Dhruv Gandhi

DS210 Final Project Report

5/4/23

In this project, I used Rust to analyze the MovieLens dataset from Kaggle (https://www.kaggle.com/datasets/grouplens/movielens-20m-dataset). I used the movie.csv and rating.csv files from this dataset for my analysis. The movie.csv file contains over 20,000 movies and includes the movieId, title, and genres. The rating.csv file contains over 20 million ratings for the movies in the movie.csv. This file has columns userId, movieId, rating, and timestamp. According to the authors of the dataset, all of the users have rated over 20 movies in this file.

The goal of analyzing this dataset is to find possible connections between certain movies based on the users that watch them. If successful, such a program could aid in recommending movies to certain users based on what other users have watched and rated in the past. In order to accomplish this goal, the following methods were completed in Rust.

First, both of the aforementioned csv files were loaded into Rust. Since the dataset is very large, I had to implement a for loop so that only the first 2,000 movies from the dataset were analyzed to evade issues with memory while running. The movies from movie.csv and the ratings from rating.csv were each assigned to their own respective vectors.

After both movie.csv and rating.csv were loaded and these vectors were created, I wrote a function that created a directed graph in which I added a node for each movie in the movie vector and edges for each rating in the ratings vector. This was achieved using the petgraph library (https://docs.rs/petgraph/latest/petgraph/), making it easier to create the desired graph. The function takes two arguments, movies and ratings, which are slices of tuples containing the movie ID, title, and rating data. The function returns a PetGraph object in which movies are source or target nodes connected by edges that are weighted by the ratings.

In order to find possible paths between the movies, a breadth-first search function was implemented. This function takes in a directed graph, a start node, and an end node and uses a

BFS iterator starting from the start node to walk through all nodes reachable from the start node in the graph. It also keeps track of the parent of each node that has been visited during the BFS using a vector called ”parents.” If the end node is reached during the BFS, the function backtracks from the end node to the start node by following the parent links to create the path and returns it as Some(path). If no path is found, it returns None.

I then analyzed the movie graph using the six degrees of separation technique. In the main function, a list of 20 random movies are looped over to find the shortest path between each pair of movies using the breadth-first search (BFS) algorithm. If a path is found, it is printed out. Otherwise, a message indicating that no path is found is printed.

Finally, a few conclusions can be made from the six degrees of separation analysis. First, since 20 movies are randomly chosen from the dataset, the output of the program will vary every time it is run. After running the program several times, however, it was observed that most movies don’t have a path of ratings connecting them to each other, and there are times when no path is found between any two of the 20 films. Yet, if a movie did have a path to another movie, it seemed to be more likely that it would have paths to multiple other movies as well.

While the output of this program is not always very telling, the applications of the program, if developed further, are very promising. The results would be far more comprehensive if the program is run on a device with more memory and the capability to analyze the entire dataset. Streaming services could then use such results in a collaborative filtering recommendation system based on the ratings of the users in this data.

I look forward to learning more about how I can further develop this program to reach this goal.