Final Project Data Checkpoint

Jinyi Gao

Project code:

https://github.com/gaojinyigg/si507_final_project

Data sources:

One data is from https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata which is a related movie CSV dataset with 5000 movies information and the corresponding credits. In addition, another data source is an api which is used to catch some information on IMDB especially the rating on IMDB and the response is json format (url: http://www.omdbapi.com/?apikey=9da705df). Furthermore, as for every movie, itunes api is used to find the theme song of every movie if the theme song exists.

I access the csv data just directly by accessing the files and translating csv into json to maintain the consistency of all data. And meanwhile I keep part of the data as one my parameters for api requests, such as using title as a search parameter.

Moreover, I access another two data set from caches since sending request for every movie is too slow (sending a get request to the api and get response for the first time, then save them into the cache for accessing later).

As for the first data, attributes are as follows: *budget, genres, homepage, id, keywords, original_language, original_title, overview, popularity, production_companies, production_countries, release_date, revenue, runtime, spoken_languages, status, tagline, title, vote_average* and *vote_count.* Additionally, the second data from api is like figure 1 and itunes' data is just like those in project 1.

Firgue 1 attributes from api

For the cache part the evidence are as follows:

```python
def get_movie_to_cache(filepath):
    movies = read_csv_to_dicts(filepath, 'utf-8')
    res = []
    target = "http://www.omdbapi.com/?apikey=9da705df&s="
    for movie in movies:
        title = movie['title']
        temp = requests.get(target+title).json()
        if 'Error' not in temp.keys():
            # pprint.pprint(temp['Search'][0])
            res.append(temp)
    write_json("cache_data.json", res)

def detail_cache(filepath):
    detailed_cache = []
    with open(filepath, encoding='utf-8') as f:
        movies = json.load(f)
        for movie in movies:
            temp = movie['Search'][0]
            id = temp['imdbID']
            temp['details'] = requests.get("http://www.omdbapi.com/?apikey=9da705df&i="+id).json()
            detailed_cache.append(temp)
    write_json("cache_data_detailed.json", detailed_cache)
```

Figure 2 the functions implemented for caching

Figure 3 part of chunks of cache data

Data Structure:

I am going to use a graph to organize my data. Every movie will be a node. And the edge between nodes will depends on the several kinds of relationships. I am going to build three kinds of relationship among movies. First one is the genre. If movies have at least two same genres, they can be added as neighbors of both. Second one is the year, since different time always illustrate the development stage of movies, so I will link them up according to the time. For example, I can make movies become neighbors if the difference of their time is among 5 years. Third one is about lasting time. It is similar as the year. For example, I can make movies to be neighbors if the difference of their lasting time is less than 20mins.

Interaction and Presentation Plans:

The methods I used to present will depend on the situation. I will try to use flask to present my data. Since I have no opinion about it right now, so my back up plan will be the command line.

As for command line, I will send message to users for them to choose what kind of graphs they want to build and show different structures for them according to their choices. For different structure, I will try to plot them by some build-up functions such as plt.plot(). In addition, some data such as poster or music can be shown via open the corresponding link for the users to check. Furthermore, I will try to implement some search function. For example, I will provide the choice that whether the users only want the movies which IMDB score is over 8 or something else.