Description:

The goal of our project is to build a personalized ranking algorithm movie search engine by implementing what we have learned from 410. We are trying to combine search engine and recommender system in a way that when the user types a query, we will not only give the corresponding result, but also some relevant movies that we would like to recommend.

In order to do so, we used datasets provided by Kaggle (https://www.kaggle.com/ayushkalla1/rotten-tomatoes-movie-database ,https://www.kaggle.com/rpnuser8182/rotten-tomatoes) and combined them together. These datasets contained information as well as synopses for a huge set of movies that we used to build up our search engine. We merged information such as directors, actors, synopses etc. to generate our own corpus to be used in search engine (see corpus_generator.ipynb), so that we could support different kinds of queries. To do so, we also convert to text to vector form(Vector space model) and used some of the advanced ideas of TF-IDF to help compensate the effects of common words.

Besides, we wrote a scraper to get related pages on IMDB (https://www.imdb.com) and movies' customer ratings that we would push to users as supplementary content. (See imdb_scraper.ipynb.) We also used these ratings in our search engine rating model to include the factor of viewers' review for rating.

In terms of ranking functions, we have built two different versions. The first one was built based on the similarity to the combined corpus, which is pure texts, and we used the cosine similarity to define similarity function. The second one was an advanced version of the first one, and it combined the cosine similarity with the user rating to generate a

more indicative score. We believed that we wanted to not only provide the results that matched best, but also relevant movies that were worth watching, so the provided rating score would serve as extra information reference.

We have built a website as user interface so that a user can browse and enter a query to search a movie. For searching results, we also provide graphics that can illustrate the ranking results of our two models mentioned above.

Software Implementation:

We used HTML/CSS for decorating our website, and we use python for the backend logic. Specifically, we use Flask as our web Framework. We also use Pandas/numpy to deal/manipulate/clean data. We use Beautiful soup and request to help scrap the data. We use Sklearn library's TFIDF Vectorizer to help us vectorize the content of movies and we used its KNN to help us find the most 5 similar objects(cosine similarity). We use plotly to build all those plots. Finally, we use AWS Elastic Beanstalk to host our webiste.

Usage:

1.Launch a browser and navigate to http://cs410.pnxppjnsf2.us-west-1.elasticbeanstalk.com/

2.In the section below about, click one of posters, which corresponds to 9 different movie genres, to see recommendations of users' preferred genre, which are precomputed with our search engine and some editor's choice. You can further click "more about movie" to see more facts and visualization about the genre you choose.

3. Scroll down to section "enter keyword" or just click "search" on the top right corner, type in a query, and click "search". The results of related movies will be shown. If you click on the title button, it will navigate you to the related IMDB website. At the bottom of the result page are some graphics representing the scores of each search result.

Attributes:

In terms of contribution, Guangya Wan implemented the search engine model; Hongxuan Chen and Lujia Kang helped with raw data processing and filtering, building the scraper and fetching data from IMDB, and the documentation; Sizhi Tan developed the user interface website.