ProjectReport

November 21, 2020

0.1 Abstract

!! TODO

0.2 Introduction

Recommender systems are algorithms that attempts to predict user preferences. For example, recommender systems are used widely by services such as Spotify and Netflix to recommend songs and movies respectively to users, based on their past preferences in the respective domains. Recommender systems have grown significantly, and they are core parts of various online content providers to the extent that said content providers offer significant rewards for improvements to their algorithms. This can be seen in Netflix Prize.

0.2.1 Problem

The problem we tried to address was attempting to build a movie recommendation engine, using publicly available data.

0.2.2 Literature

!! TODO

0.2.3 New Idea

After learning about PageRank, and it's implementations, we attempted to use the power iteration method of calculating PageRank to attempt to build a movie recommendation system.

0.3 Method

0.3.1 Implementation Details

All the code for this project is written in Python 3.8.5. The libraries used in this project and their usages are as follows

Library	Usage	Link
`numpy==1.19.4`	Various mathematical applications, including handling of very large matrices, matrix multiplication, and various other matrix operations	https://numpy.org/
`scipy==1.3.3`	Used `scipy.optimize` to perform gradient descent and find optimal combination of weights for different similarity parameters	https://www.scipy.org/
`pandas==1.1.3`	For handling and processing of raw and processed data, in the form of `.csv` files	https://pandas.pydata.org/
`re`	Python Regular Expressions library for pattern matching while processing raw data	https://docs.python.org/3/library/re.html
`numba==0.51.2`	A Python compiler that allows for compiling a subset of python and numpy to machine code, used while testing methods and processing data to accelerate some computations	https://numba.pydata.org/
`graph-tool`	A Python library written in C++ for handling graphs, used while testing different methods of PageRank and methods of processing movie similarity. This was eventually not used in the final result	https://graph-tool.skewed.de/

0.3.2 Data Collection and Processing

The data for this project is all from publicly available sources. Our first goal was getting a comprehensive set of movies. Initially, we tried getting IMDb data, but realised this is only a very small and scattered subset. Eventually, we found and decided to use the GroupLens MovieLens 25M dataset, present in the `ml-25/` directory. This consists of a relatively small set of around 62k movies, but this was more than enough for our purposes. In addition to just movies, this dataset also included for each movie it's title, IMDb ID, year of release, genre(s), tags, and perhaps most importantly 25 million (hence the name) anonymised user ratings for the movies. For additional details about the specific data contained in this dataset, refer to `ml-25m/README.txt`.

For additional data about each movie, we built a web scraper that used the IMDb ID to go to the IMDb URL and extract the relevant information.

!! TODO Harsh

To clean the movie data and collect it into a proper format, `pandas` and `re` libraries were used to handle the csv files and process text patterns respectively. This process is present in `cleaned_movies_generator.py`. Additionally, most of the movies in the final dataset were irrelvant, primarily because they were of other languages. Additionally, we could not process the entirety of the data. For example, a single 62000×6200 matrix of half-precision (2 byte) floats that would represent similarity between movies would take up over `7GB` of space. To this end, movies were filtered to include only those in English, Hindi, Urdu and Punjabi. Additionally, there were some movies whose IMDb IDs were invalid (the webpage resulted in a 404 error while scraping) and these movies were dropped. This resulted in a set of 23843 movies, which are available in `processed_data/cleaned_subsetted_movies.csv`.

Additionally, `ml-25m/ratings.csv` contained in addition to userID, movieID, and rating, the timestamp for each rating. Since this was of no use to us, the column was dropped and the resulting dataset is in `ml-25m/timeless-ratings.csv`.

0.3.3 Approaches Used

Initially, we tried a graph-centric approach. In this method, the movies would be nodes on a weighted, undirected graph, and edges would indicate similar movies, the edge weight being the relative similarity for each movie. We chose `graph-tool` as our library of choice since in comparison to various other libraries, it came out significantly faster in various Benchmark tests.

The plan was to use various parameters of the collected data to create graphs that represent those similarities. For example, using the user ratings a graph was created (`graphs/ratings_graph.gt.xz`) in which users and movies were linked by edges weighted with the rating given to that movie by that user. This was intended to be used to calculate one similarity parametric between two movies, by using the fact that similar movies would have fewer edges connecting them, and these edges would have higher weight. This allowed us to then create a parametric, where the similarity between two movies ($S_{u,v}$) is directly proportional to the number of shortest paths ($N_{u,v}$) (in terms of number of edges) between two movies, directly proportional to the average weight along the paths ($A_{u,v}$), and inversely proportional to the number of edges in the paths ($E_{u,v}$). Mathematically,

$$S_{u,v} = \frac{N_{u,v} A_{u,v}}{E_{u,v}}$$

This approach quickly fell apart when we constructed the graph, which turned out to be extremely slow to process, and impossible to run many centrality or relatedness algorithms on simply due to the large memory requirements.

[ ]: