Thomas Sosa
Tomas Arenas
Zack Wittmann

Movie Recommendation System

Source code: https://github.com/Sosa714/movie-recommender
Source Code Note: Most current version is from thomas-branch

Project data note: We worked with the data by putting it all in a folder called 'data' that was in the same directory as the ipynb program file.

Some examples of our model doing predictions

```
Based on ['Pulp Fiction (1994)', 'Three Colors: Blue (Trois couleurs: Bleu) (1993)', 'Underground (1995)', 'Dirty Dancing
Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)
Garden State (2004)
Midnight in Paris (2011)

Based on ['Braveheart (1995)', 'Apollo 13 (1995)', 'Rob Roy (1995)', 'French Kiss (1995)', 'Star Wars: Episode IV - A New
Pirates of the Caribbean: The Curse of the Black Pearl (2003)
Back to the Future (1985)
Jewel of the Nile, The (1985)

Based on ['Toy Story (1995)', 'City of Lost Children, The (Cité des enfants perdus, La) (1995)', 'Twelve Monkeys (a.k.a. 1
Jewel of the Nile, The (1985)
Beastmaster, The (1982)
Saw V (2008)

Based on ['Pulp Fiction (1994)', 'Blade Runner (1982)', 'Terminator 2: Judgment Day (1991)', '2001: A Space Odyssey (1968)
Ghostbusters (a.k.a. Ghost Busters) (1984)
Pirates of the Caribbean: The Curse of the Black Pearl (2003)
Army of Darkness (1993)
```

Above is the output of the top 3 recommended movies for the first 4 users in the users_liked_genres&tags.csv. This was the first output we produced which is why it's a little unorganized.

```
Top 10 movie recommendations for User 52: ['The Leading Man (1996)' 'Ogre, The (Unhold, Der) (1996)'
 'Woman of Paris, A (1923)' 'Lost in Yonkers (1993)'
 'Mildred Pierce (2011)' 'The Missing (2014)'
 'The Disappearance of Eleanor Rigby: Him (2013)'
 'Southside With You (2016)' 'Perfectos desconocidos (2017)'
 'Braven (2018)']
```

Above are the top 10 word embeddings for user 52, who we chose to represent our model because of the screenshot below.

Thomas Sosa
Tomas Arenas
Zack Wittmann

userId	Most Frequent Genre/Tag	Bias Ratio
52	drama	0.923077
100	drama	0.731707
47	drama	0.600000
55	drama	0.574468
1	drama	0.543860

Above is the screenshot of the top 5 users(out of the first 100) with the most bias towards a genre. We used user 52. As you can see, they have a heavy preference toward ‘drama’, which are the types of (most) movies recommended.

userId	Comedy Count	Comedy Bias Ratio
1267	1	1.000000
3195	20	1.000000
1368	21	0.954545
2133	14	0.777778
1369	10	0.769231

Above is a screenshot that shows the users(includes all users this time) with the most liked comedies to other genres. If you look at user 1267 up in ratings.csv, he watched 4 movies, but the only movie they rated a 4.0 or above was a comedy.

```
Top 10 movie recommendations for User 3195: ['Blast from the Past (1999)' 'Punchline (1988)' 'Head Over Heels (2001)'
 'Heartbreakers (2001)' 'Houseboat (1958)' 'Guarding Tess (1994)'
 'Bachelor Mother (1939)' 'Failure to Launch (2006)'
 'Man About Town (2006)' 'More the Merrier, The (1943)']
```

Above is the screenshot that shows the model is recommending comedy movies to user 3195, who liked 20 comedies. After looking up most of the movies, all of them do seem to be comedies. A lot of them are romance as well, which means they didn’t like pure comedies only, but also romantic comedies.

If you made changes to your plans since the project update, what are they?

For the most part, no. Our original idea of using KNN to compare the word embeddings for genres/tags was followed all the way through. However, between the last update and now, we figured out a way to make our model better. That is to use the relevance data from genome_scores.csv. At first, we dismissed genome-scores and genome-tags as datasets we won't need to use since they aren't relevant to our model. One of us looked at them closer and realized they are actually very relevant. Quite literally even, because genome-scores gives relevance scores to each movie tag for each movie in a range between 0 to 1. For example, if Toy Story is tagged 'martial arts' by a user. That relevance score will be extremely low, something like .00003. Makes sense, since Toy Story is not a martial arts movie. But if it's tagged 'children' by a user, then the score would be near 1.

We were able to use these relevance scores to weigh our word embeddings. That way, tags that had no actual relevance to what a movie is about, will not heavily skew the averaged word embedding of the movie. This really proved useful because some of the tags users put for movies were very odd. That Toy Story example? It was real, someone actually tagged Toy Story 'martial arts'.

A description of everything you had to do to make the project work

At first, we were thinking of creating a tourist recommendation system. But we couldn't find any data for it online. Naively, we thought that creating our own datasets wouldn't be that hard. Luckily a graduate friend of Thomas' gave him some tips on how to start an ML project. The most important of which is that he doesn't do a project unless he has data for it already or it's a topic he knows well and can easily find data. We took this advice to heart and would be grateful we did. After doing this project and looking at what actual data should look like, we would have been completely lost at the data gathering part of the tourist idea.

After doing some research, we found the MovieLens dataset. It is a dataset that is used a lot in research projects on movies. The ml-25m version contains 62,000 movies, 162,000 users, and 15 million relevance scores across 1,129 tags. This dataset is essentially the dream dataset for a beginner. Not only did it make the project progress so smoothly but it even kept on giving the further we got into it, as discussed with the relevance scores.

We knew the next step was to pre-process and clean the dataset. We wanted two datasets in the end. A movie-based dataset, movies_genres&tags, and a users-based dataset, users_liked_genres&tags. The movies-based dataset would have essentially four columns: movieId, movies, genres&tags, and relevance_scores. The user-based dataset would only have two columns: userId and user_liked_genres&tags. We would create these by doing some initial cleaning on movies.csv, ratings.csv and tags.csv then merging all of them to get what we wanted. After movies_genres&tags was created, it had to be merged with genome_tags and genome_scores as well to attach the relevance score data.

Thomas Sosa
Tomas Arenas
Zack Wittmann

In terms of that initial cleaning though, first we dropped data we didn't need, like timestamps or rows with missing values. Then, we changed ratings.csv to only include ratings at or above 4.0. This number was arbitrary, we imagine anything 4.0 or above means the user liked the movie. Tags.csv data was changed to make it all lowercase words, which is what we needed to do if we wanted to use the genome-tags.csv data since it seems that movielens already preprocessed tags in that data to be lowercase only. Obviously, we also lowercase the genres in movies.csv as well so that when we merged there wouldn't be unnecessary duplicate genre/tags that can skew data.

After reviewing the two cleaned datasets that we created, it was time to build our model with it. Before we can actually do that, because we were using KNN, we had to have actual number values for the tags. Not only that, but the input had to be the same size for every case. Meaning if one movie had 10 genres/tags associated with it, the others had to too, unless it wouldn't work. So, for the genres&tags feature for each dataset, we turned each genre/tag into a word embedding, then averaged them out. In other words, we took all the genres/tags of the movies a user liked, and turned each of them into word embeddings. Then we averaged them out to get one word embedding for each user. We could later use the users-based dataset as a query for the KNN built on the movies-based dataset. Speaking on the movies based dataset, we actually did an extra step to the averaging of word embeddings due to necessity. You see, the movies in the movies-based dataset would have all the tags that every user has ever tagged that movie. This can be a problem due to the unpredictability of user inputs. So, the group at MovieLens saw fit to help with this problem by giving relevance scores for each tag given to each movie. The readme on how they did that is pretty vague. It amounts to we used our own ML model, trust us. So we did, and what we did with these relevance scores is we multiplied them to each word embedding to get a more accurate value of how much a particular tag represents a particular movie. Then we averaged the results out.

Considering we were using KNN, it was extremely simple. We essentially just plugged in our movie embedding values into a KNN model that uses cosine similarity.

```
movie_embeddings = np.stack(movies_genres_tags['movie_embedding'].values)
```

…

```
knn = NearestNeighbors(n_neighbors=10, metric='cosine')
knn.fit(movie_embeddings)
```

From there, we could just use the .kneighbors() function of KNN to input a user's averaged word embedding and get their recommended movies.

```
user_embedding_3195 =
users_liked_genres_tags.loc[users_liked_genres_tags['userId'] == 3195,
'user_embedding'].values[0]
```

```
distances, indices = knn.kneighbors([user_embedding_3195])
```

```
# Retrieve the top 10 recommended movies for user 3195
```

Thomas Sosa
Tomas Arenas
Zack Wittmann

```
recommended_movie_ids =
movies_genres_tags.iloc[indices[0]]['movieId'].values
# Map movie IDs to their titles
recommended_movie_titles =
movies_genres_tags[movies_genres_tags['movieId'].isin(recommended_movie_id
s)]['title'].values
print("Top 10 movie recommendations for User 3195:",
recommended_movie_titles)
```

Which model(s) did you choose?

We chose K-Nearest Neighbors(KNN). We knew from the start that we would want to use a simple model since this was the first ML project for all of us. KNN is one of the most straightforward models to understand and use, so it was perfect for us.

Did you compare your model with any other model?

Not as of now no, but luckily I'm sure we can find other models online easily since movie recommendation, as well as any recommendation system really, is a common ML project for both beginners and advanced practitioners.

How can you say your model worked?

We did one evaluation and plan to implement another, the evaluation we did is the eye test(using a bit of data) and the other evaluation we plan to implement is hits@k. I know the eye test thing sounds absolutely horrible, but hear us out:

hits@k - This was the original plan to evaluate, recommended to us by Dr. Piplai. The only thing that worried us is that, considering our extremely large dataset of movies, none of the movies that the user already watched and liked would appear on the recommendations. Even if k was a large number in the 100s, that is nothing compared to 62,000 movies. This brings up the question, at what point is using hits@k redundant if we set k to a large size. We'll probably write some code up with hits@k for the presentation on Tuesday, but for the purposes of this report we think the other method will be good enough. What we'll do is hits@k with k the size of 10, 100, 1,000, and 10,000 and observe that.

Eye test - We looked for users with obvious bias when it comes to their liked movies. Obviously, when you look at users_liked_genres&tags.csv, this can be a bit difficult to decipher. So we just wrote something to find the most watched genre of the first 5 users. We also made sure that the ratio between that most watched genre and the rest of the genres was large to make sure we are recommending the right movie. From this and adjusting it a bit to be able to look for specific genres, we found that the user 3185 liked comedy the most. Then, we just looked at the

recommended movies by that user and checked if they matched his biggest genre preference. I pasted an output above that you probably already saw.

If your model gave you unsatisfactory results, what are the reasons?

From what we've seen our model gave satisfactory results. They can definitely be improved, but that's what we'll talk about in the next section

What improvement do you think can be made to your project, and what would you have done differently?

An improvement we can make is to obviously add more features to the project. The ml-25 zip file contained a links.csv. These were codes to the IMDB website for each movie. From this website, we can pull more relevant features to add to our model. Such as director, lead actor, supporting actor, and length.

There are a few things we think we could have done differently. One of the things we planned to do is use relevance scores for the users_liked_genres&tags dataset. Originally, we wanted to have the users_liked_genres&tags dataset to include **every** genre/tag for the movies they liked. Now keep in mind, for the movies-based dataset, each movie had a row for itself. With the users-based dataset, a bunch of movie ids would be associated with a user, then the genres/tags for each of those movies, then we'd have a ton of relevance scores for **each** of their tags for **each** of their movies. Needless to say, the resulting csv when attempting a bunch of merges was unreadable, so we couldn't even be sure if we did this correctly. We dropped the idea as it was just not feasible for us, so we decided to simplify the users-based dataset to what we have now. We can argue that it still makes sense to do it the way we have it. Which is that the user has every genre/tag of the movies they like, along with all the tags they themselves gave the movies they like. Basically, their own tags would reaffirm their tastes. Also, relevance scores are only really used to evaluate tags from the tags.csv data, genres come from the movies.csv data and are 100% guaranteed to represent a movie, a users' liked genres/tags would be mostly genres. With more time I'm sure we can go back to the old idea and figure it out and optimize it.

The only other things we can think of that we could change is to add weight to users' liked ratings. Right now, a 4.0 rating is treated the same as a 5.0. At the same time, we would probably want to add more weight to recommendations with a high average score, but that goes back to the IMDB links thing. Right now it might recommend garbage movies haha.

Overall, we're pretty content with the progress we made in this project. Once we get further into our Machine Learning journey, we plan to revisit it and prioritize updating the model used from KNN to something a bit more advanced.