Thomas MacDougall
ITCS-6162
Programming Assignment

Movie Recommendation System Report

Movie recommendation systems are used to help users discover new movies that they might enjoy. As users utilize applications like Letterboxd and IMDb, these systems get to know the user's tastes and preferences more and more, and can make more informed recommendations based on this data. Keeping users engaged and using these applications is vital to the long-term success of the companies that run them, and so ensuring that users always have something new to watch is of natural importance. There are several different approaches to this, of course; user-based approaches compare users and their tastes in order to find similar users, while item-based approaches find movies similar to ones a given user has already liked. Random-walk approaches operate differently; they construct a graph of movies and users, and "walk" between nodes, counting the number of times a node has been visited, and makes recommendations based on that.

The dataset used for this project, MovieLens100k, is in three parts: users, ratings, and movies, with each part having several different features. The users dataset contains over 1000 datapoints, and compiles age, gender, and occupation, and assigns each user an ID. The movies dataset is similar, and describes over 1000 movies, with their titles and release dates, and also assigns each one an ID. Finally, the ratings dataset ties the two together, with almost 1000 datapoints describing a user and movie ID, and the rating that that user gave the movie (as well as a timestamp). Little preprocessing was performed, besides altering the timestamp format, and removing null datapoints from the movie dataset. As the dropped movie IDs are expected in the other datasets, care was taken later on to ensure that any pointers to dropped movies are skipped over and ignored.

Below are descriptions of the methodologies taken for each approach described above, as well as the steps taken to build each function.

User-Based Collaborative Filtering // Implementation Details

The idea here was to measure the similarity of users based on their movie ratings. This was performed by first finding the cosine similarity between users, using a user movie matrix (a reshaped ratings dataframe, where each row is a user, and each column is a movie). This matrix is naturally very sparse. The cosine similarity between each row was then stored in a separate dataframe.

The steps taken for the algorithm itself are quite typical of a user-based algorithm. Given a user, we retrieve any similar users (using the cosine similarity matrix), and get the movie ratings from these users. Then, we compute the average rating for each movie (weighted by the similarity score), and sort them in descending order. This sorted list of movies represent (on average) the top *num* movies the given user would likely enjoy.

Item-Based Collaborative Filtering // Implementation Details

Similar in concept to user-based filtering, we first calculate the cosine similarity between pairs of movies, based on their user ratings. However, unlike user-based, we first transpose the user movie matrix, as similarity must be between movies instead of users. Once similarity is calculated, it is once again stored in a separate list for later use.

Given a movie, the function then attempts to find the corresponding movie ID, and extract any similar movies from the similarity matrix. Then, those similar movies are sorted in descending order, and the top *num* are retrieved. Their names are mapped to the IDs, and returned as a list.

Random-walk-based Pixie Algorithm // Implementation Details

Visualizing these datasets as graphs is effective because they scale up so much better than other approaches. Essentially, given a random-walk algorithm, we can recommend nodes based on how many times they're visited by the random walker. Nodes with more in-links get recommended more than those with less, and similar nodes are close by; a properly constructed graph allows us to find users with similar tastes, and movies that are frequently watched together.

This graph was built from the dataset, with each individual movie and user being represented as nodes. Edges were drawn between nodes if a given user has rated a given movie. The algorithm itself is quite simple— given a user, at each step, it picks a random movie to walk to, keeping count of each individual movie it's visited. Then, it visits a user connected to that movie, and repeats a given number of times. The walker is likely to stay within a neighborhood of similar movies/users, thus making solid recommendations (based on how long the walk is). As we keep track of how many times each movie is visited, we can simply sort that list in descending order at the end, and return the top *n* as our recommendations.

Results and Evaluation

User-based recommendations:

```
#Testing
recommend_movies_for_user(10, num = 5)
```

Ranking	Movie Name
1	Great Day in Harlem, A (1994)
2	They Made Me a Criminal (1939)
3	Santa with Muscles (1996)
4	Someone Else's America (1995)
5	Entertaining Angels: The Dorothy Day Story (1996)

```
#Testing
recommend_movies_for_user(11, num = 6)
```

Ranking	Movie Name
1	They Made Me a Criminal (1939)
2	Prefontaine (1997)
3	Marlene Dietrich: Shadow and Light (1996)
4	Star Kid (1997)
5	Someone Else's America (1995)
6	Entertaining Angels: The Dorothy Day Story (1996)

With the user-based recommendations in particular, there appears to be some issue with taking the weighted sum of ratings, causing slight inaccuracies in the recommendations. However, recommendations still are consistent and at least somewhat accurate.

Item-based recommendations:

```
recommend_movies("Jurassic Park (1993)", num=5)
```

ranking	movie_name
1	Top Gun (1986)
2	Empire Strikes Back, The (1980)
3	Raiders of the Lost Ark (1981)
4	Indiana Jones and the Last Crusade (1989)
5	Speed (1994)

```
recommend_movies("Top Gun (1986)", num=5)
```

ranking	movie_name
1	Jurassic Park (1993)
2	True Lies (1994)
3	Batman (1989)
4	Die Hard: With a Vengeance (1995)
5	Speed (1994)

Item-based recommendations appear accurate and consistent. Similar movies from similar time periods are being recommended, and as shown, one movie recommended in one list ends up recommending the first when queried in turn.

Graph-based recommendations

```
weighted_pixie_recommend(1, walk_length=15, num=5)
```

15

ranking	movie_name
1	Clerks (1994)
2	Daytrippers, The (1996)
3	Copycat (1995)
4	Leaving Las Vegas (1995)
5	Old Yeller (1957)

```
weighted_pixie_recommend(1, walk_length=1500, num=5)
```

1499

ranking	movie_name
1	Ulee's Gold (1997)
2	Contact (1997)
3	Mission: Impossible (1996)
4	Jaws (1975)
5	Liar Liar (1997)

With small walks, recommendations are unstable and quite random, as the graphs are dense and it is difficult to get consistent results with such a low sample. However, increasing the walk length greatly improves the stability, providing consistent recommendations.

Conclusion

While traditional item and user-based recommendation systems are still fairly robust, graph-based methods seem quite strong in comparison. They're faster, more easily implemented, and are easier to implement. While not an issue in this case, the two former can struggle with smaller datasets, where users or items are lacking in numbers, but graph-based approaches with random walk algorithms sidestep this problem handily. Hybrid models of item and user systems are an obvious solution to that issue, though there is still uncertainty about whether a hybrid model would still beat out a graph-based method. In this case, all three approaches ran quickly and successfully, and could likely be implemented in IMDb (or similar platforms) with little difficulty. As stated in part one, a graph-based approach was once used in Pinterest to recommend content to users, and so graph-based approaches could likely be used in other social media platforms as well— for instance, Instagram could construct a graph of users and tags, and make recommendations to users in a similar fashion.