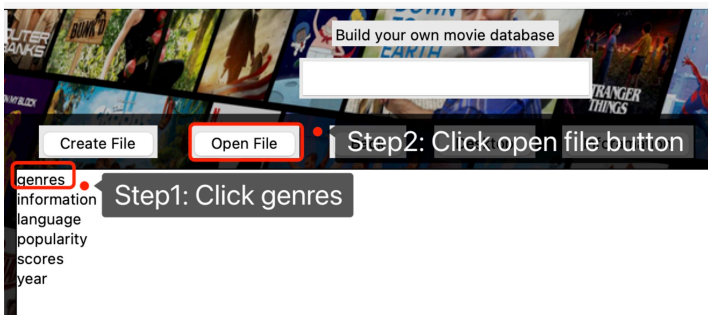

Fig. 9 Instruction of view content

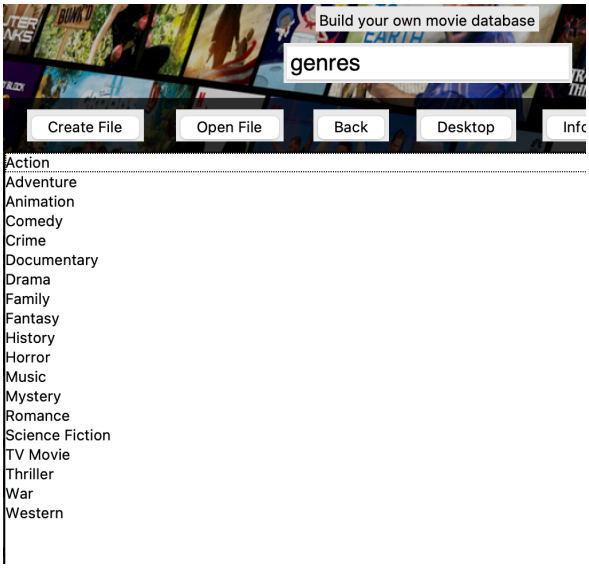

Fig. 10 Overview of genres content

After completing the above operations, users can see all types of movies in the genre file(Fig. 10).

As shown in Fig11, we have a delete button. If the user wants to delete the file or the content in the file, just click the content you want to delete, and then click the delete button to delete the thing you want to delete. The back button, it is to return to the previous layer of files.

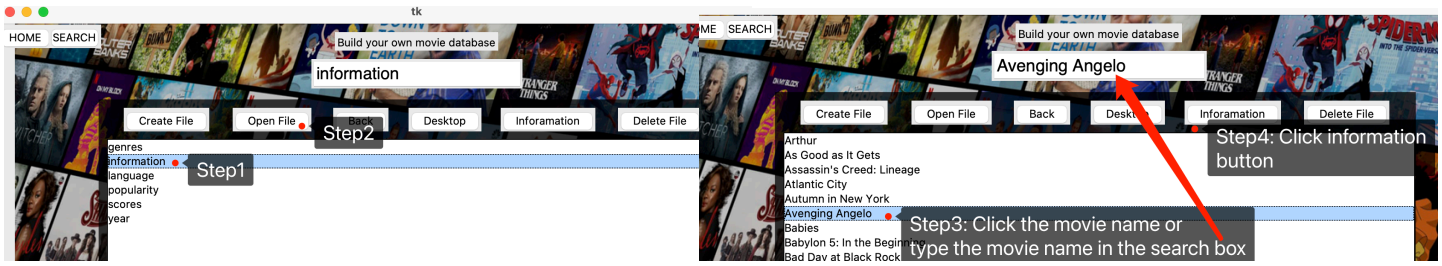

Fig. 11 Overview of information button function

The information button is used to view more detailed things. For example, click the movie in the information file, click the open file button, and then click the name of the movie user want to view as shown in Fig.11.

Then click the information button, user can see the specific information in this movie, such as genre, language, and release time as shown in Fig.12. If users want to delete or add some information in this file, it can be done as mentioned before.

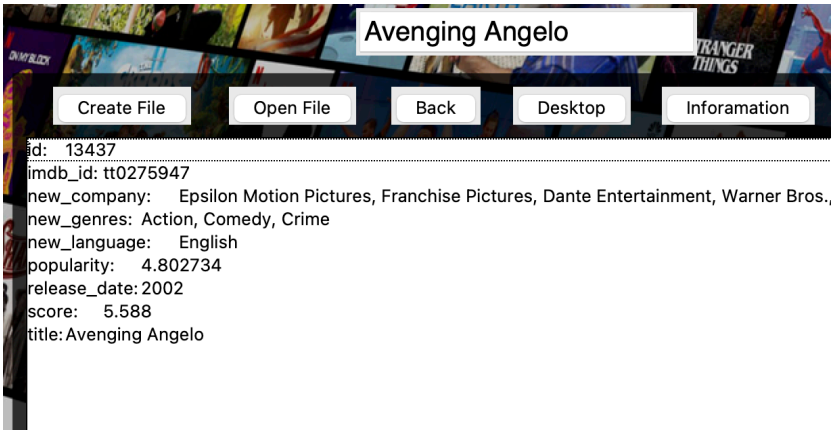

Fig. 12 Overview of information button function result

6.2. Search movie function introduction

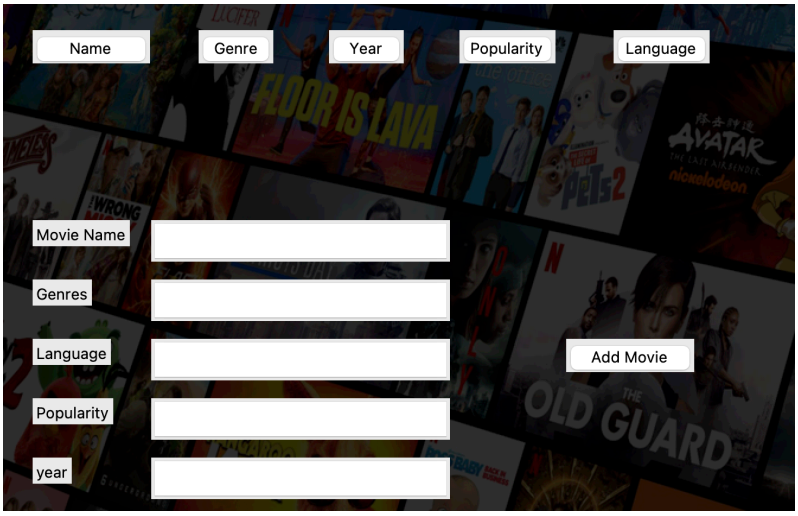

Fig. 13 Overview of search movie content

In the search movie interface, users can search movie based on movie name, genres, year, popularity and language. For example, if the user wants to watch a movie which release in 1999, click the button of the year and search for 1998 in the box. Then click the button of the open file to see all the movies released in 1998 as shown in Fig.13.

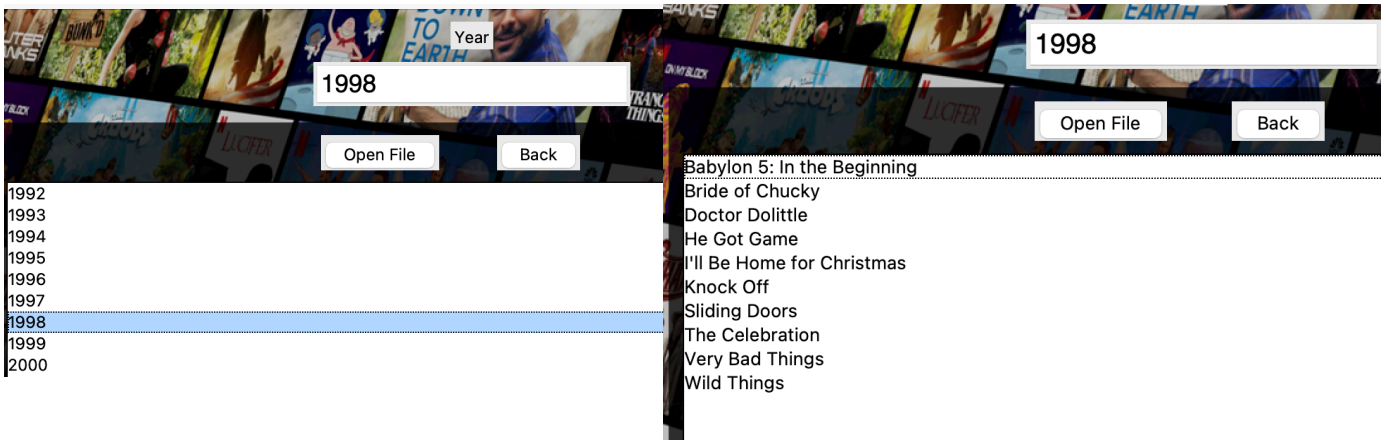

Fig. 14 Overview of search movie based on year content

In the search movie interface, users can add movie information to the firebase database. As shown in figure 13., by entering the movie name, genre, language, popularity, and release year, then clicking add movie button, this information directly upload to the firebase database. After adding movies' information, users can

search this new information by search name, genre, year, popularity score, and language (as shown in Fig.14). The purpose of this part is to help user to realize building their movie data and give others suggestion of whether the movie is great or bad through the popularity score.

6.3. Recommendation movie function introduction

In movie recommendation interface, when users enter a range of year and popularity score, the list box will show the movie's name which satisfy the range. As shown in Fig.15, if users want to search the movies which release start from 1930 to 1950, the box will show the movie name by clicking the search year button. In the same way, if users want to search based on popularity, just need to type the number in this interface.

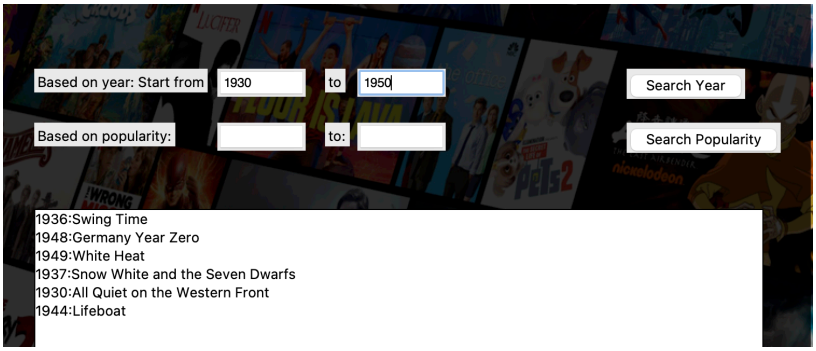

Fig. 15 Overview of movie recommendation content

7. Challenge

7.1. Tkinter

For the Tkinter, we did not find a lot of packages that fit our dataset and design. Since we use the firebase as our database and the format of our data is JSON, we need to transform the format of JSON to lists. Also, since our datasets are stored in the cloud, when we use each

DataNode, we have to change the JSON path in real time. It is very hard to capture the real path.

7.2. MapReduce Task: Database Partition

The MapReduce() process was complicated, we spent more than two weeks figuring out what the difference between map() and reduce() was in partition system. Besides the confusion of concept, we also had difficulty deciding the rule of partition, which meant that we had no clue for partition rules.

Appendix:

Team Member and Responsibility

Name	Responsibilities
Xiangyuan Chi	Data cleaning analysis and visualization, prediction model implementation, mapPartition and mapReduce
Huiyun Peng	GUI design and implement

Github Link: https://github.com/uscPenny/DSCI551_FinalProject
Google Drive Link: https://drive.google.com/drive/folders/1U5MHueXaBolr3WwumossoLAPffLZH4DK?usp=share_link
Data Link: https://drive.google.com/drive/folders/1U5MHueXaBolr3WwumossoLAPffLZH4DK?usp=share_link
Youtube Link: https://youtu.be/2EW-dUMYMHM
Data Reference: https://www.kaggle.com/code/ibtesama/getting-started-with-a-movie-recommendation-system/notebook