Tag-Based Movie Recommendation System

April 30, 2022

Joshua Guillot - Jguil87@lsu.edu

Han Lee - hlee48@lsu.edu

Introduction

Motivation

- We both agreed we wanted to create our project on something that a vast majority of people can understand. Out of a variety of topics, we both enjoy movies, so decided to do something based on movies. There are several streaming services available to the public like YouTube, Netflix, Disney+, etc. Although they all have recommendation systems, we wanted to attempt to compete with them.

Project Description

- The project processes data that includes movies and tags. Tags can consist of genres (e.g. action), awards (e.g. Oscar - Best Foreign Film), and even actors/actresses (e.g. Tom Cruise). It'll then take a user's input, and recommend a list of movies based on that input. The input is however many movies the user wishes to base the recommendation off of, and the recommended list based on tags.

Division of Roles

- Final Report - Joshua
- Final Report (editing) - Han
- Coding - Joshua, Han
- Midterm Report - Han
- Midterm Report (editing) - Joshua

System Design

Chosen Big Data Technologies

- Hadoop; Python - NLTK, Gensim
- We originally picked Hadoop and Java as that's what we're familiar with, but we realized that other frameworks would be better suited for this data processing and

recommendation. Python allows for various libraries and frameworks that are useful for mathematical calculations and data, as demonstrated in this project.

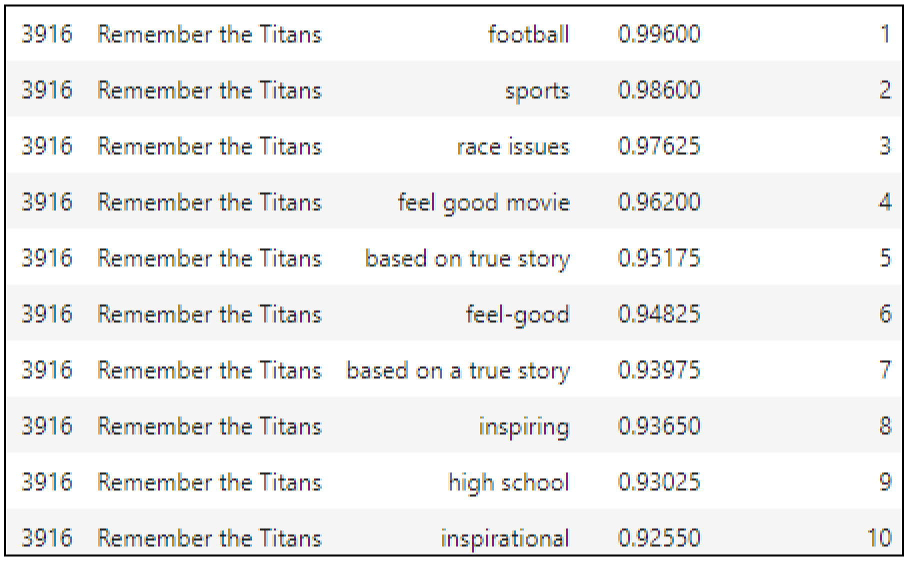
Chosen Datasets

- MovieLens Tag Genome Dataset 2021
- https://grouplens.org/datasets/movielens/tag-genome-2021/
- Download link: https://1drv.ms/x/s!AsJ09lgh4to5yVoqLO9JWMVtxcx9?e=Wgb7Jm

Detailed Description of Components

Each Component Description

- The project is a single file.
- There are hundreds of tags for each movie, and there are thousands of movies in the dataset. We process this data by utilizing MapReduce in Hadoopto isolate the top 50 most relevant tags per movie. We use these tags to help produce the model we will use in our recommendation system
- The program initially determines and condenses the number of desired tags for each movie that's processed. The dataset provides the relevance for each tag to a movie on a scale of 0-1. There are potentially hundreds of tags provided for each movie, many of which prove to be insignificant. We chose to pick the top 50 tags for the movies as there is a noticeable plateau in relevance around the 50th tags for all movies.



3916	Remember the Titans	football	0.99600	1
3916	Remember the Titans	sports	0.98600	2
3916	Remember the Titans	race issues	0.97625	3
3916	Remember the Titans	feel good movie	0.96200	4
3916	Remember the Titans	based on true story	0.95175	5
3916	Remember the Titans	feel-good	0.94825	6
3916	Remember the Titans	based on a true story	0.93975	7
3916	Remember the Titans	inspiring	0.93650	8
3916	Remember the Titans	high school	0.93025	9
3916	Remember the Titans	inspirational	0.92550	10

After filtering out irrelevant tags, the program measures the similarity of tags and therefore the similarity between movies by using cosine similarity. This process requires all the movies to be represented as vectors, or a set of three numbers. Movies would inherently be similar if their numbers in the corresponding dimensions were similar (e.g. .9 and 1.1 in the first dimension for two movies). We use doc2vec for the model and the vectors. It needed to be trained, then was able to convert documents into vectors. Cosine similarity is then used to quantify the similarity between two or more vectors. Cosine similarity is calculated by finding the cosine of the angle between vectors.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

After these calculations, the program uses the similarities found to generate a final list of movie recommendations. The simplest way of calculating the results is by finding the average of the vectors of the movies that the user has watched. With that mean vector, the program can use cosine similarity and provide a list of recommendations.

Evaluation & Test

Software Manual

- Type in the names of movies you wish to get recommendations for in the list “user_movies”, then run the program. Currently, the program automatically selects 3 movies at random for testing purposes.

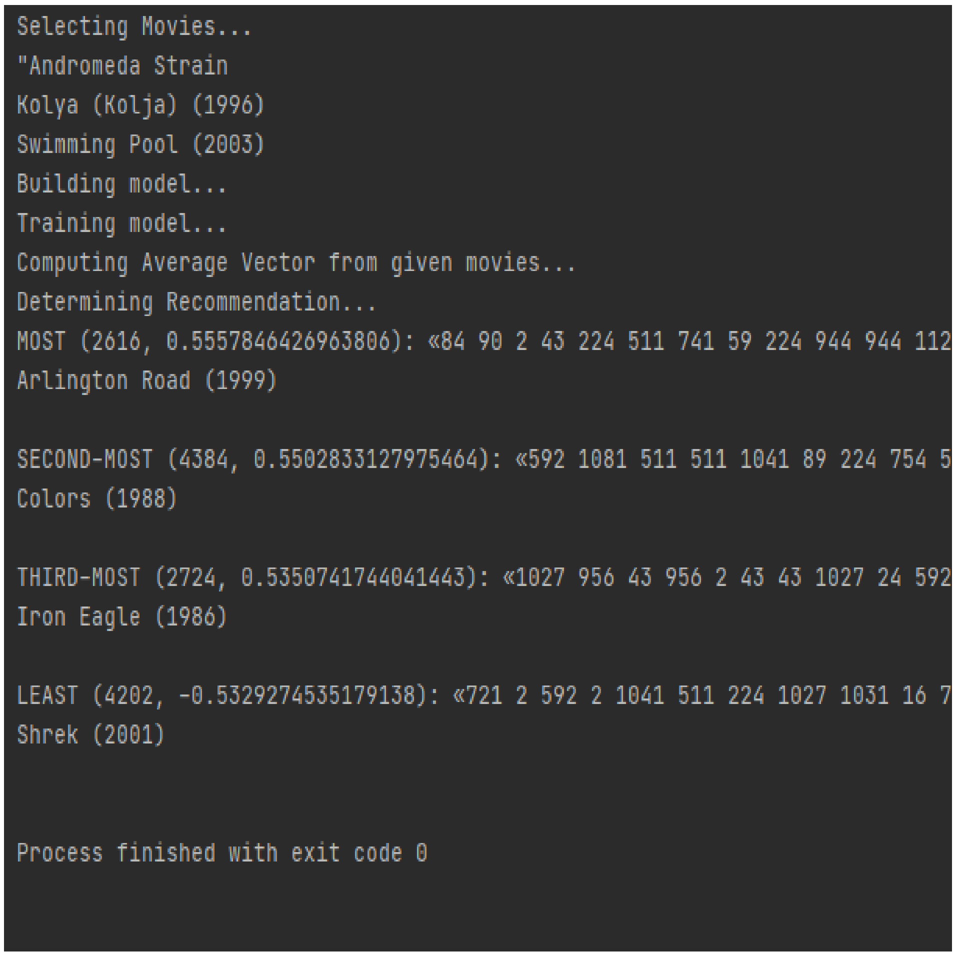
Evaluation/Test Environment

- The required dependencies for the correct environment are that the python program needs access to both movies.txt and myfile.

Evaluation/Test Results

- Inputting the movies:
    - Andromeda Strain
    - Kolya (Kolja) (1996)
    - Swimming Pool (2003)
- Results in this list of movie recommendations:
    - Arlington Road (1999)
    - Colors (1988)
    - Iron Eagle (1986)

Some screenshots (or videos) that show the working of your application



```
Selecting Movies...
"Andromeda Strain
Kolya (Kolja) (1996)
Swimming Pool (2003)
Building model...
Training model...
Computing Average Vector from given movies...
Determining Recommendation...
MOST (2616, 0.5557846426963806): «84 90 2 43 224 511 741 59 224 944 944 112
Arlington Road (1999)

SECOND-MOST (4384, 0.5502833127975464): «592 1081 511 511 1041 89 224 754 5
Colors (1988)

THIRD-MOST (2724, 0.5350741744041443): «1027 956 43 956 2 43 43 1027 24 592
Iron Eagle (1986)

LEAST (4202, -0.5329274535179138): «721 2 592 2 1041 511 224 1027 1031 16 7
Shrek (2001)


Process finished with exit code 0
```

Conclusion

- In conclusion, after inputting a movie or list of movies, the program will check the tags affiliated with those movies and output a curated list of movie recommendations with similarities to those tags based on average similarity. Utilizing the immense dataset available to us, we were able to avoid the ‘cold start problem’
- We learned a lot about movie recommendation systems, ways of measuring similarity, and vectors from this project. Python has several methods of visualizing data, which although isn’t necessary for this program, is very intriguing and useful for imagining the similarities between movies.

- A drawback to this program is it only works with movies that are in the provided dataset. This content-based filtering system we used is optimal for cold starts, where there is no initial data on the user. For future or more advanced systems, the software could utilize collaborative filtering or be hybrid. It could store all the data of the user's watched films and how satisfied they were with the recommendations. It could then build a more curated recommendation list. A future iteration could also implement average ratings (out of 5 stars) to build a list of well-reviewed movies.

Appendix

File name	# of lines
Main.py	50
MovieReducer	60
MovieMapper	20