CS410: Text Information System Project Proposal

Team ACT

This report describes the progress we have made for the team project in CS410 Text Information System course. We colored the completed tasks green in the chart below that describes the work breakdown that we planned in the proposal. The pending task that we are working on is colored yellow. The non-colored tasks are to-dos.

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	15 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)	10 hrs
Search Result Page - Develop a web page that gets input and returns corresponding results	15 hrs
Main Function - Integrate model outputs with webpage	5 hrs
Model Evaluation	10 hrs
Testing / Debugging	8 hrs
Demo Creation - Take a video that shows how our search result page and codes run behind	2 hrs
Total	100 hrs

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

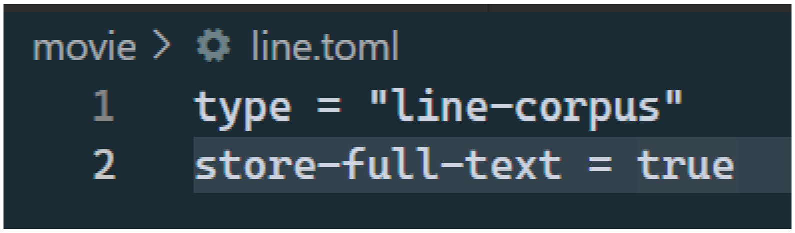
Since the MonvieLens25 dataset did not contain the names of cast and directors, the most challenging part of data preprocessing was finding the most effective way to obtain this data for nearly 60,000 movies. We originally planned to acquire them from IMDb, the world's largest movie database, but were 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.

2. 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. At this time, the search system is a cli-based and using OkAPI BM25 ranking algorithm. In the near future, the search engine will have a web user interface.

In the section above, we created a corpus.txt file that contains a list of movies with their meta data such as a 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.

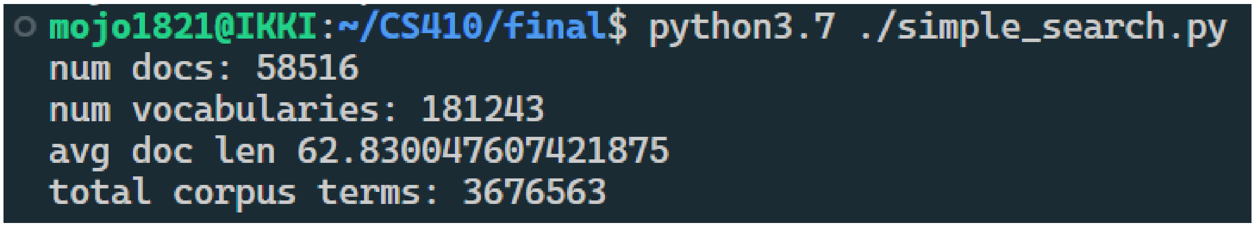
Since the MeTA toolkit supports creating an inverted index by config, we used the following setting to see the ranked results (list of movies found).



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

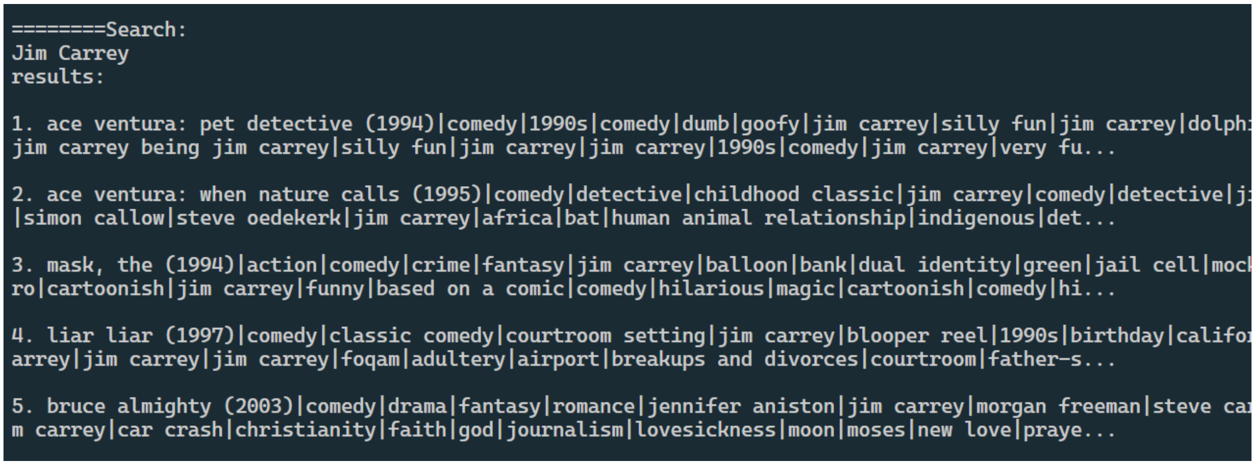
(a config file for creating an inverted index)

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

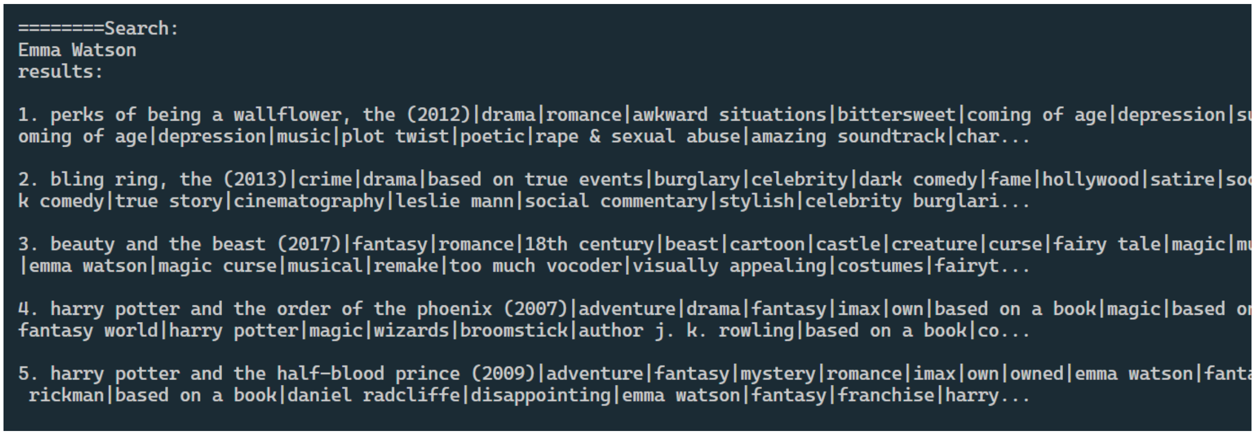


```
mojo1821@IKKI:~/CS410/final$ python3.7 ./simple_search.py
num docs: 58516
num vocabularies: 181243
avg doc len 62.830047607421875
total corpus terms: 3676563
```

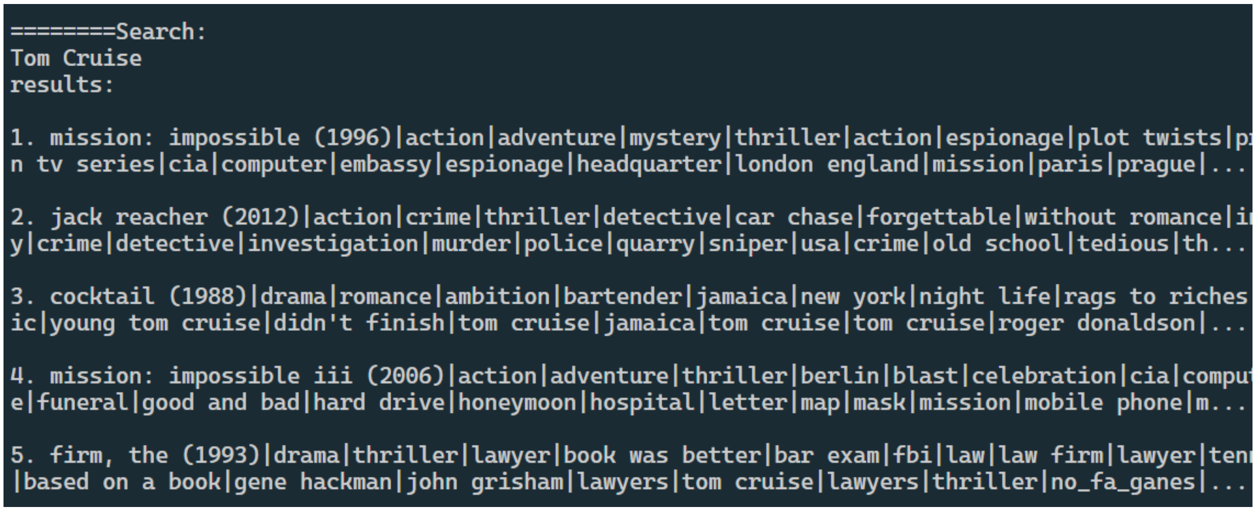
The following screenshots are a few examples of search queries and their corresponding results:



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

1. ace ventura: pet detective (1994)|comedy|1990s|comedy|dumb|goofy|jim carrey|silly fun|jim carrey|dolph
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|j
|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|califo
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 ca
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:
Tom Cruise
results:

1. mission: impossible (1996)|action|adventure|mystery|thriller|action|espionage|plot twists|pr
n tv series|cia|computer|embassy|espionage|headquarter|london england|mission|paris|prague|...

2. jack reacher (2012)|action|crime|thriller|detective|car chase|forgettable|without romance|in
y|crime|detective|investigation|murder|police|quarry|sniper|usa|crime|old school|tedious|th...

3. cocktail (1988)|drama|romance|ambition|bartender|jamaica|new york|night life|rags to riches
ic|young tom cruise|didn't finish|tom cruise|jamaica|tom cruise|tom cruise|roger donaldson|...

4. mission: impossible iii (2006)|action|adventure|thriller|berlin|blast|celebration|cia|comput
e|funeral|good and bad|hard drive|honeymoon|hospital|letter|map|mask|mission|mobile phone|m...

5. firm, the (1993)|drama|thriller|lawyer|book was better|bar exam|fbi|law|law firm|lawyer|tenn
|based on a book|gene hackman|john grisham|lawyers|tom cruise|lawyers|thriller|no_fa_ganes|...
```

3. 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*

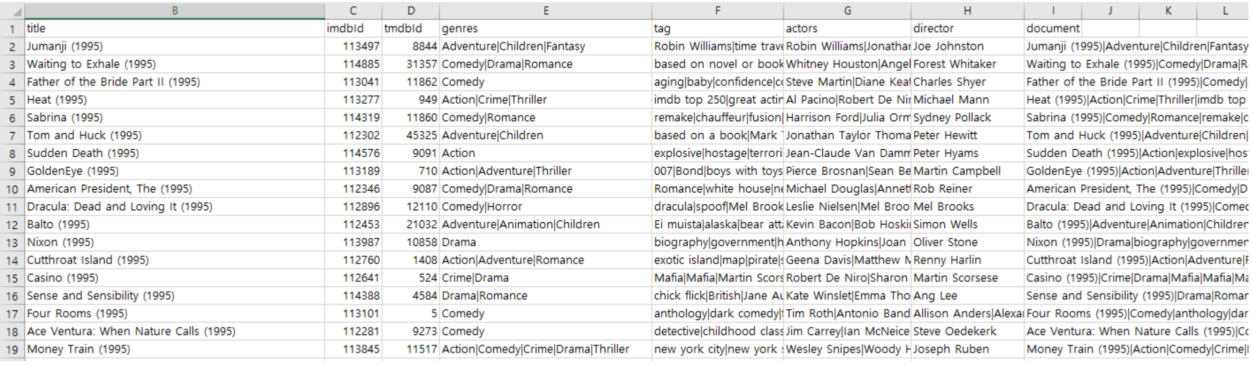
Based on the assumption, users sharing similar patterns in their preferences and interests can be identified based on their implicit feedback such as ratings.

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

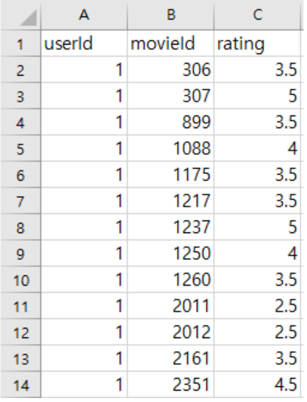
[1] https://en.wikipedia.org/wiki/Collaborative_filtering

We utilized two preprocessed data to build 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 trave	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 Hoski	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|governmen			
14	Cutthroat Island (1995)	112760	1408	Action|Adventure|Romance	exotic island|map|pirate|	Geena Davis|Matthew M	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|			

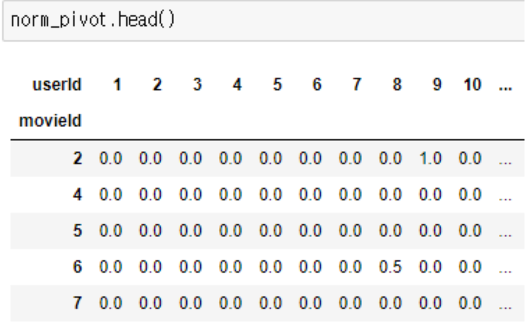
movie info data which contains the title, director, genre, etc.



	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

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	...

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

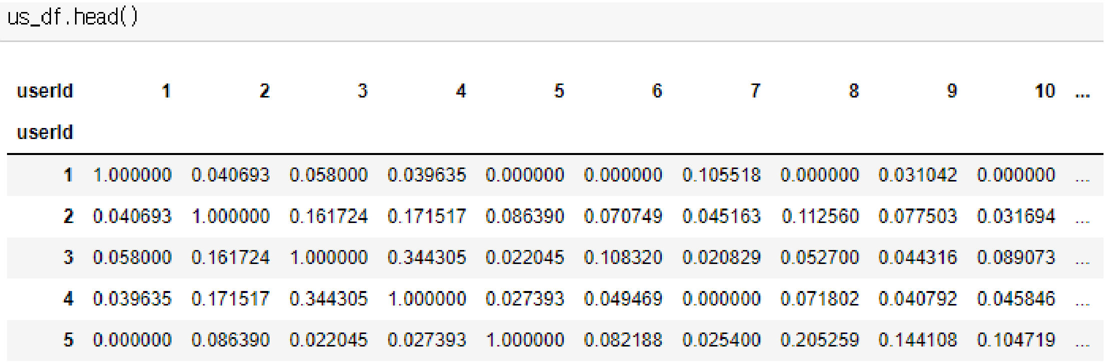
normalized pivoted table (userId, movidId)

Normalizer equation[2]

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

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

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
```

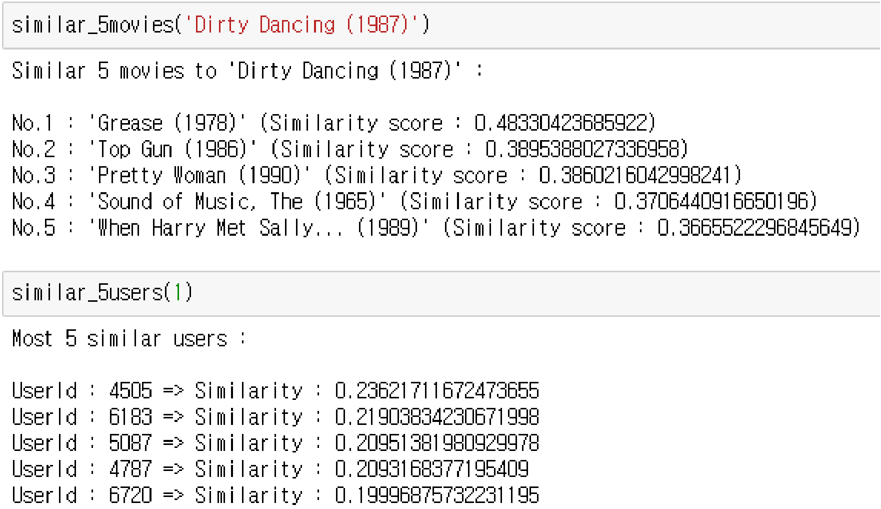
In the process, we defined several utility functions: a function that matches *movieId* and *title*, and a function that returns a list of movies and users having high similarity with the given input.

```
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")
   top_five = is_df[movie].sort_values(ascending=False)[1:6]
   for item, score in top_five.items():
```

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

```
        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 : 0.3895388027336958)
No.3 : 'Pretty Woman (1990)' (Similarity score : 0.3860216042998241)
No.4 : 'Sound of Music, The (1965)' (Similarity score : 0.3706440916650196)
No.5 : 'When Harry Met Sally... (1989)' (Similarity score : 0.3665522296845649)
```

```
similar_5users(1)
```

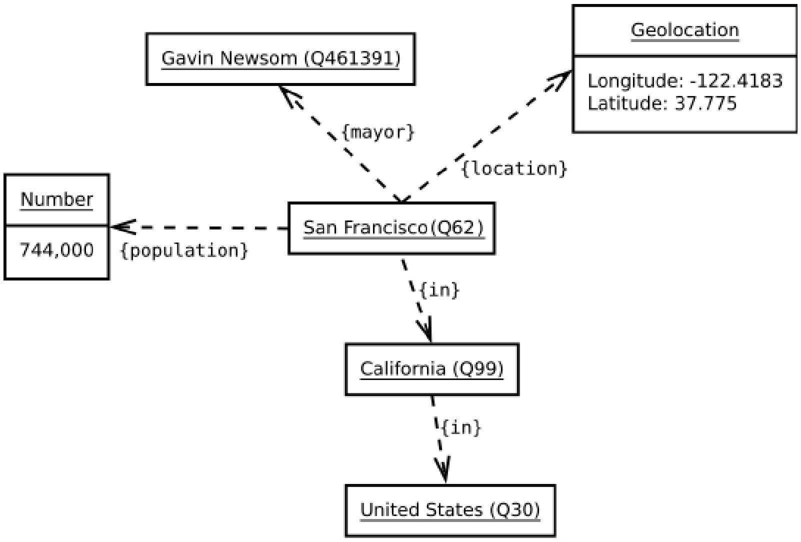
```
Most 5 similar users :

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
```

4. WikiData Knowledge Graph

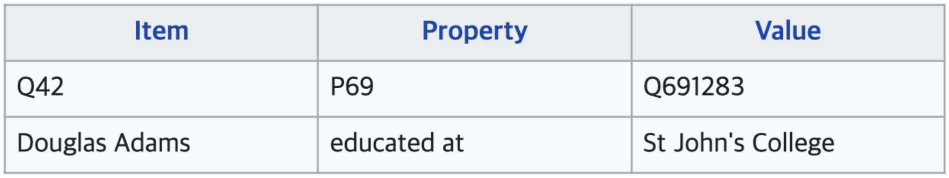
Wikimedia provides a free and open knowledge base named WikiData[4]. WikiData is used as the main storage for Wikipedia, Wikivoyage, Wiktionary, Wikisource, etc. WikiData draws attention during the COVID-19 pandemic since it provides useful information from various articles and statistics. Not only WikiData 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.

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



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

We can reach out to WikiData by using its API[6] or Query Service[7], which uses SPARQL, RDF query language. There are more than 100 properties for films available on Wikidata, which will greatly extend our database and enhance item-based collaborative filtering. For our project, we can consider the following properties (not limited to):

1. Film: Q11424, Short Film (Q24862), Television Film (Q506240), Anthology Film (Q336144)
2. Title: P1476, Publication Date (P577), Genre (P136), Director (P57), Cast Member (P161), Screenwrite (P58), Award Received (P166)

For example, the code snippet below shows the query that returns the romance film lists.

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

[6] https://www.mediawiki.org/wiki/API:Query

[7] https://query.wikidata.org

```
SELECT DISTINCT ?item ?itemLabel WHERE {
 SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE]". }
 {
   SELECT DISTINCT ?item WHERE {
     ?item p:P136 ?statement0. -- property: genre (P136)
     ?statement0 (ps:P136/(wdt:P279*)) wd:Q11424. -- value: film (Q11424)
     ?item p:P921 ?statement1. -- property: main subject (P921)
     ?statement1 (ps:P921/(wdt:P279*)) wd:Q1189047. -- value : romance (Q1189047)
   }
 }
}
```

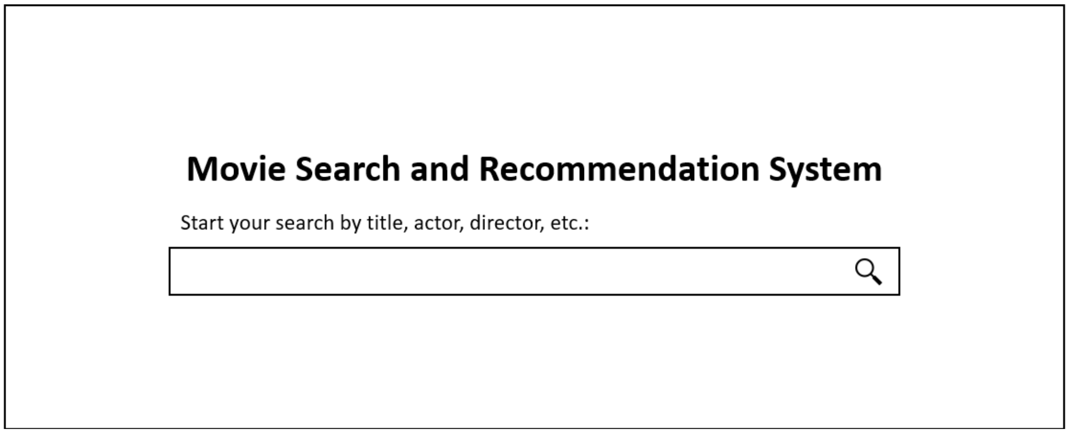
SPARQL only allows us to query the knowledge base by the property and property value. Therefore, we have to decide on which properties we should use to build our own knowledge graph for the recommendation system. Currently, we are planning to utilize genres, cast members, main subjects, directors, and character names as our properties. This is the most challenging part for us since none of our members have used knowledge graphs before. We should discover how to let the recommendation system utilize our own knowledge graph for better performance.

In the next few weeks, we would finalize developing our own knowledge graph and start to build the final recommendation system. Then we would try to make a search result page and evaluate our model performance by conducting tests involving real users.

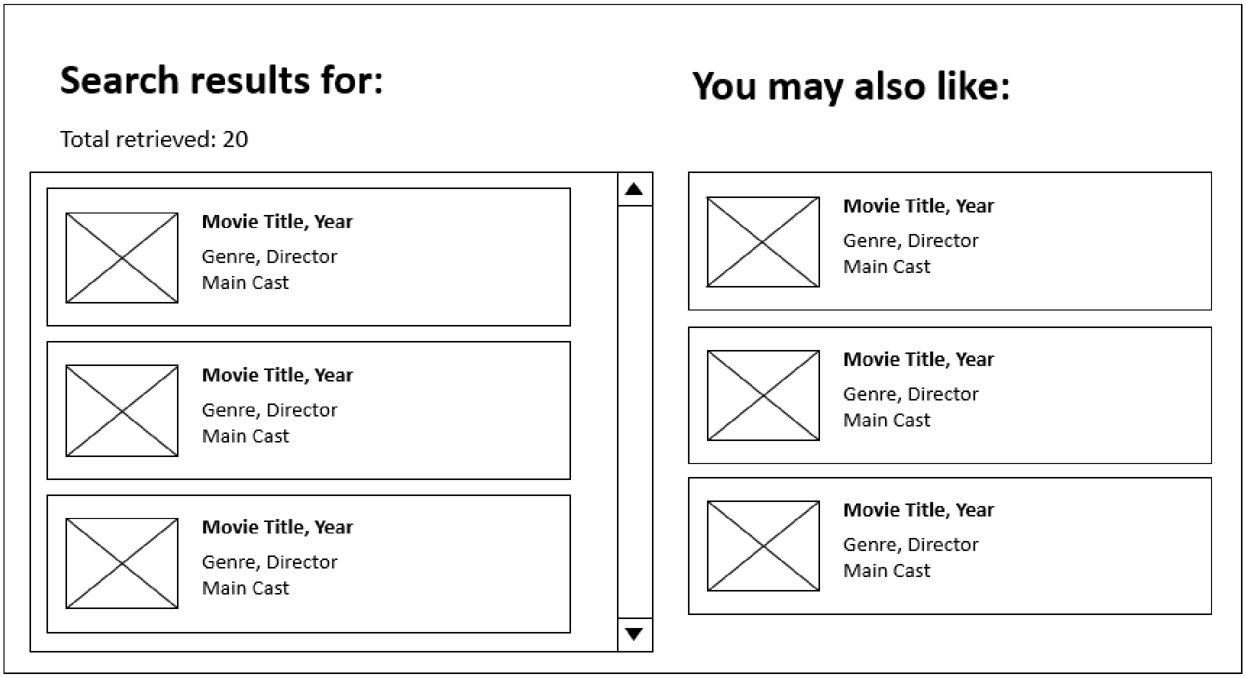
5. Web User Interface

We plan to build a web app using HTML, CSS and Flask and host it on Heroku, or any suitable cloud hosting platform. We envision it to be a simple search engine comprising two pages - a homepage and a search results page with recommendations. The layout and mode of interaction will be similar to commercial search engines that we are familiar with, like Google and Bing.

The wireframes below illustrates the user interface design. The homepage will be a simple search bar with a line of instructions for the user.


Movie Search and Recommendation System
Start your search by title, actor, director, etc.:

The search results page would comprise of two “containers” side-by-side, with the left container displaying the list of search results. It will be scrollable and paginated. The right container will display a list of movies similar to that of the search results (recommendations). The list of recommendations will be kept short, for example within 5 movies, to prevent overwhelming the user.



6. Challenges

The current plan is to host our website on Heroku, however Heroku is ending free hosting in end November 2022. We may need to host the website elsewhere (i.e. familiarising with a different environment), or continue using Heroku paid hosting.