CS410: Text Information System Project Final Report

Team ACT

Table of Contents

1. Team ACT information

Member Name	Contact Details
Hyeonjae Cho (Captain)	hc53@illinois.edu
Xi Chen	xi12@illinois.edu
Sean Kan	clkan2@illinois.edu
Danwoo Park (NetID: danwoop2)	danwoop2@illinois.edu
Kihyuk Song	kihyuks2@illinois.edu

2. Overall Project Description

We built a movie search and recommendation system that gets input on any queries and returns a list of relevant films, with the following variables: “Title”, “Genre”, “Director”, and “Cast”. Together with the search results, a list of similar movies (recommendations) to the top search result will also be returned.

The movie recommendation system is made by exploiting publicly available data from the following websites:

- MovieLens 25M datasets for movie titles, genres, tags, and user ratings
    - https://grouplens.org/datasets/movielens/25m/
- IMDB for cast and director names
    - https://www.imdb.com/interfaces/
- WikiData for creating a knowledge graph that describes real-world entities and captures the relationships between them.
    - https://query.wikidata.org/

The reason why our team chose to do both a search and recommendation system is that as the lecture says, future intelligent information systems are expected to be highly personalized and alleviate users' effort to perform a task. The recommendation system is the way the information system desires to be. In addition, this project is important in that we are trying to utilize the WikiData Knowledge Graph algorithm, which is a state-of-the-art recommendation algorithm.

We built a BM25 model first which is used to retrieve movies relevant to the query. At the same time, we need to construct the knowledge graph with WikiData API extracting movies-related real-world entities and relationships. Since we have user rating data, we would like to develop a collaborative filtering model, which would recommend content purely based on the rating data. In the end, the hybrid model can be summarized as below:

Recommender = λ * Results from the collaborative filtering model + (1- λ) * Results from KG created by the WikiData API. Where λ is a parameter that we can control (λ = [0,1]). Currently, the value is set as 0.5.

The model output is shown in a simple demo web page that gets movie title inputs from users. The page would return search results that show related contents and entities inferred from the model. The codes and functions we have implemented would be elaborated more in the following sections.

3. Task Breakdown

Task	Estimated Time
Data Preprocessing - Data integration between MovieLense and IMDB (Movie, tag, genre, crews)	5 hrs
Text Retrieval System with BM25 - Use MeTA library to build our own BM25 model - Implement the search function on a remote server	20 hrs
Collaborative Filtering System - Based on MovieLens user rating data	15 hrs
WikiData Knowledge Graph - Use pre-built KG by WikiData API, extract movies-related entities from matched movie titles result	15 hrs
User Interface Design	5 hrs
Web App - Develop a demo web app that gets the search input and returns corresponding results - Front-end and Back-end integration	15 hrs
Main Processor Function - Integrate back-end functions and acts as the key interface between front-end and back-end	10 hrs
Testing and Debugging	13 hrs
Demo Preparation and Recording - Take a video that shows how our search result page and codes run behind	2 hrs

Total	100 hrs

4. Technical Details

4.1. Data Preprocessing

For data preprocessing, we have excluded all movie entries in the MovieLens25m dataset that either contain non-ASCII (non-English) characters or have an invalid id, merged values for duplicates, added in cast and director names by utilizing Kaggle and the TMDb API, combined all the relevant information (ie: movie titles, genres, tags, cast and director names) to create our corpus, and performed various data cleansing tasks. To ensure that data consumed by the collaborative filtering model coincides with our corpus, we've also removed user ratings for movies that are irrelevant to this project.

The following files were generated as a result of this data-wrangling process:

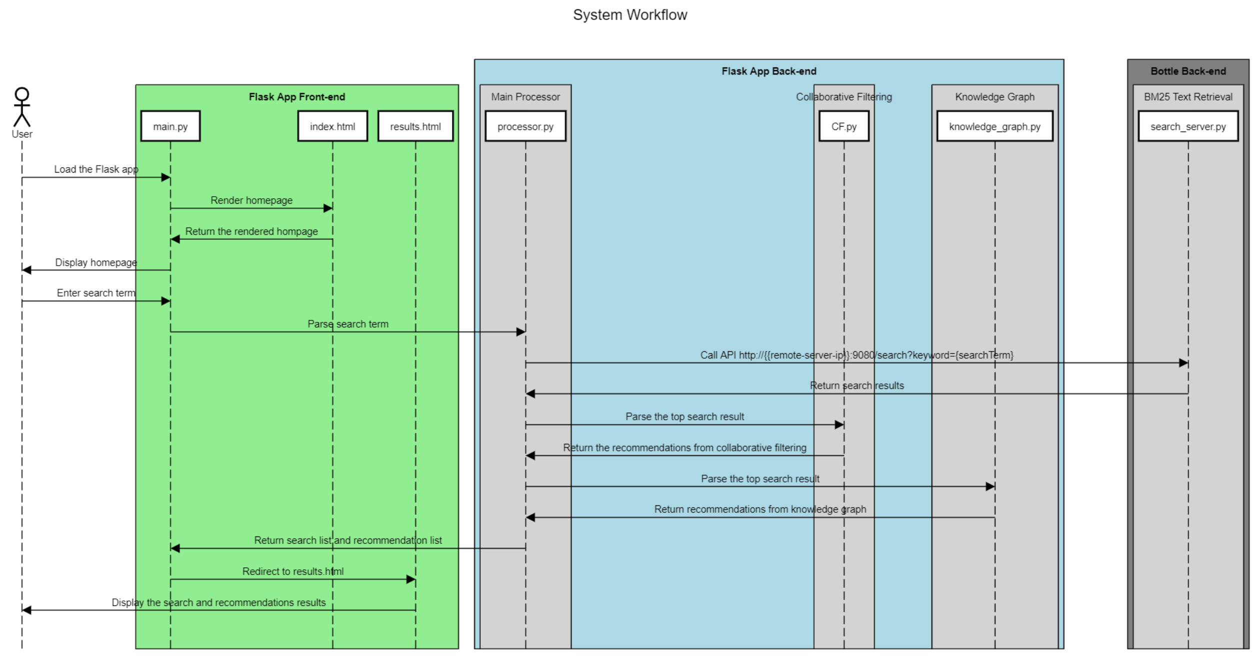
Corpus.txt – A list of movies along with information such as tags, genres, and names of cast and directors for our BM25 model

Ratings.csv – User ratings for the collaborative filtering system

Output.csv – Additional movie info for the collaborative filtering system

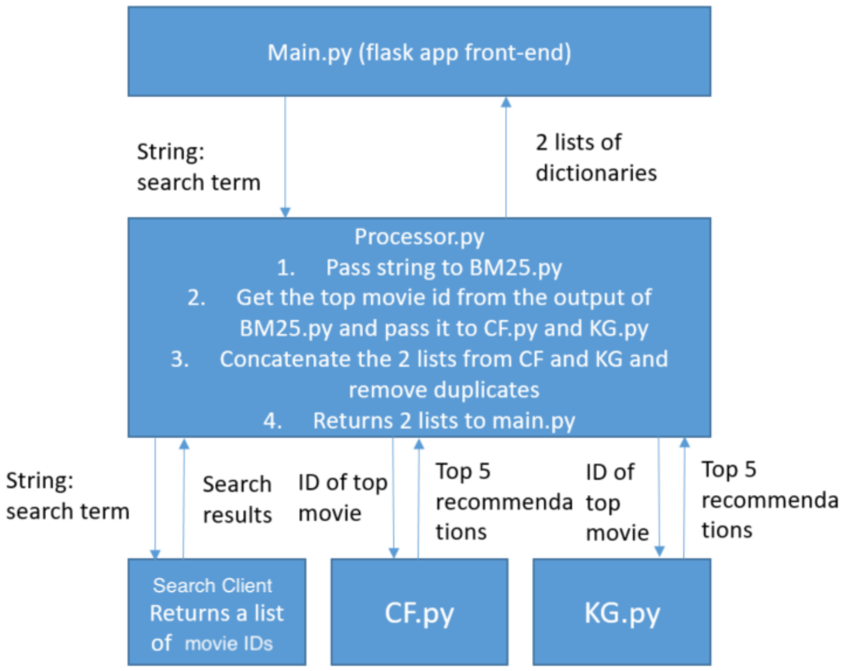
Since the MonvieLens25 dataset did not contain the names of cast and directors, the most challenging part of data preprocessing was finding the most effective way to obtain this data for nearly 60,000 movies. We originally planned to acquire them from IMDb, the world's largest movie database, but were thwarted by its download limit for free users. So we ended up extracting as much data as possible from a popular dataset on Kaggle and downloading the remaining parts via TMDb, a smaller open-source alternative to IMDb. The entire process was resource-intensive as it took us several hours to retrieve all the necessary information, not to mention the additional time needed to clean the data itself.

4.2. Overall System Workflow



4.3. Main Processor Function

The main purpose of the processor function is to integrate all necessary models so that the front end does not have to handle intermediate outputs.



1. Pass string to BM25.py
2. Get the top movie id from the output of BM25.py and pass it to CF.py and KG.py
3. Concatenate the 2 lists from CF and KG and remove duplicates
4. Returns 2 lists to main.py

Flow Chart of Processor.py

As the flow chart above shows when a user gives input, the main processor calls the API of the search client to run BM25. BM25 takes a search keyword and returns a list of movie IDs that are determined by movie information the model has.

Then the very beginning item of the list is given to the collaborative filtering and knowledge graph as input. Both models run their own algorithms to discover the recommendable movie IDs utilizing their pre-trained knowledge. Collaborative filtering is based on rating data and the knowledge graph is from WikiData. We concatenated two results while removing duplicates. Those outputs are then combined with additional information to be displayed on the search result page. The information about actors, directors, and genres is involved to help the user grasp the recommended movies.

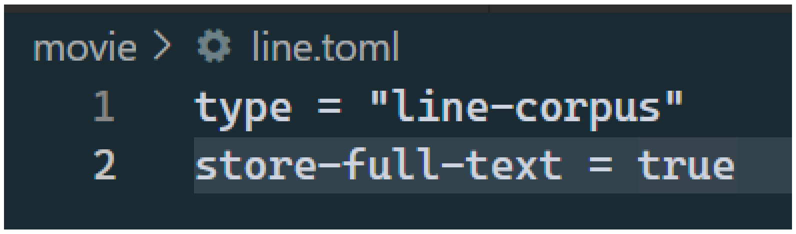
Finally, the list of dictionaries containing movie recommendations is passed to the front end.

4.4. Text Retrieval System

By using the “corpus.txt” file we created above, we were able to create a simple search system using the MeTA toolkit.

In the section above, we created a corpus.txt file that contains a list of movies with their metadata such as the title of the movie, actors, and genres. Each line of the corpus.txt file can be considered as a single document that represents a movie.

Since the MeTA toolkit supports creating an inverted index by config, we used the following setting to see the ranked results (list of movies found).

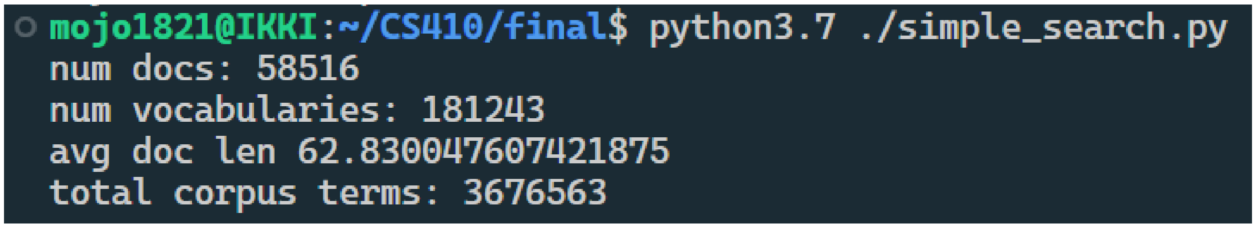


movie > line.toml

```
type = "line-corpus"
store-full-text = true
```

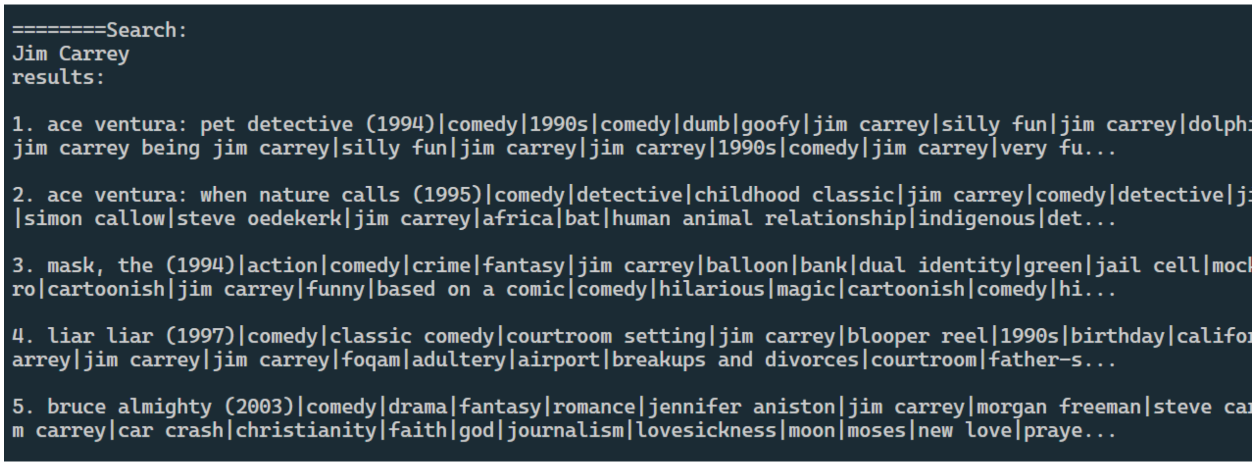
(a config file for creating an inverted index)

After creating an inverted index for our corpus.txt file, we can see the following statistics about our documents.

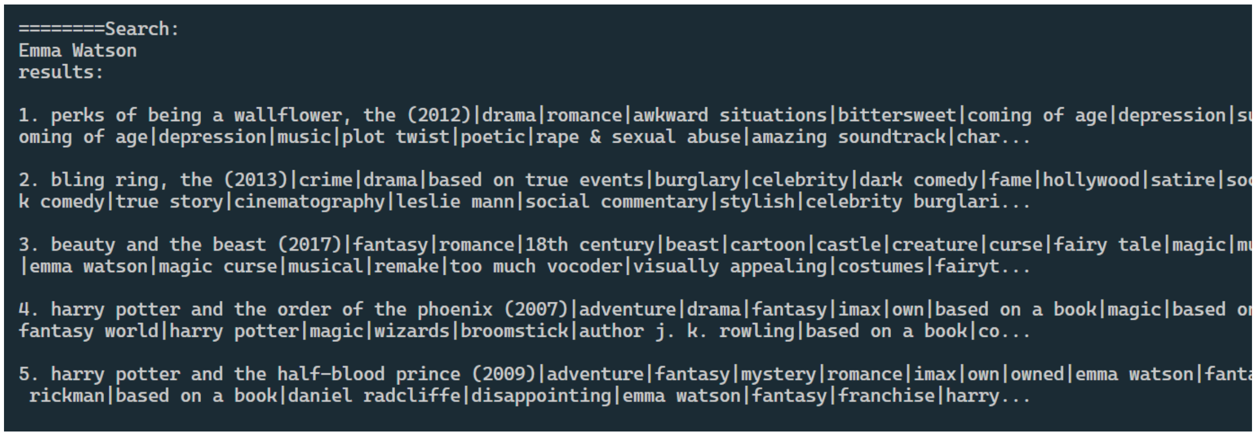


```
mojo1821@IKKI:~/CS410/final$ python3.7 ./simple_search.py
num docs: 58516
num vocabularies: 181243
avg doc len 62.830047607421875
total corpus terms: 3676563
```

The following screenshots are a few examples of search queries and their corresponding results:



```
========Search:
Jim Carrey
results:

1. ace ventura: pet detective (1994)|comedy|1990s|comedy|dumb|goofy|jim carrey|silly fun|jim carrey|dolphi
jim carrey being jim carrey|silly fun|jim carrey|jim carrey|1990s|comedy|jim carrey|very fu...

2. ace ventura: when nature calls (1995)|comedy|detective|childhood classic|jim carrey|comedy|detective|ji
|simon callow|steve oedekerk|jim carrey|africa|bat|human animal relationship|indigenous|det...

3. mask, the (1994)|action|comedy|crime|fantasy|jim carrey|balloon|bank|dual identity|green|jail cell|mock
ro|cartoonish|jim carrey|funny|based on a comic|comedy|hilarious|magic|cartoonish|comedy|hi...

4. liar liar (1997)|comedy|classic comedy|courtroom setting|jim carrey|blooper reel|1990s|birthday|califor
arrey|jim carrey|jim carrey|foqam|adultery|airport|breakups and divorces|courtroom|father-s...

5. bruce almighty (2003)|comedy|drama|fantasy|romance|jennifer aniston|jim carrey|morgan freeman|steve car
m carrey|car crash|christianity|faith|god|journalism|lovesickness|moon|moses|new love|praye...
```



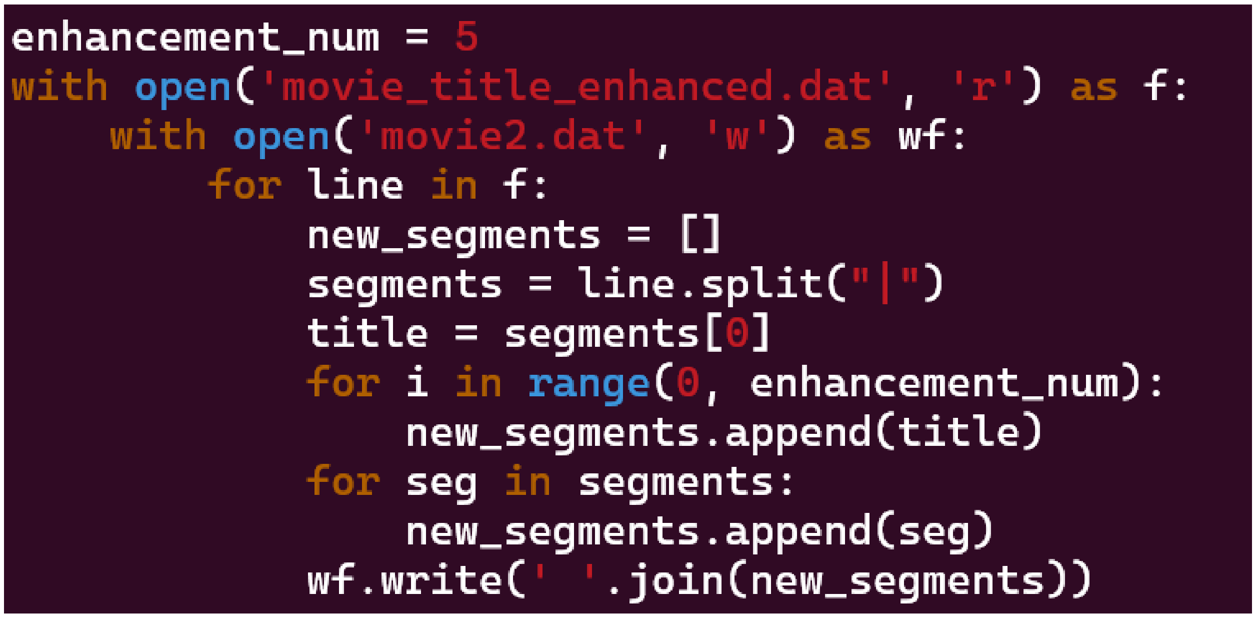
```
========Search:
Emma Watson
results:

1. perks of being a wallflower, the (2012)|drama|romance|awkward situations|bittersweet|coming of age|depression|su
oming of age|depression|music|plot twist|poetic|rape & sexual abuse|amazing soundtrack|char...

2. bling ring, the (2013)|crime|drama|based on true events|burglary|celebrity|dark comedy|fame|hollywood|satire|soc
k comedy|true story|cinematography|leslie mann|social commentary|stylish|celebrity burglari...

3. beauty and the beast (2017)|fantasy|romance|18th century|beast|cartoon|castle|creature|curse|fairy tale|magic|mu
|emma watson|magic curse|musical|remake|too much vocoder|visually appealing|costumes|fairyt...

4. harry potter and the order of the phoenix (2007)|adventure|drama|fantasy|imax|own|based on a book|magic|based on
fantasy world|harry potter|magic|wizards|broomstick|author j. k. rowling|based on a book|co...

5. harry potter and the half-blood prince (2009)|adventure|fantasy|mystery|romance|imax|own|owned|emma watson|fanta
 rickman|based on a book|daniel radcliffe|disappointing|emma watson|fantasy|franchise|harry...
```

Search Improvements

In the final project, we evaluated the search system empirically by throwing some queries to the search system and seeing the results. Some queries like “Jumanji” showed weird ranking results. When we used “jumanji” as a search term, “zathura” came out as a top search result. The reason for such a result was because the movie “Zathura” contained a list of tags such as “wanna be Jumanji”, “poor sequel to Jumanji”. Also, the length of the document for the movie “Zathura” was relatively shorter than “Jumanji” series movies. Furthermore, we were not dealing with the title words more significantly than other words that are contained in the document, which means if the documents with a title “A” contain other movie names such as “B” or “C”, those movies can show up in the results for a search term “A”. Therefore, we can come up with a few solutions.

1. We can enhance title words by pre-processing documents.
2. We can adjust the document length normalization parameter for the Okapi BM25 ranking algorithm to do less normalization. (lowering the “b” value would do that)

Preprocessing

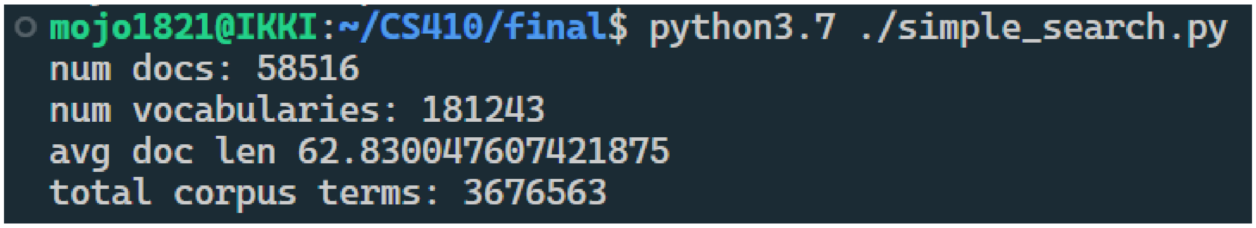


```
enhancement_num = 5
with open('movie_title_enhanced.dat', 'r') as f:
    with open('movie2.dat', 'w') as wf:
        for line in f:
            new_segments = []
            segments = line.split("|")
            title = segments[0]
            for i in range(0, enhancement_num):
                new_segments.append(title)
            for seg in segments:
                new_segments.append(seg)
            wf.write(' '.join(new_segments))
```

Using the above script, the “movie2.dat” file that contains documents to be indexed will have five times more title words than the original documents.

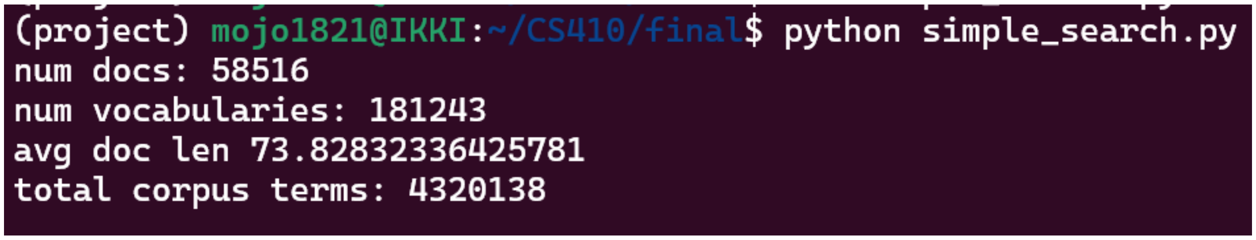
Below is the comparison of the results before and after the preprocessing:

Before enhancing the title words



```
○ mojo1821@IKKI:~/CS410/final$ python3.7 ./simple_search.py
  num docs: 58516
  num vocabularies: 181243
  avg doc len 62.830047607421875
  total corpus terms: 3676563
```

After enhancing the title words



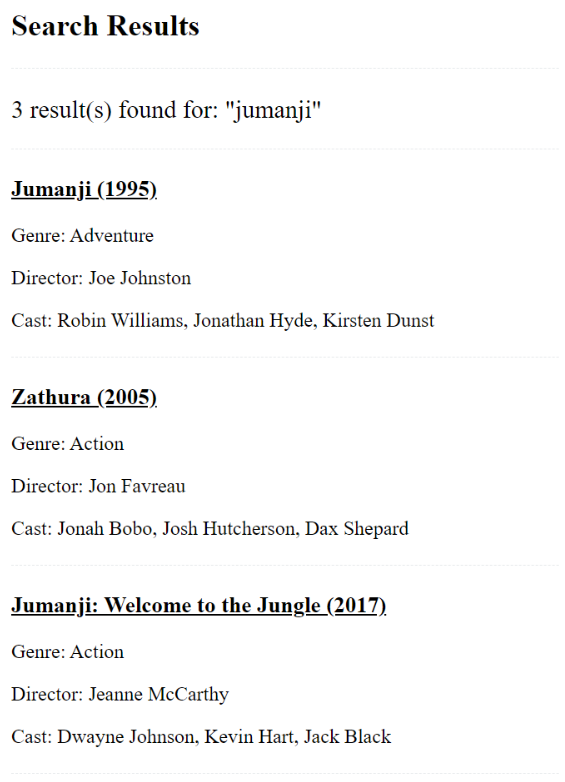
```
(project) mojo1821@IKKI:~/CS410/final$ python simple_search.py
num docs: 58516
num vocabularies: 181243
avg doc len 73.82832336425781
total corpus terms: 4320138
```

You can see that the number of documents are the same but the average document length has been increased and total corpus terms has been increased because of the enhanced title words.

Parameter Optimization

As I mentioned above, some movie documents lack meta data such as tags, which means they have shorter document length. Famous movies tend to have more tags than less famous movies, and if we want to favor the famous movies, we need to make the document length normalization less effective. As a result, after a number of experiments, we finally fixed the “b” value to 0.3 (0 means no document length normalization). Both k1 and k3 values are set to empirically proved values, which is 1.2 and 6.

As a result, we can now see desired results like the following:



Search Results

3 result(s) found for: "jumanji"

Jumanji (1995)

Genre: Adventure

Director: Joe Johnston

Cast: Robin Williams, Jonathan Hyde, Kirsten Dunst

Zathura (2005)

Genre: Action

Director: Jon Favreau

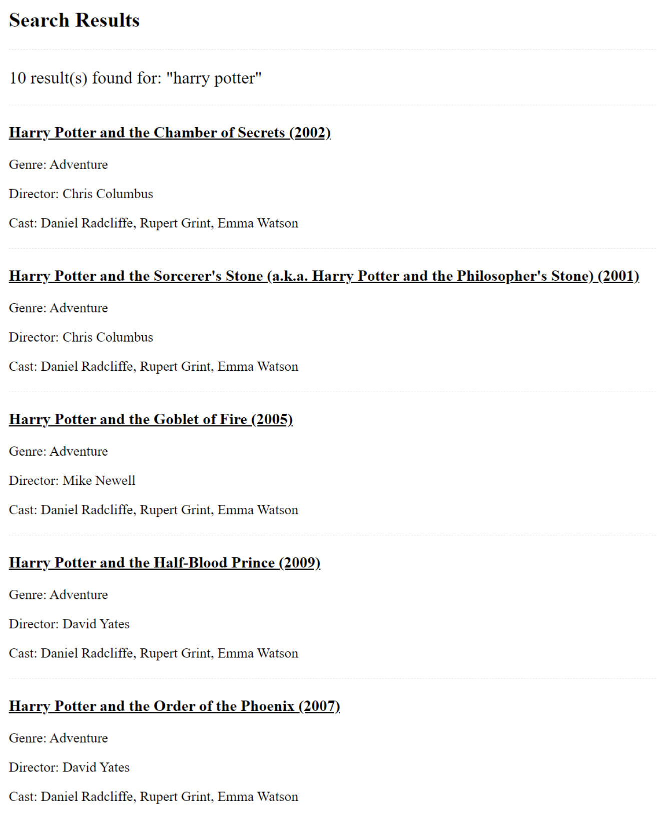
Cast: Jonah Bobo, Josh Hutcherson, Dax Shepard

Jumanji: Welcome to the Jungle (2017)

Genre: Action

Director: Jeanne McCarthy

Cast: Dwayne Johnson, Kevin Hart, Jack Black



Search Results

10 result(s) found for: "harry potter"

Harry Potter and the Chamber of Secrets (2002)

Genre: Adventure

Director: Chris Columbus

Cast: Daniel Radcliffe, Rupert Grint, Emma Watson

Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)

Genre: Adventure

Director: Chris Columbus

Cast: Daniel Radcliffe, Rupert Grint, Emma Watson

Harry Potter and the Goblet of Fire (2005)

Genre: Adventure

Director: Mike Newell

Cast: Daniel Radcliffe, Rupert Grint, Emma Watson

Harry Potter and the Half-Blood Prince (2009)

Genre: Adventure

Director: David Yates

Cast: Daniel Radcliffe, Rupert Grint, Emma Watson

Harry Potter and the Order of the Phoenix (2007)

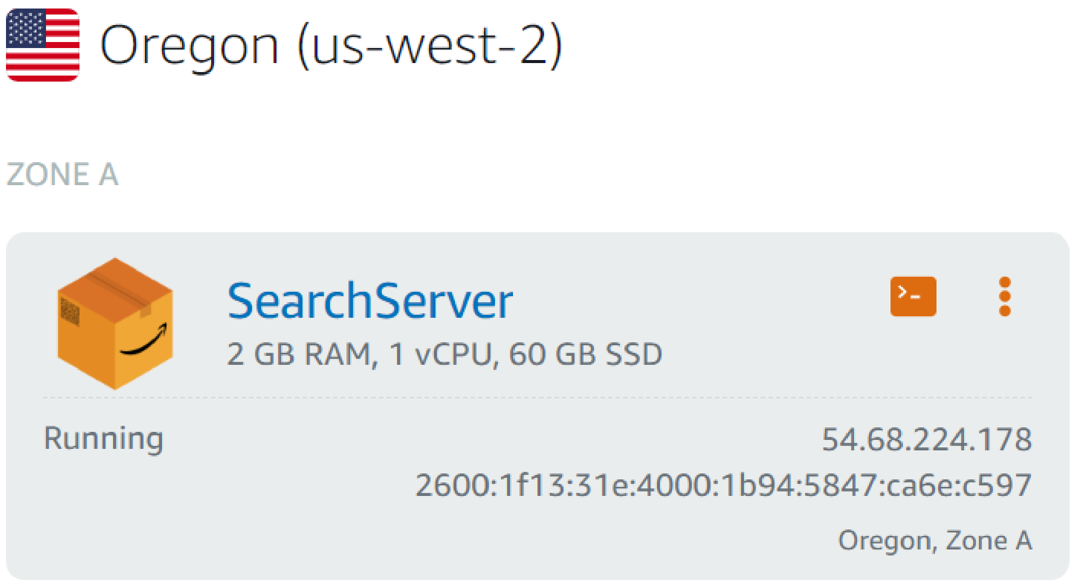
Genre: Adventure

Director: David Yates

Cast: Daniel Radcliffe, Rupert Grint, Emma Watson

Troubleshooting

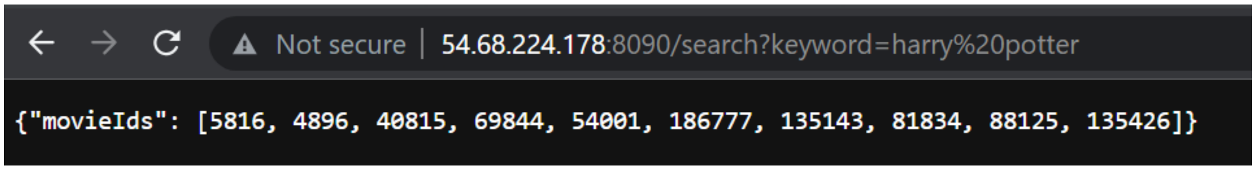
Since Metapy was not working with Flask, which we used to serve our application, the search server used a different framework called “bottle”. The search server runs independently and provides an HTTP JSON API **GET /search?keyword={searchTerm}**.
The search server returns a list of movie IDs for a given *searchTerm*.
Furthermore, the search server is running on an AWS Lightsail instance and can be accessed remotely. This was because it was hard to make Metapy run on Apple silicon-based machines, so I created a Linux instance on the AWS cloud instance and made it public.
For example, to get the search results for “Harry Potter”, you can type in the following URL on the browser.


Oregon (us-west-2)

ZONE A

SearchServer
2 GB RAM, 1 vCPU, 60 GB SSD
Running
54.68.224.178
2600:1f13:31e:4000:1b94:5847:ca6e:c597
Oregon, Zone A

http://54.68.224.178:8090/search?keyword=harry%20potter


Not secure | 54.68.224.178:8090/search?keyword=harry%20potter

```
{"movieIds": [5816, 4896, 40815, 69844, 54001, 186777, 135143, 81834, 88125, 135426]}
```

By doing this, you don't need to run the search server locally.

4.5. Collaborative Filtering System

Collaborative filtering(CF) is a method that predicts users' interests based on preference data observed by many users and is a technique used in recommendation systems.[1]

The fundamental assumption of CF: *the past trends of users will remain the same in the future*

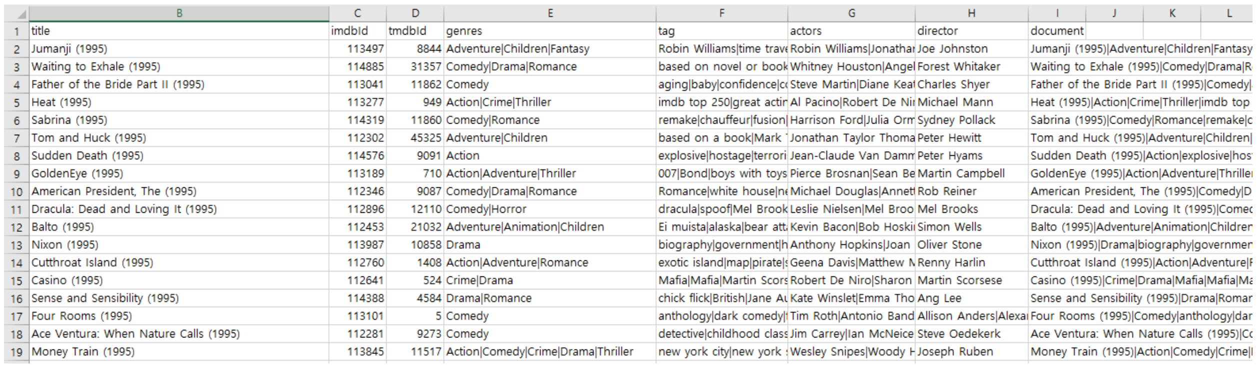
Based on the assumption, users sharing similar patterns in their preferences and interests can be identified based on their implicit feedback such as ratings.

In this project, we implemented two collaborative filtering models using Python.

1) User-based CF: returns a list of users based on user similarity scores
2) Item-based CF: returns a list of movies based on item similarity scores

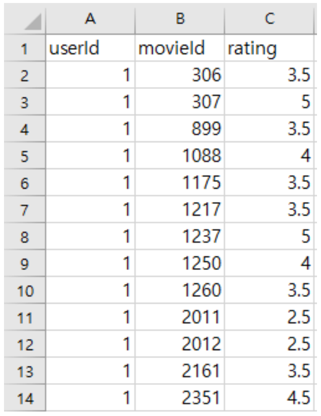
[1] https://en.wikipedia.org/wiki/Collaborative_filtering

We utilized two preprocessed data to build the CF model. First, user rating data from MovieLense and second, the movie information that contains the brief movie information.



	B	C	D	E	F	G	H	I	J	K	L
1	title	imdbId	tmdbId	genres	tag	actors	director	document			
2	Jumanji (1995)	113497	8844	Adventure|Children|Fantasy	Robin Williams|time trav	Robin Williams|Jonathar	Joe Johnston	Jumanji (1995)|Adventure|Children|Fantasy			
3	Waiting to Exhale (1995)	114885	31357	Comedy|Drama|Romance	based on novel or book	Whitney Houston|Angel	Forest Whitaker	Waiting to Exhale (1995)|Comedy|Drama|R			
4	Father of the Bride Part II (1995)	113041	11862	Comedy	aging|baby|confidence|c	Steve Martin|Diane Keat	Charles Shyer	Father of the Bride Part II (1995)|Comedy|			
5	Heat (1995)	113277	949	Action|Crime|Thriller	imdb top 250|great actir	Al Pacino|Robert De Nir	Michael Mann	Heat (1995)|Action|Crime|Thriller|imdb top			
6	Sabrina (1995)	114319	11860	Comedy|Romance	remake|chauffeur|fusion|	Harrison Ford|Julia Orm	Sydney Pollack	Sabrina (1995)|Comedy|Romance|remake|c			
7	Tom and Huck (1995)	112302	45325	Adventure|Children	based on a book|Mark '	Jonathan Taylor Thoma	Peter Hewitt	Tom and Huck (1995)|Adventure|Children|			
8	Sudden Death (1995)	114576	9091	Action	explosive|hostage|terrori	Jean-Claude Van Damm	Peter Hyams	Sudden Death (1995)|Action|explosive|hos			
9	GoldenEye (1995)	113189	710	Action|Adventure|Thriller	007|Bond|boys with toys	Pierce Brosnan|Sean Be	Martin Campbell	GoldenEye (1995)|Action|Adventure|Thrille			
10	American President, The (1995)	112346	9087	Comedy|Drama|Romance	Romance|white house|ne	Michael Douglas|Annett	Rob Reiner	American President, The (1995)|Comedy|D			
11	Dracula: Dead and Loving It (1995)	112896	12110	Comedy|Horror	dracula|spoof|Mel Brook	Leslie Nielsen|Mel Broo	Mel Brooks	Dracula: Dead and Loving It (1995)|Comec			
12	Balto (1995)	112453	21032	Adventure|Animation|Children	Ei muista|alaska|bear att	Kevin Bacon|Bob Hoskii	Simon Wells	Balto (1995)|Adventure|Animation|Children			
13	Nixon (1995)	113987	10858	Drama	biography|government|h	Anthony Hopkins|Joan	Oliver Stone	Nixon (1995)|Drama|biography|governmer			
14	Cutthroat Island (1995)	112760	1408	Action|Adventure|Romance	exotic island|map|pirate|	Geena Davis|Matthew N	Renny Harlin	Cutthroat Island (1995)|Action|Adventure|F			
15	Casino (1995)	112641	524	Crime|Drama	Mafia|Mafia|Martin Scors	Robert De Niro|Sharon	Martin Scorsese	Casino (1995)|Crime|Drama|Mafia|Mafia|Ma			
16	Sense and Sensibility (1995)	114388	4584	Drama|Romance	chick flick|British|Jane Au	Kate Winslet|Emma Tho	Ang Lee	Sense and Sensibility (1995)|Drama|Romar			
17	Four Rooms (1995)	113101	5	Comedy	anthology|dark comedy|	Tim Roth|Antonio Band	Allison Anders|Alexa	Four Rooms (1995)|Comedy|anthology|dar			
18	Ace Ventura: When Nature Calls (1995)	112281	9273	Comedy	detective|childhood class	Jim Carrey|Ian McNeice	Steve Oedekerk	Ace Ventura: When Nature Calls (1995)|Cc			
19	Money Train (1995)	113845	11517	Action|Comedy|Crime|Drama|Thriller	new york city|new york	Wesley Snipes|Woody H	Joseph Ruben	Money Train (1995)|Action|Comedy|Crime|			

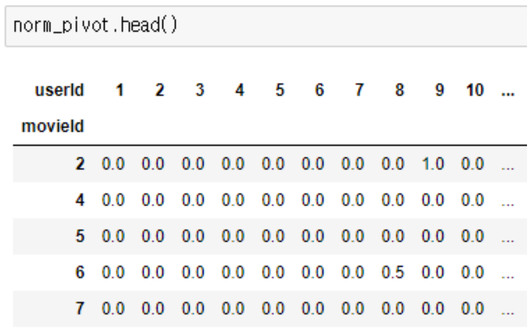
movie info data which contains the title, director, genre, etc. (output.csv file)



	A	B	C
1	userId	movieId	rating
2	1	306	3.5
3	1	307	5
4	1	899	3.5
5	1	1088	4
6	1	1175	3.5
7	1	1217	3.5
8	1	1237	5
9	1	1250	4
10	1	1260	3.5
11	1	2011	2.5
12	1	2012	2.5
13	1	2161	3.5
14	1	2351	4.5

rating data (ratings.csv file)

We loaded two data files as a Python pandas data frame and merged them with the *movieId* column. Then we pivoted the merged data frame and normalized the values by min-max normalizer.



```
norm_pivot.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...
movieId											
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.0	...
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

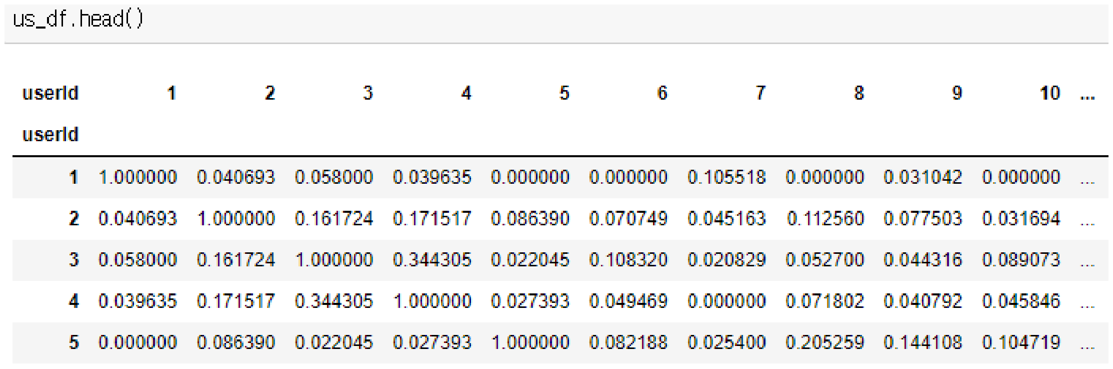
normalized pivoted table (userId, movidId)

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

Normalizer equation[2]

[2] https://www.digitalocean.com/community/tutorials/normalize-data-in-python

As known, the collaborative filtering model has a drawback in that it calculates the similarity by matrix factorization. To facilitate the process, we transformed the sparse pivoted table into a compressed sparse row by *scipy.sparse.csr_matrix()*[3] of the Scipy module. Then we produced both item similarity and user similarity tables by the *sklearn.metrics.pairwise.cosine_similarity()* function.



```
us_df.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...
userId											
1	1.000000	0.040693	0.058000	0.039635	0.000000	0.000000	0.105518	0.000000	0.031042	0.000000	...
2	0.040693	1.000000	0.161724	0.171517	0.086390	0.070749	0.045163	0.112560	0.077503	0.031694	...
3	0.058000	0.161724	1.000000	0.344305	0.022045	0.108320	0.020829	0.052700	0.044316	0.089073	...
4	0.039635	0.171517	0.344305	1.000000	0.027393	0.049469	0.000000	0.071802	0.040792	0.045846	...
5	0.000000	0.086390	0.022045	0.027393	1.000000	0.082188	0.025400	0.205259	0.144108	0.104719	...

user similarity table

Finally, we could build the collaborative filtering model with the similarity tables. The imported libraries we used for building the CF models are given as the following code snippet:

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity
import operator
```

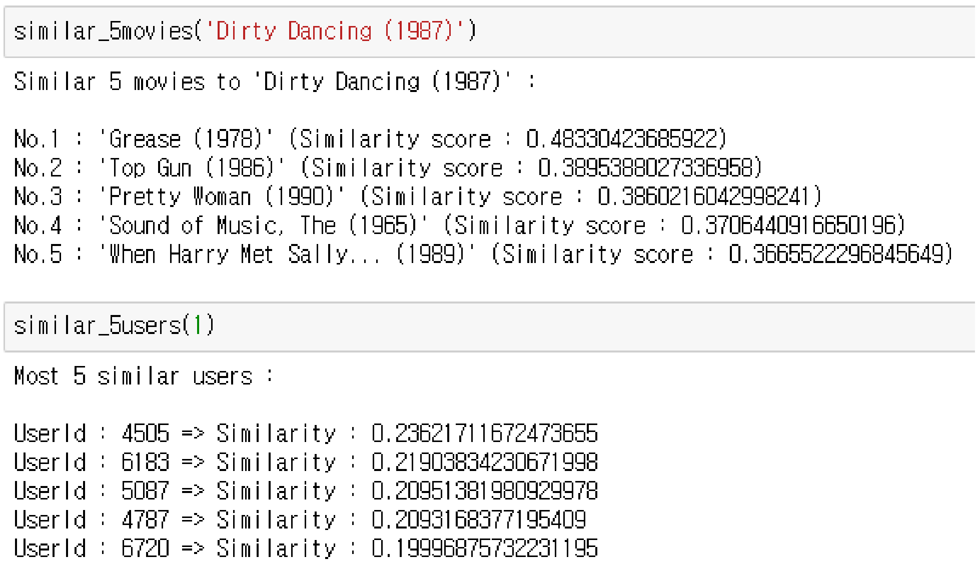
In the process, we defined several utility functions: a function that matches *movieId* and *title*, and a function that returns a list of movies and users having high similarity with the given input.

```
def find_id(title):
   id = int(match_df[match_df['title'] == title].movieId)
   return id

def find_title(id):
   title = match_df[match_df['movieId'] == id].title.values[0]
   return title

def similar_5movies(title):
   movie = find_id(title)
   num = 1
   print(f"Similar 5 movies to '{title}' : \n")
```

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

```
    top_five = is_df[movie].sort_values(ascending=False)[1:6]
    for item, score in top_five.items():
        title = find_title(item)
        print(f"No.{num} : '{title}' (Similarity score: {score})")
        num += 1

def similar_5users(user):
    if user not in norm_pivot.columns:
        return('No data of user {}'.format(user))
    print('Most 5 similar users: \n')
    top_five = us_df.sort_values(by=user, ascending=False).loc[:, user][1:6]
    for user, similarity in top_five.items():
        print(f"UserId: {user} => Similarity: {similarity}")
```

The result of user-based and item-based collaborative filtering is given below:



```
similar_5movies('Dirty Dancing (1987)')
```

```
Similar 5 movies to 'Dirty Dancing (1987)' :

No.1 : 'Grease (1978)' (Similarity score : 0.48330423685922)
No.2 : 'Top Gun (1986)' (Similarity score : 0.38953880273369958)
No.3 : 'Pretty Woman (1990)' (Similarity score : 0.3860216042998241)
No.4 : 'Sound of Music, The (1965)' (Similarity score : 0.37064409166650196)
No.5 : 'When Harry Met Sally... (1989)' (Similarity score : 0.3665522296845649)
```

```
similar_5users(1)
```

```
Most 5 similar users :

UserId : 4505 => Similarity : 0.23621711672473655
UserId : 6183 => Similarity : 0.21903834230671998
UserId : 5087 => Similarity : 0.20951381980929978
UserId : 4787 => Similarity : 0.2093168377195409
UserId : 6720 => Similarity : 0.19996875732231195
```

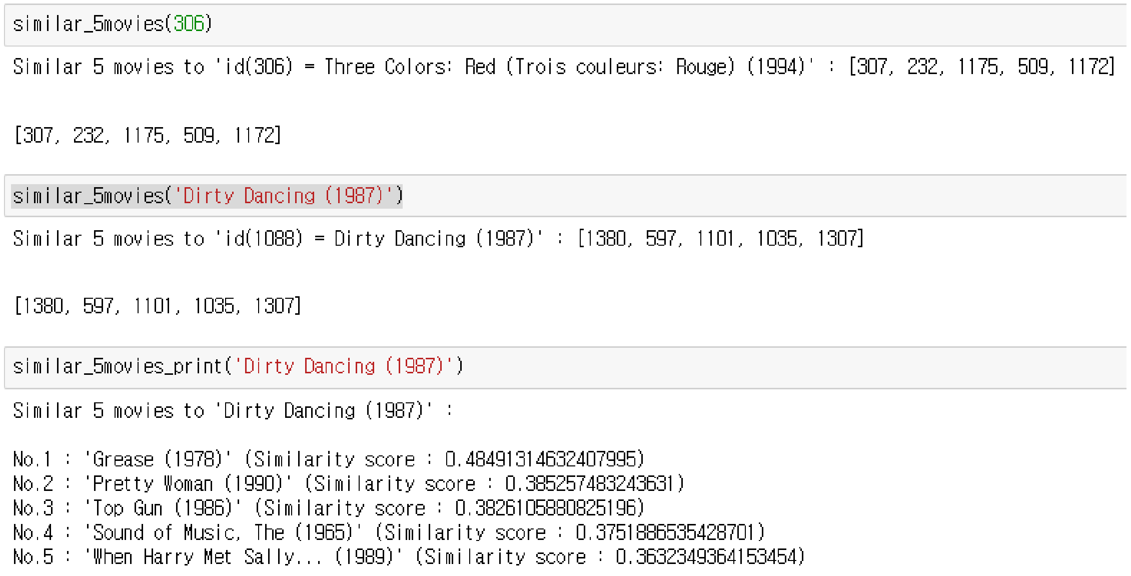
The *similar_5movies()* function is item-based collaborative filtering and the *similar_5users()* function is user-based collaborative filtering. We decided to utilize item-based collaborative filtering for movie recommendation because the model has to return recommendable *movieIds* based on the given input of the search model.

Originally, the function just printed 5 recommendable movies as described in the progress report. However, we modified the function result to return a list of 5 *movieIds* so that it can be passed to the processor and shown on the result page. Also, by utilizing *if* condition and *find_id()* functions, we enabled the *similar_5movies()* function to be able to serve both *movieId* (integer type) and *movieTitle* (string type) as input.

Modified function code

```
def similar_5movies(title):
   if type(id) != int:
       id = find_id(id)
   title = find_title(id)
   movie_list = []
   top_five = is_df[movie].sort_values(ascending=False)[1:6]
   for item, score in top_five.items():
       movie_list.append(item)
   print(f"similar 5 movies to \`id({id}) = {title}\` : {movie_list}\n’)
   return movie_list
```

Below is an example where *movieId* 306 was given as input and 5 movies were recommended in *movieId* format. As described above, this function also works when the string type movie title is passed to the parameter, where the example is given with 'Dirty Dancing (1987)'.



```
similar_5movies(306)

Similar 5 movies to 'id(306) = Three Colors: Red (Trois couleurs: Rouge) (1994)' : [307, 232, 1175, 509, 1172]


[307, 232, 1175, 509, 1172]

similar_5movies('Dirty Dancing (1987)')

Similar 5 movies to 'id(1088) = Dirty Dancing (1987)' : [1380, 597, 1101, 1035, 1307]


[1380, 597, 1101, 1035, 1307]

similar_5movies_print('Dirty Dancing (1987)')

Similar 5 movies to 'Dirty Dancing (1987)' :

No.1 : 'Grease (1978)' (Similarity score : 0.48491314632407995)
No.2 : 'Pretty Woman (1990)' (Similarity score : 0.385257483243631)
No.3 : 'Top Gun (1986)' (Similarity score : 0.3826105880825196)
No.4 : 'Sound of Music, The (1965)' (Similarity score : 0.3751886535428701)
No.5 : 'When Harry Met Sally... (1989)' (Similarity score : 0.3632349364153454)
```

As shown in the figure, item-based collaborative filtering, the *similar_5movies()* function returns the list of the top 5 most relevant *movieIds* arranged by the degree of relevance in descending order. Then this result is passed to the *processor* function.

* As an additional modification, because of the size limit by the public GitHub repository, we extracted the intermediate output as pickle files such as pivot_input.pickle and match_df.pickle.

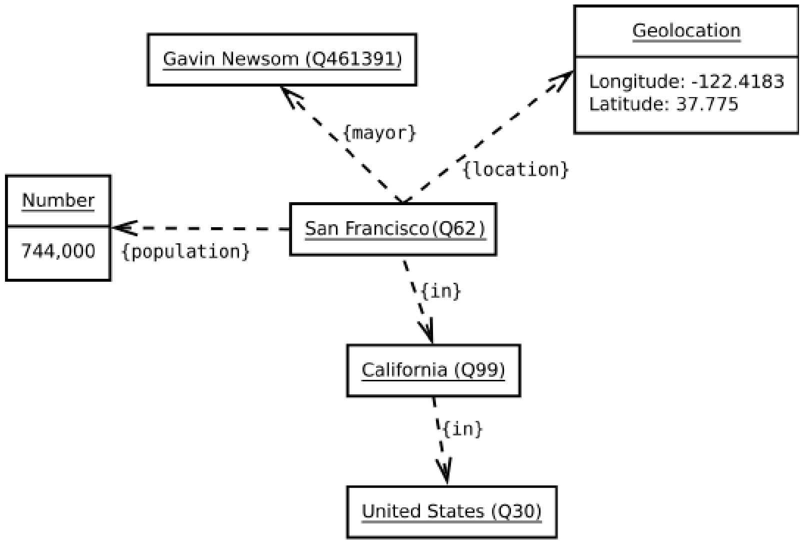
4.6. WikiData Knowledge Graph

Wikimedia provides a free and open knowledge base named WikiData[4]. WikiData is used as the main storage for Wikipedia, Wikivoyage, Wiktionary, Wikisource, etc. WikiData draws attention

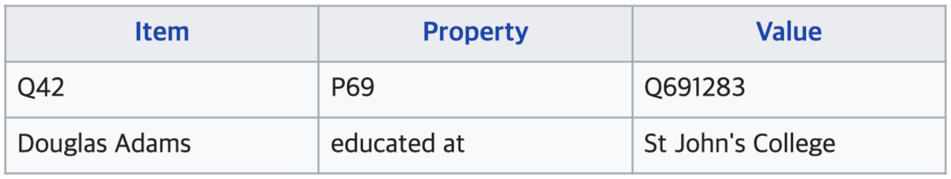
[4] https://www.wikidata.org/wiki/Wikidata:Main_Page

during the COVID-19 pandemic since it provides useful information from various articles and statistics. Not only WikiData put no request limit on the usage of its API, but the base provides its own query language to discover certain items and relations, unlike Google KG. Therefore, we chose WikiData to build our own knowledge graph for the recommendation system.

The figure below shows an example of WikiData.



As shown above, the repository mostly consists of items, which are identified by Q# labels, and a number of aliases. The item can be described in depth by Statement. Each statement has properties and values and each property can be identified by P# labels. The Statement can have a number of properties that elaborates on the corresponding items and the figure below shows an example. The property can be a pointer to an external database so the data type can be text, audio clip, photo, or even video. To get the item ID and property ID, we can refer to WikiData's wiki page. Here[5], they provide a website that shows all items and properties in document form.



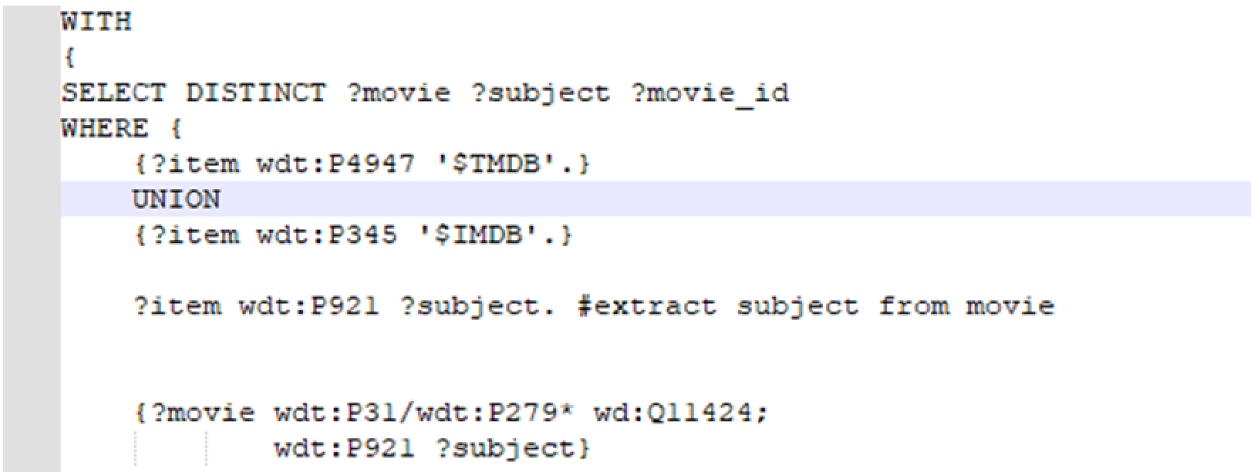
Item	Property	Value
Q42	P69	Q691283
Douglas Adams	educated at	St John's College

The most efficient way to extract information from WikiData is to run SPARQL queries using the pywikibot library in Python. Since there are more than 100 film-related properties in the database, we have conducted several tests to find the right balance between the complexity of our model and query run time. We ended up developing a script that takes movie id as an input and searches for related entities in the WikiData knowledge base with the following queries:

[5] https://www.wikidata.org/wiki/Q20856802 (item example link),
https://www.wikidata.org/wiki/Property:P69 (property example link)

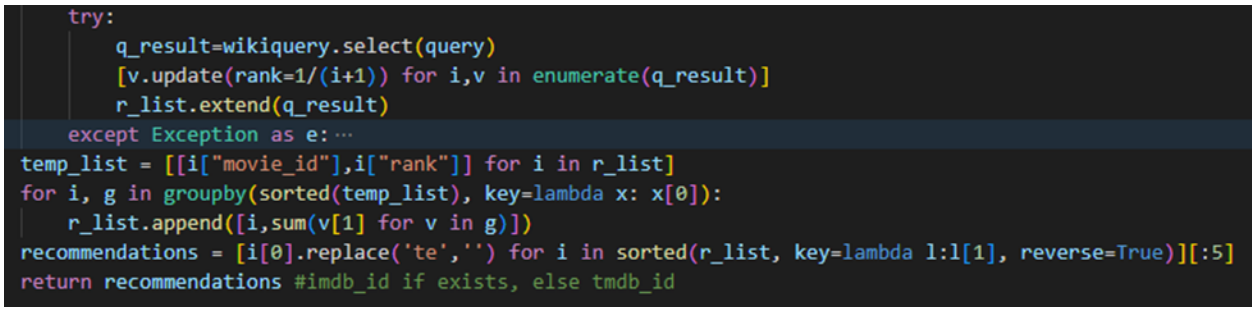
- Query #1 finds movies that share the same subject as input.
- Query #2 looks for movies with the same leading actor or director as the input. Since there isn't any field in WikiData that indicates who plays the main role, we defined it as the actor with the most movie awards..
- Query #3 locates movies containing the same character as input.

Here is a snippet of the first query; note that the results are sorted by the number of matched properties:



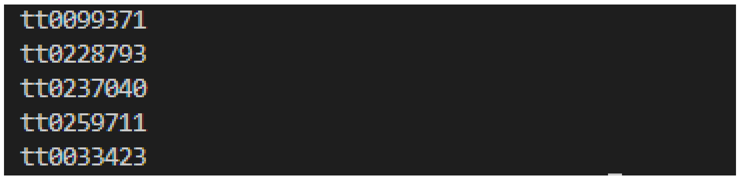
```
WITH
{
SELECT DISTINCT ?movie ?subject ?movie_id
WHERE {
    {?item wdt:P4947 '$TMDB'.}
    UNION
    {?item wdt:P345 '$IMDB'.}

    ?item wdt:P921 ?subject. #extract subject from movie


    {?movie wdt:P31/wdt:P279* wd:Q11424;
            wdt:P921 ?subject}
```

Since SPARQL does not support rank(), we have to create our own function by calculating the reciprocal of the position of each movie, that is, 1/r. The first element would have a reciprocal rank of 1/1 = 1. The second element would have a value of ½, and so on. As we need to concatenate results from all three queries, taking the reciprocal ensures films that are ranked high separately would also appear at the top of the combined list. We then select the top five movies and return their IMDB ids (TMDB ids if unavailable) as the final output to be parsed by the main processor.



```
    try:
        q_result=wikiquery.select(query)
        [v.update(rank=1/(i+1)) for i,v in enumerate(q_result)]
        r_list.extend(q_result)
    except Exception as e:···
temp_list = [[i["movie_id"],i["rank"]] for i in r_list]
for i, g in groupby(sorted(temp_list), key=lambda x: x[0]):
    r_list.append([i,sum(v[1] for v in g)])
recommendations = [i[0].replace('te','') for i in sorted(r_list, key=lambda l:l[1], reverse=True)][:5]
return recommendations #imdb_id if exists, else tmdb_id
```

Here is an example with *Top Gun* as the input:



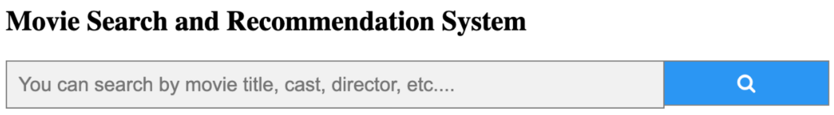
```
tt0099371
tt0228793
tt0237040
tt0259711
tt0033423
```

4.7. Web Application

We built a web app using HTML, CSS and Flask. It consists of two pages - a homepage (index.html) and a search results page (results.html) with recommendations. The Flask app is loaded by running *"python main.py"*:

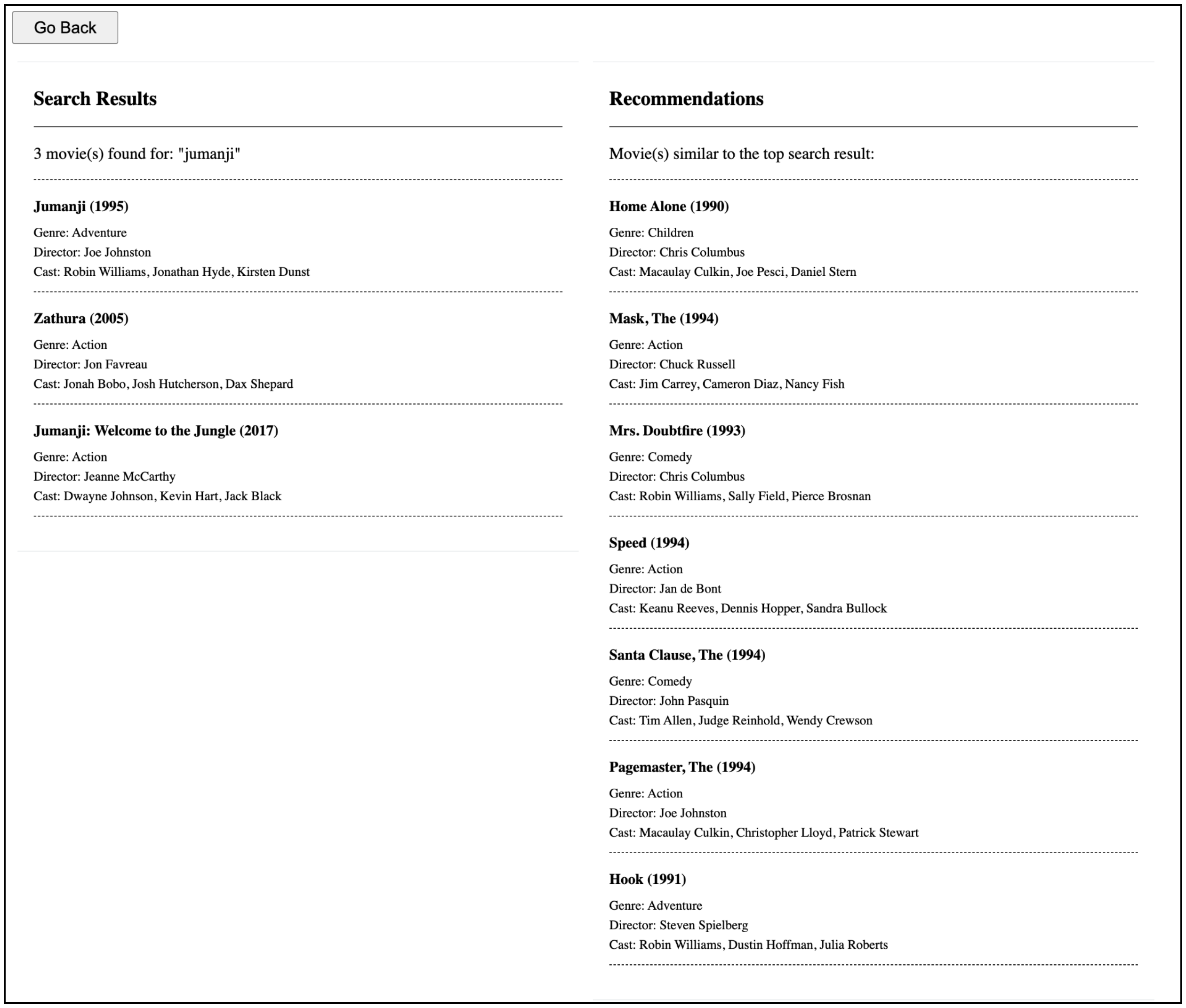
1. Upon loading, main.py renders the index.html file and displays the homepage.
2. When the user enters the search term and clicks the blue search button, the function will retrieve the search term and parse it to the back-end main processor function "processor.py".
3. When the processor returns the search and recommendation lists to the front-end, "main.py" will render the two lists in the "results.html" page and redirects the user to the results page.

The homepage is a simple search bar. The layout is similar to commercial search engines like Google so it's very intuitive for the user. The placeholder in the search bar is "You can search by movie title, cast, director, etc...." and acts as a line of instructions to the user. The user can enter the search term and click the blue search button to view the results.


Movie Search and Recommendation System
You can search by movie title, cast, director, etc....

The results page comprises of two lists side-by-side. The list of search results will be displayed on the left. The search will return up to 10 movies. The list of recommended movies, based on the top search result, will be displayed on the right. The list of recommendations will be kept at maximum 10 movies to prevent overwhelming the user. The user can also click the "Go Back" button on the top left of the page to return to the homepage and start a new search.

Results based on "jumanji":



Search Results

3 movie(s) found for: "jumanji"

Jumanji (1995)
Genre: Adventure
Director: Joe Johnston
Cast: Robin Williams, Jonathan Hyde, Kirsten Dunst

Zathura (2005)
Genre: Action
Director: Jon Favreau
Cast: Jonah Bobo, Josh Hutcherson, Dax Shepard

Jumanji: Welcome to the Jungle (2017)
Genre: Action
Director: Jeanne McCarthy
Cast: Dwayne Johnson, Kevin Hart, Jack Black

Recommendations

Movie(s) similar to the top search result:

Home Alone (1990)
Genre: Children
Director: Chris Columbus
Cast: Macaulay Culkin, Joe Pesci, Daniel Stern

Mask, The (1994)
Genre: Action
Director: Chuck Russell
Cast: Jim Carrey, Cameron Diaz, Nancy Fish

Mrs. Doubtfire (1993)
Genre: Comedy
Director: Chris Columbus
Cast: Robin Williams, Sally Field, Pierce Brosnan

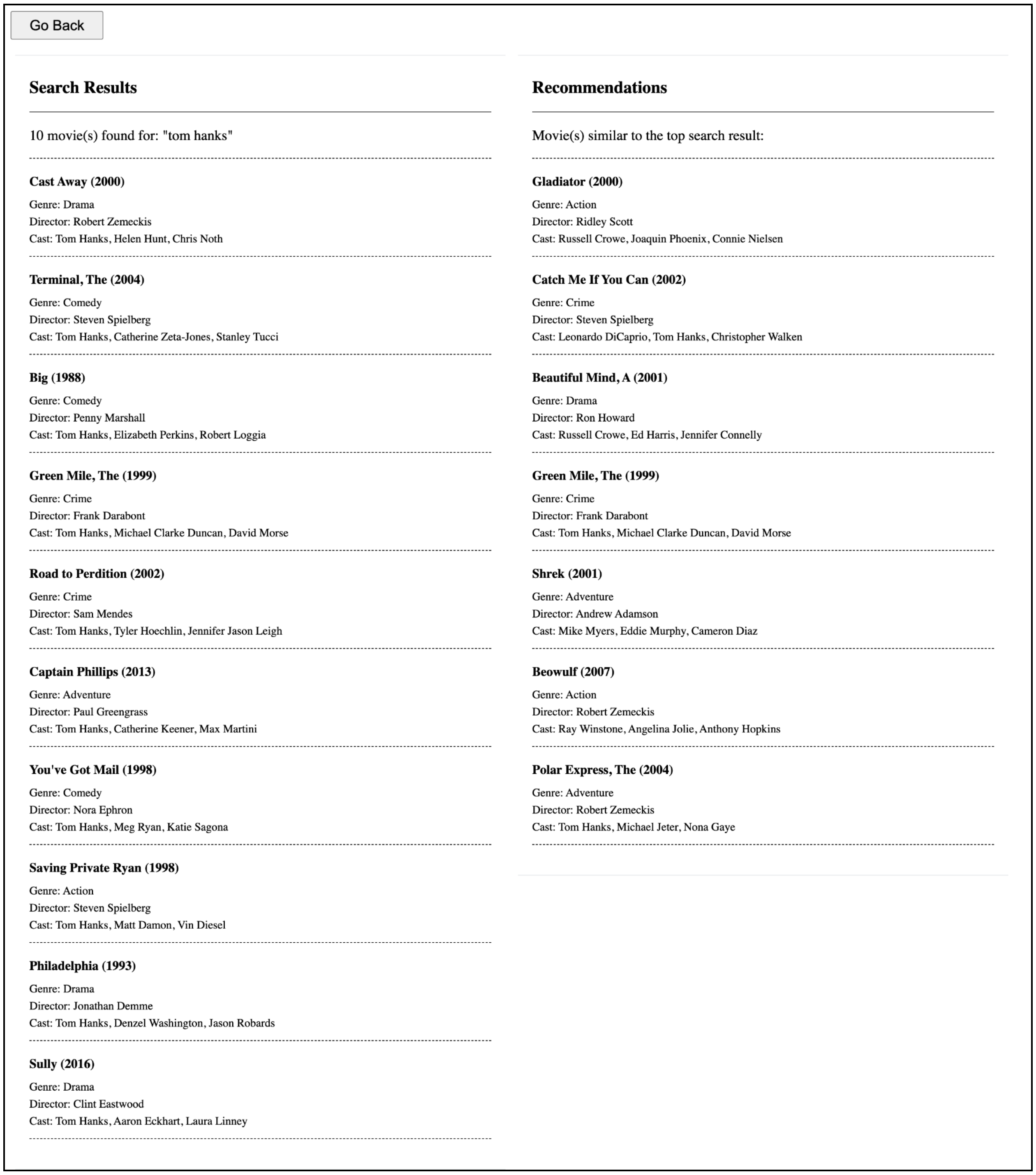
Speed (1994)
Genre: Action
Director: Jan de Bont
Cast: Keanu Reeves, Dennis Hopper, Sandra Bullock

Santa Clause, The (1994)
Genre: Comedy
Director: John Pasquin
Cast: Tim Allen, Judge Reinhold, Wendy Crewson

Pagemaster, The (1994)
Genre: Action
Director: Joe Johnston
Cast: Macaulay Culkin, Christopher Lloyd, Patrick Stewart

Hook (1991)
Genre: Adventure
Director: Steven Spielberg
Cast: Robin Williams, Dustin Hoffman, Julia Roberts

Results based on “tom hanks”:



Search Results

10 movie(s) found for: "tom hanks"

Cast Away (2000)
Genre: Drama
Director: Robert Zemeckis
Cast: Tom Hanks, Helen Hunt, Chris Noth

Terminal, The (2004)
Genre: Comedy
Director: Steven Spielberg
Cast: Tom Hanks, Catherine Zeta-Jones, Stanley Tucci

Big (1988)
Genre: Comedy
Director: Penny Marshall
Cast: Tom Hanks, Elizabeth Perkins, Robert Loggia

Green Mile, The (1999)
Genre: Crime
Director: Frank Darabont
Cast: Tom Hanks, Michael Clarke Duncan, David Morse

Road to Perdition (2002)
Genre: Crime
Director: Sam Mendes
Cast: Tom Hanks, Tyler Hoechlin, Jennifer Jason Leigh

Captain Phillips (2013)
Genre: Adventure
Director: Paul Greengrass
Cast: Tom Hanks, Catherine Keener, Max Martini

You've Got Mail (1998)
Genre: Comedy
Director: Nora Ephron
Cast: Tom Hanks, Meg Ryan, Katie Sagona

Saving Private Ryan (1998)
Genre: Action
Director: Steven Spielberg
Cast: Tom Hanks, Matt Damon, Vin Diesel

Philadelphia (1993)
Genre: Drama
Director: Jonathan Demme
Cast: Tom Hanks, Denzel Washington, Jason Robards

Sully (2016)
Genre: Drama
Director: Clint Eastwood
Cast: Tom Hanks, Aaron Eckhart, Laura Linney

Recommendations

Movie(s) similar to the top search result:

Gladiator (2000)
Genre: Action
Director: Ridley Scott
Cast: Russell Crowe, Joaquin Phoenix, Connie Nielsen

Catch Me If You Can (2002)
Genre: Crime
Director: Steven Spielberg
Cast: Leonardo DiCaprio, Tom Hanks, Christopher Walken

Beautiful Mind, A (2001)
Genre: Drama
Director: Ron Howard
Cast: Russell Crowe, Ed Harris, Jennifer Connelly

Green Mile, The (1999)
Genre: Crime
Director: Frank Darabont
Cast: Tom Hanks, Michael Clarke Duncan, David Morse

Shrek (2001)
Genre: Adventure
Director: Andrew Adamson
Cast: Mike Myers, Eddie Murphy, Cameron Diaz

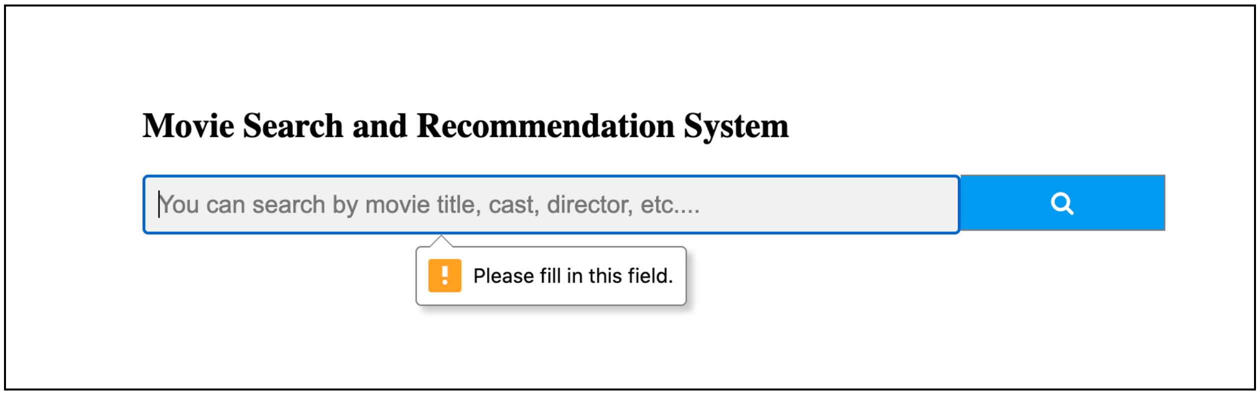
Beowulf (2007)
Genre: Action
Director: Robert Zemeckis
Cast: Ray Winstone, Angelina Jolie, Anthony Hopkins

Polar Express, The (2004)
Genre: Adventure
Director: Robert Zemeckis
Cast: Tom Hanks, Michael Jeter, Nona Gaye

4.8 Error Handling

We implemented basic error handling for our app.

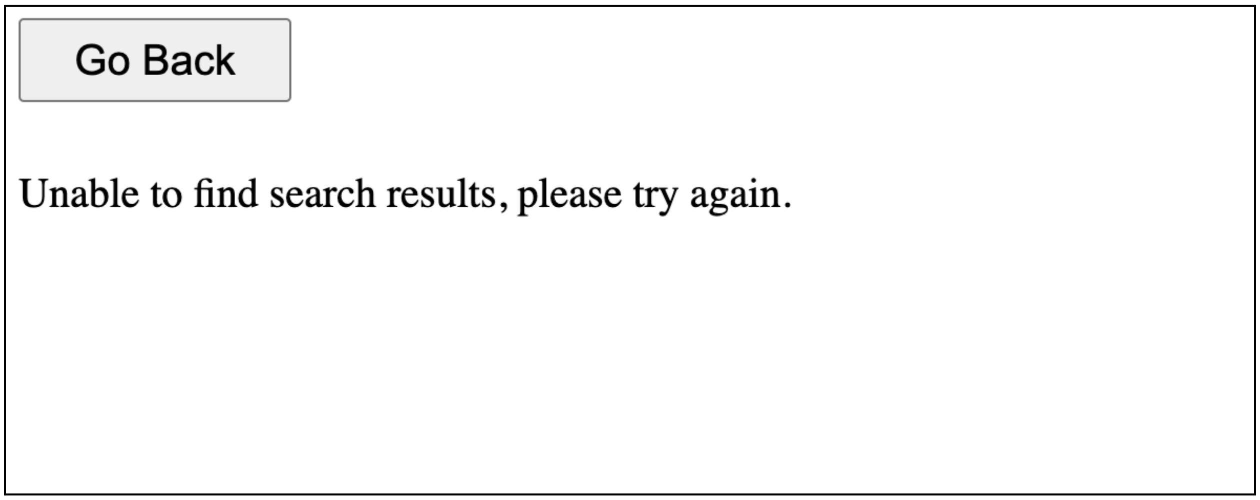
Firstly, it does not allow the user to enter an empty search query:


Movie Search and Recommendation System

You can search by movie title, cast, director, etc....

Please fill in this field.

Secondly, we have a generic error message for all types of errors. The user can click the "back" button to go back to the home page and enter a new search query.



Unable to find search results, please try again.

7. Challenges and Future Enhancements

We completed all the functionalities that were initially planned. We faced some challenges and had to come up with alternative solutions:

1. We initially wanted to host the app on Heroku, but Heroku free hosting ended in November 2022, so we hosted the app on the local environment.
2. We discovered that Metapy is not compatible with Flask, so we hosted the search function on an external server.

Future enhancements:

1. We can enhance our system as an interactive recommendation system by ensuring both recommendation models are updated seamlessly as the user gives ratings on the search result page. Currently, both models are independent of the user's preferences since they are trained separately. To serve the reciprocal system, we have to prepare scalable

computing resources which allow both models to be trained in real-time and serve the result to the front end with the least time lag.

2. Our project has limitations because of the Metapy library since it is supported on the older Python versions. We had to devise workarounds to ensure the Metapy functions will run as per normal, which adds unnecessary overheads, such as having a separate, remote search server. We chose Metapy since it was the main function taught and used in this course. We can explore a different library next time.
3. We implemented basic error handling and ensured our codes worked for generic use cases. If we had more time and if this was implemented in a larger-scale project, we can do more rigorous testing to ensure the search and recommendation quality.

8. Member Contribution

Member	Tasks
Hyeonjae Cho	Project Lead, Main Processor Function, Testing & Debugging
Xi Chen	Web User Interface, Flask App and FE/BE Integration, Testing & Debugging
Sean Kan	Data Preprocessing, Knowledge Graph, Testing & Debugging
Danwoo Park	BM25, Search Server, Testing & Debugging
Kihyuk Song	Collaborative Filtering, Testing & Debugging

Appendix

A. List of Packages

```
bottle==0.12.23
certifi==2022.9.24
charset-normalizer==2.1.1
click==8.1.3
Flask==2.2.2
idna==3.4
importlib-metadata==5.1.0
itsdangerous==2.1.2
Jinja2==3.1.2
joblib==1.2.0
MarkupSafe==2.1.1
metapy==0.2.13
```

```
mwparserfromhell==0.6.4
numpy==1.21.6
pandas==1.3.5
python-dateutil==2.8.2
pytz==2022.6
pywikibot==7.7.2
requests==2.28.1
scikit-learn==1.0.2
scipy==1.7.3
six==1.16.0
threadpoolctl==3.1.0
typing_extensions==4.4.0
urllib3==1.26.13
Werkzeug==2.2.2
zipp==3.11.0
```