Movie Recommendation System

IST718 Big Data Analytics

Martin Alonso, LaRue Brown, Rashad Davis

2019-03-30

Introduction

Recommending users what movies to watch is a difficult task. Current systems use data from users, identifying previous movies watched and whether they liked the movie or not, getting an idea of what audiences like or hate. The result is an algorithm that presents users a list of potential movies that may pique their interest.

But these systems sometimes lack detailed analysis, or do take these analyses into account and stow them within their black box. Nevertheless, by not explicitly mentioning the additional data such as a wider rating system, user tags, actors or directors involved, and even genres, any recommendation system ends up wanting more precision.

For this project, our goal is to build a movie recommendation system that takes into account user ratings, how users tag movies, the genres of the movies they watch, movie director, and other available data, in order to recommend new movies that the user may not have seen before based on their past viewing experience.

To build this system, we'll use the Movie Lens database, which features 20 million movie ratings for 27,000 movies provided by users over the past 20 years. We also added data from The Movie Database, importing information regarding movie director, duration, PG-rating, description, and the movie poster.

Analysis

The Movie Lens database contains six comma-separated value files. These files contain data regarding Movies, Ratings, Tags, Links, Genome-Tags, and Genome-Scores. These last two files will not be used as they contain no relevant information regarding the actual movies and users.

The Movies database (table 1) has 27.278 observations and three variables: movieId, Title, and Genres. Aside from movieId, the Title and Genres columns have additional information that we mined. Title not only has the movie title, it also has the year the movie was released. This column was split into title and release year. Regarding the genres column, this has up to nine genres for a single movie. We split these genres into separate columns and created a sparse matrix to identify whether the movie had the genre or not (table 2).

The ratings data set has 20,000,263 observations and four variables (table 3). These columns have data about the userId, movieId, rating, and timestamp for when the rating was applied. The timestamp data was converted to datetime format and split into year, month, and day that the movie was rated.

Similarly, the tags data set also has four columns: userId, movieId, tag, and timestamp (table 4), this last one referring to the date that the tag was applied. However, this dataset has 465,564 observations. Using the Python NLTK package, the tag data was lower cased, cleaned for white space, and stemmed. However, the presence of non-Latin characters, plus

very specific tags, leaves this data with still room to be cleaned. We kept the tags with over 1,000 observations and created a sparse matrix of these tags, similar to what we did with the genres table (table 5).

Finally, the links data set has the movieId, tmdb link, and imdb link for each movie in the dataset, which we used to access data from the Movie Database.

Table 1: Movie data set

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Table 2: Movie data set cleaned

	movieId	title	releaseYear	decade	g1	g2	g3	g4	g5	g6	g7	g8	g9	g10	genresMatrix
0	1	Toy Story	1995	1990	Adventure	Animation	Children	Comedy	Fantasy	NaN	NaN	NaN	NaN	NaN	[0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ...
1	2	Jumanji	1995	1990	Adventure	Children	Fantasy	NaN	NaN	NaN	NaN	NaN	NaN	NaN	[0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ...
2	3	Grumpier Old Men	1995	1990	Comedy	Romance	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ...
3	4	Waiting to Exhale	1995	1990	Comedy	Drama	Romance	NaN	NaN	NaN	NaN	NaN	NaN	NaN	[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, ...
4	5	Father of the Bride Part II	1995	1990	Comedy	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...

Table 3: Ratings data set

	userId	movieId	rating	ratingDateTime	ratingDate	ratingYear	ratingMonth	ratingDay
0	1	2	3.5	2005-04-02 23:53:47	2005-04-02	2005	4	2
1	1	29	3.5	2005-04-02 23:31:16	2005-04-02	2005	4	2
2	1	32	3.5	2005-04-02 23:33:39	2005-04-02	2005	4	2
3	1	47	3.5	2005-04-02 23:32:07	2005-04-02	2005	4	2
4	1	50	3.5	2005-04-02 23:29:40	2005-04-02	2005	4	2

Table 4: Tags data set

	userId	movieId	tag	timestamp
0	18	4141	mark waters	1240597180
1	65	208	dark hero	1368150078
2	65	353	dark hero	1368150079
3	65	521	noir thriller	1368149983
4	65	592	dark hero	1368150078

Table 5: Tags data set cleaned

tag	userId	movieId	action	adventure	aliens	animation	atmospheric	based on a book	bd r	betamax	...	surreal	tense	thought provoking	thriller	time travel	true story	er
0	65	48082	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2011.0	0.0	0.0	0.0	0.0	0.0	0.
1	96	106696	0.0	0.0	0.0	2014.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	121	778	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	121	1288	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	121	2706	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.

After cleaning the data sets, we proceeded to analyze the data, trying to find interesting patterns and relationships within the data sets that might point us to relationships between movies, users, ratings, and tags that would allow us to build a good recommendation system.

We first wanted to know how many movies are being made each year (image 1). Grouping the number of movies by the year they were released, we found that the number of movies created first grew at a linear rate between 1891, the first year a movie was released according to the data, and the mid-1980s. However, around 1990, the number of movies made grew at a seemingly exponential rate, with the number of movies growing from an average of 170 movies per year between 1891 to 1985, to 600 movies per year between 1985 and 2014. Of importance is that this list does not contain indie movies, movies from smaller studios, or around the world. If we were only to include from Bollywood, the number of movies produced would surely double.

The other interesting item we can take away from the movies dataframe are the genres. Movies aren't usually boxed into one single genre - the average movie is linked to two genres. Part of understanding how recommendation systems work is understanding how genres mesh together; in other words, which genre combinations occur more frequently.

To answer this question, we resorted to market-basket analysis (table 6). We focused on finding relationships between genres that had a support greater than 0.01 and a lift over two; and opted to sort the values by lift and confidence in descending order. Surprisingly, we found that the most common relationship between genres occurred with movies classified as children movies. These movies are usually binned with adventure, action, animation, and fantasy movies, four additional genres that have seen a rise over the past 20 years with works from studios like DreamWorks, Disney, and Pixar. Additionally, movies classified as mystery or thriller also have a strong relationship, though this antecedent-consequent behavior is to be expected as most mystery movies need suspense to thrive on.

Image 1: Movies released per year.

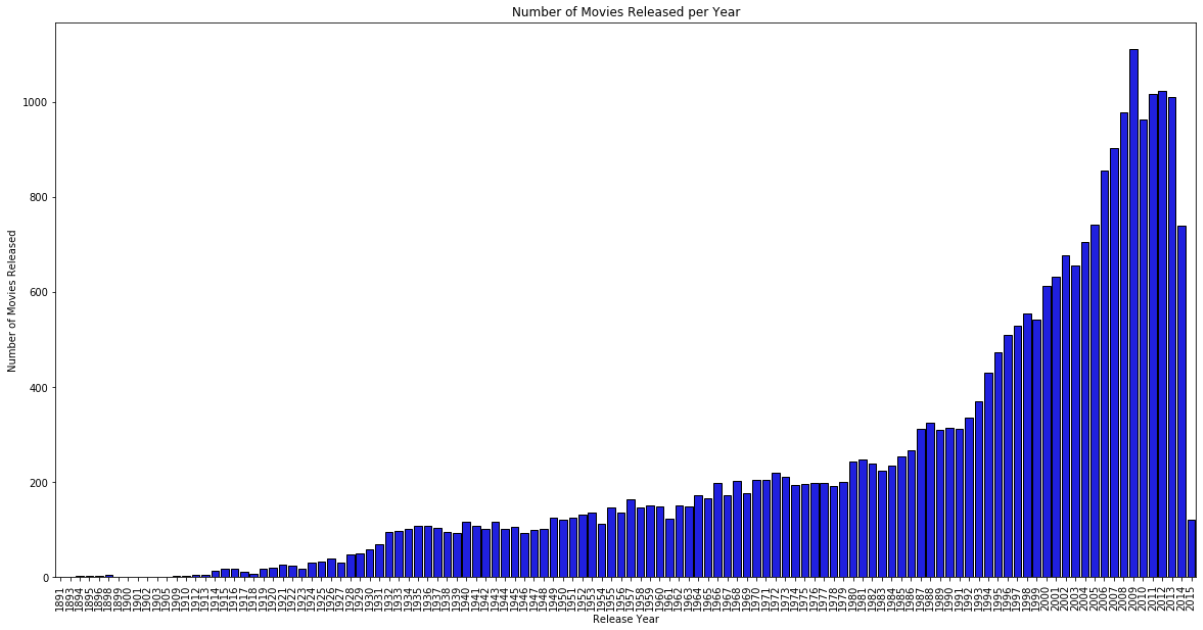


Table 6: Genre Market-Basket analysis

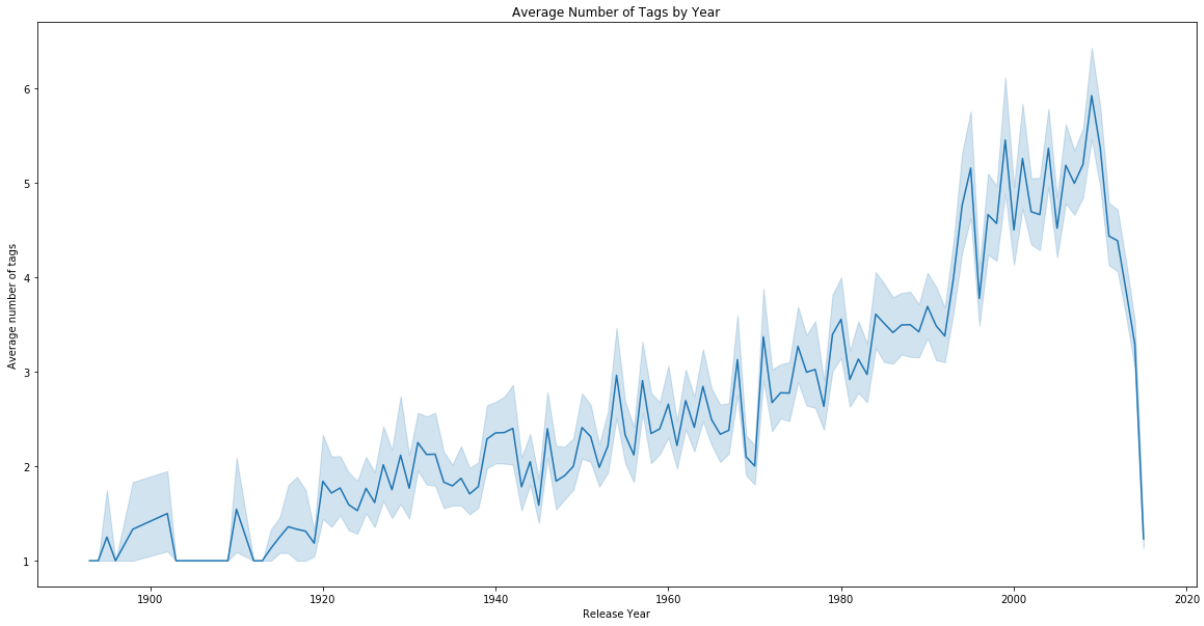
	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
335	(Animation)	(Children)	0.037649	0.041755	0.017230	0.457644	10.960143	0.015658	1.766817
334	(Children)	(Animation)	0.041755	0.037649	0.017230	0.412643	10.960143	0.015658	1.638441
763	(Children)	(Fantasy)	0.041755	0.051763	0.011401	0.273047	5.274903	0.009240	1.304398
762	(Fantasy)	(Children)	0.051763	0.041755	0.011401	0.220255	5.274903	0.009240	1.228921
652	(Children)	(Adventure)	0.041755	0.085380	0.016497	0.395083	4.627344	0.012932	1.511977
653	(Adventure)	(Children)	0.085380	0.041755	0.016497	0.193216	4.627344	0.012932	1.187734
1018	(Fantasy)	(Adventure)	0.051763	0.085380	0.018733	0.361898	4.238666	0.014313	1.433344
1019	(Adventure)	(Fantasy)	0.085380	0.051763	0.018733	0.219407	4.238666	0.014313	1.214765
70	(Animation)	(Adventure)	0.037649	0.085380	0.012904	0.342746	4.014350	0.009690	1.391577
71	(Adventure)	(Animation)	0.085380	0.037649	0.012904	0.151138	4.014350	0.009690	1.133695

Having explored the relationship between genres, it was necessary to also explore similar relationships between the tags that were being applied by users to all the movies in the data set. However, relationships between the tags were not as strong as those present within the genres, with the support value needing to be greater than $1x10e^{-5}$ before finding any significant rules, and even with this support, the lift barely passed 1 on three tag relationships. Exploring the tag dataset, we found the problem: After cleaning the data for special characters, lower-casing the tags, and stemming them using the PorterStemmer from the NLTK package, we observed tags that were written with non-Latin characters. There are also tags that only appeared relevant to a single movie, in addition to tags that, though referring to the same thing, were written differently that they would require manual reviewing before any additional work could be done (i.e. James Bond, Bond, 007, 007 James Bond).

We opted to not focus on doing any additional work on the tags, and instead focused on looking at the average tags movies receive per year that the movie was released and study

how tags evolve over time. The idea behind this latter analysis was to understand how tags influence the rise of other tags, similar to fads, and how these might affect recommendations.
Over the years spanning the data set, we found that the number of tags has increased from an average of one tag per movie, to a current average of six tags (image 2). Nothing telling but it makes sense given that, nowadays, movies have more ideas behind them that push users to be as detailed as possible when tagging movies.

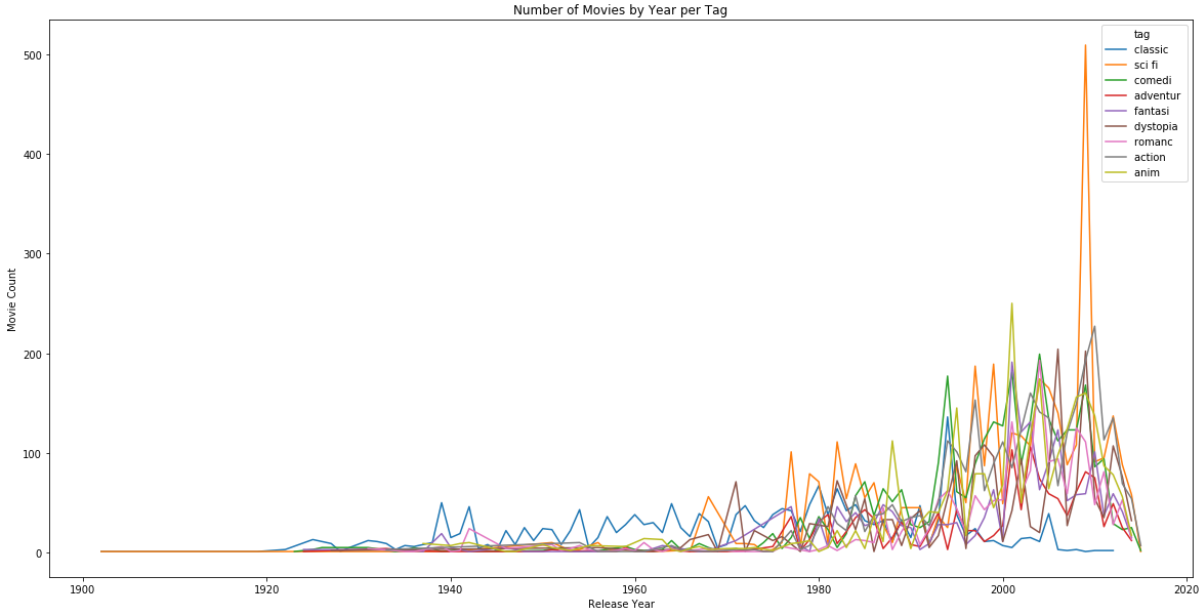
Image 2: average number of tags per movie



This also seems to be the case when looking at the number of movies that have a tag the year they were released (image 3). We looked at ten of the most common tags to find if they had a set of years where they appeared more frequently, and save for some, most seem to hold a steady pattern. There are also some very interesting patterns.
For example, movies that have a classic tag see a steady number of movies released each year. But not many movies receive this tag, which is interesting as they may appear to be either old movies, like Casablanca, or movies that receive an “instant classic” tag upon release.
Other noteworthy patterns are the rise of science fiction movies which start to appear in the 1970s and see a steady rise over the years, and peaking in the 2010s. Comedies are the only movies that saw a large number of releases in the 80s; the 1990s saw a decline in these movies before they increased their numbers in the early aughts.
In reality, the only natural conclusion that we could draw from analyzing the tags applied to movies is that only the most recent movies (movies released over the past 25 years) are more likely to have received a tag than movies that were released further back. This may skew the recommendation system into recommending movies with tags, ergo more recent movies, than movies that were released before the 1990s.

For the ratings dataset, rather than focusing on the timestamp of when a movie was rated, we decided to focus on the average rating by year each movie was released. Using the movieId variable as a link between both tables, we merged the movie and ratings table to get the year each movie was released.

Image 3: Tag evolution by release year



Nevertheless, since the number of movies released per year has grown, we used a weighted average to account for the total number of movies released per year (image 4). What we found was that, for most of the years that movies were made, the average rating was 4.0. But, over the past years, as more movies started being released, the average rating dropped to approximately 3.5. We suspect that this means that the average user is remiss to score a movie negatively, preferring to give a movie a rating between 3.0 and 4.5 rather than penalize it too heavily.

We again binned the ratings data, this time counting the number of movies by rating (image 5), finding that ratings are skewed to the right, meaning that our previous hypothesis is true: Users are very unlikely to give movies a low rating, and the majority of movies have a rating between 3.5 and 4.0.

Finally, the last data that we decided to mine for information was from the the Movie DataBase website (TMDB). Though not part of the original data set, the links table has a link for each movie in the movies data set. Using this link, we scraped information regarding movie director, actors, movie description, duration, PG-rating, and the movie poster. This data was scraped because, in addition to using the tags, genres, and ratings data provided by Movie Lens 20M, we also wanted to account for users who like certain directors or actors within the model. We also wanted to account movies with similar PG-ratings, thus avoiding recommendations that may not seem appropriate to different age groups.

Lastly, the movie description interested us, as we could use keywords from the descriptions to try and identify key ideas and themes across movies, giving us a better idea of the movies actual plot and whether it would interest our potential end-users.

Image 4: Weighted Average Rating per Movie Release Year

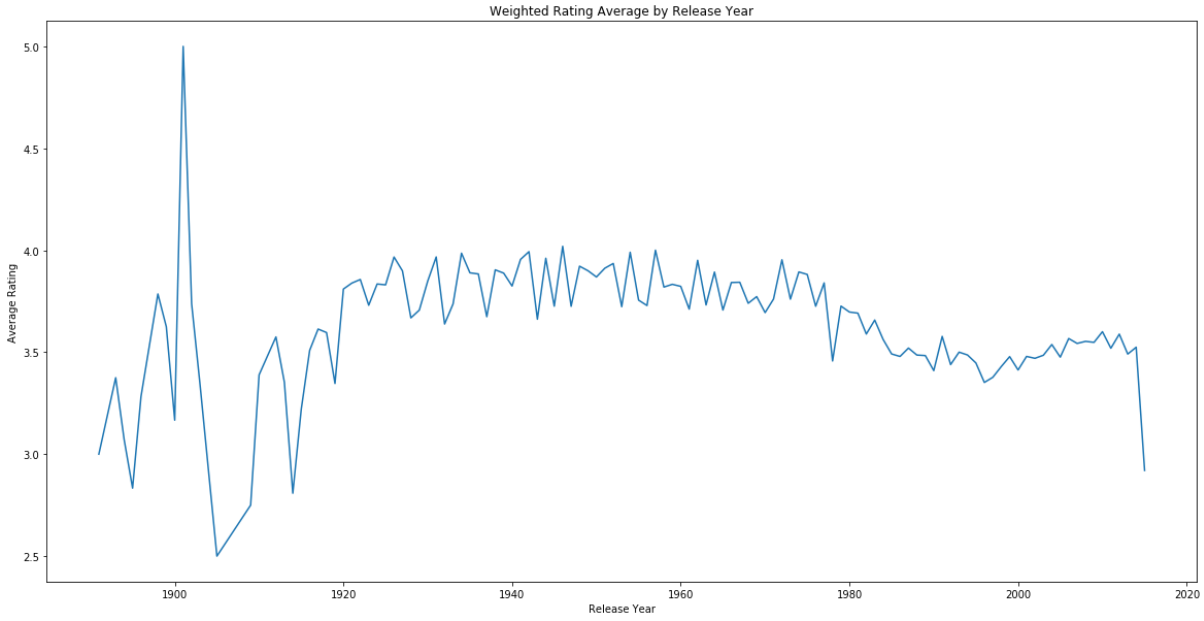
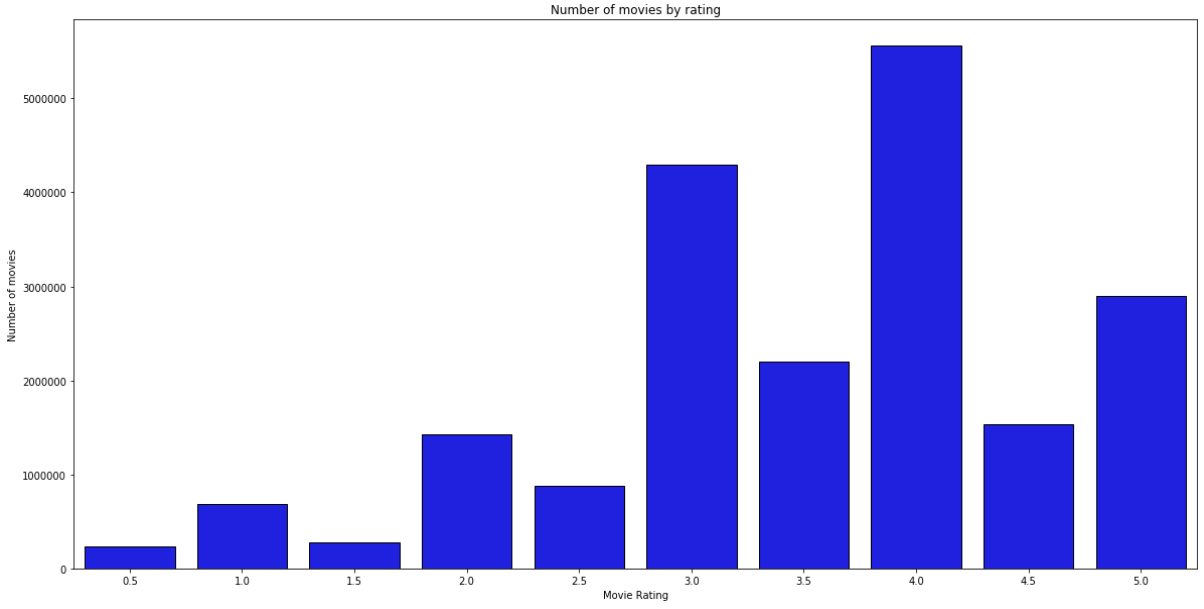


Image 5: Number of movies per rating



Finally, having cleaned all the data sets, we combined these into a single dataframe (table 7), sans the data scraped from TMDB, that will be used when creating our recommendation models. This final data set has 44,881 observations spread out over 25 variables (table 8).

Table 7: Final data set.

	movieId	title	releaseYear	decade	g1	g2	g3	g4	g5	g6	...	userId	rating	ratingDateTime	ratingDate	ratingY
0	1	Toy Story	1995	1990	Adventure	Animation	Children	Comedy	Fantasy	NaN	...	3596	1.0	2010-11-21 04:00:52	2010-11-21	2010
1	1	Toy Story	1995	1990	Adventure	Animation	Children	Comedy	Fantasy	NaN	...	10616	3.5	2010-06-24 05:25:56	2010-06-24	2010
2	1	Toy Story	1995	1990	Adventure	Animation	Children	Comedy	Fantasy	NaN	...	11355	4.0	2012-11-29 02:02:39	2012-11-29	2012
3	1	Toy Story	1995	1990	Adventure	Animation	Children	Comedy	Fantasy	NaN	...	11621	4.0	2006-07-07 07:41:09	2006-07-07	2006
4	1	Toy Story	1995	1990	Adventure	Animation	Children	Comedy	Fantasy	NaN	...	18390	5.0	2006-07-10 22:46:50	2006-07-10	2006

Table 8: Final data set description

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 44881 entries, 0 to 44880
Data columns (total 25 columns):
movieId           44881 non-null object
title             44881 non-null object
releaseYear       44881 non-null int64
decade            44881 non-null object
g1                44881 non-null object
g2                39505 non-null object
g3                27667 non-null object
g4                12569 non-null object
g5                3695 non-null object
g6                845 non-null object
g7                292 non-null object
g8                2 non-null object
g9                2 non-null object
g10               2 non-null object
genresMatrix      44881 non-null object
userId            44881 non-null object
rating            44881 non-null float64
ratingDateTime    44881 non-null datetime64[ns]
ratingDate        44881 non-null object
ratingYear        44881 non-null object
ratingMonth       44881 non-null int64
ratingDay         44881 non-null int64
imdbId            44881 non-null int64
tmdbId            44849 non-null float64
tagsMatrix        44881 non-null object
dtypes: datetime64[ns](1), float64(2), int64(4), object(18)
memory usage: 8.9+ MB
```

Modeling

For this exercise, we opted to build two recommendation systems. The first one is a user-based recommendation system that uses the LightFM python package; the second is a movie-based similarity score system. Both systems will use 50 percent of the data as a training set, the remainder 50 percent will be used as the testing set.

In order to use the LightFM model, we have to pivot the data, having the userIds as the observations and the movieIds as the variables, creating a sparse matrix, flagging each movie that the user gave a rating too, irrelevant of how the user rated the movie (table 9).

The LightFM model was run through 100 epochs and using 5 threads, measuring the precision with which it recommends movies to users. By precision, we're measuring how the recommended movies compare in genre, ratings, and description to the movies already watched by the user. Under this measurement, a perfect score would be 100 percent, but given that we are looking to recommend movies that a user hasn't seen based on those that the user has already seen, we're expecting a lower precision score, since our system will be recommending more than one movie to each user.

Table 8: Sparse matrix for LightFM model

movieId	1	2	6	7	10	16	17	18	19	21	...	119065	119145	119655	120134	127098	128600	128734	128738	128981	128991
userId																					
92475	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
105707	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
32964	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
62396	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
62716	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The issue with this first model, however, is that this model receives a user ID and returned a list of movies that would be recommended for that user based on movies watched which is highly subjective given that some recommendations don't make sense. An individual may not want to go from Step Brothers to Brokeback Mountain or Lord of the Rings.

The second movie-based model receives a movie ID as input and returns a list of movies that are related based on weights provided by the coders. This algorithm is courtesy of Movie recommendation algorithm. We created a sparse matrix based on both genres and the number of movies the user has rated. Then, the algorithm searches through the list of tags each movie has received, with user assigned weights (table 10), and compares the movie's genres and tags to those of the other movies in the data set, looking for those most similar to each other.

The model then takes as input the movie a user has watched, extracts the ratings, genres, and tags that were given to the movie by other users, and ranks the most similar movies, returning a list that shows the user a list of movies most likely to pique his interest, with a lower similarity score meaning a higher likelihood of the movie being of interest to the user.

Table 10: User assigned weights for similarity model

Parameters	Weight
Genre	1.0
Tag	1.5
Title	0.8
Rating	0.2
Ratings Received	0.005
Year Released	0.1

Results

After training the LightFM model, we ran the testing set through it, predicting five movies for each userId in the sparse matrix. We printed out the results for three of users along with the precision scores. Sadly, the scores were very low, the highest barely breaking three percent. Then again, the objective of this model is not to predict movies a user might have seen but those that might interest our potential user, so a precision score is probably not the best way of measuring the results of our model (table 11).

Table 11: LightFM model results

UserId	Movies Seen	Movies Recmmended	Precision
121	[This is Spinal Tap, American Pie, American Pie2, National Lampoon's Van Wilder, Old School, The 40-year old Virgin, Lord of War, Knocked Up, Step Brothers]	[Shawshank Redemption, Pulpu Fiction, The Two Towers, The Fellowship of the Ring, Inception]	2.97%
129	[Donnie Darko]	[Away We Go, Scott Pilgrim vs the World, Pearl Harbor, Happy Together, Avatar]	3.08%

The results of the similarity model ignore what the user watched previously and opt for matching the movie inputted and returning those that are more similar to the tone of the selected movie. The results (table 12) of this model are more encouraging given the movies being return and the similarity scores.

We found that many movies had a similarity score very close to negative one, meaning that the movies being recommended are very similar to the movie being used as an input.

Table 12: Results for similarity model

```
# Step Brothers

similarityResult  = checkSimilarity(60756)
similarityResult.head(10)
```

	movieId	title	genres	rating_count	rating_avg	similarity
11168	46970	Talladega Nights: The Ballad of Ricky Bobby (2...	Action|Comedy	2352.0	3.200680	-1.039294
7958	8641	Anchorman: The Legend of Ron Burgundy (2004)	Comedy	5258.0	3.492773	-0.818223
18629	92719	Tim and Eric's Billion Dollar Movie (2012)	Comedy	41.0	2.426829	-0.758088
12469	58156	Semi-Pro (2008)	Comedy	597.0	2.869347	-0.725175
4720	4816	Zoolander (2001)	Comedy	7499.0	3.255634	-0.701474
13106	63131	Role Models (2008)	Comedy	1930.0	3.626425	-0.675701
22344	107348	Anchorman 2: The Legend Continues (2013)	Comedy	343.0	3.055394	-0.611551
12032	54503	Superbad (2007)	Comedy	6670.0	3.786207	-0.590662
13262	64969	Yes Man (2008)	Comedy	1863.0	3.456522	-0.553432
12691	59784	Kung Fu Panda (2008)	Action|Animation|Children|Comedy|IMAX	4468.0	3.712847	-0.539667

```
# Lord of War

similarityResult  = checkSimilarity(36529)
similarityResult.head(10)
```

	movieId	title	genres	rating_count	rating_avg	similarity
12326	56788	Charlie Wilson's War (2007)	Comedy|Drama|War	2074.0	3.689007	-0.603572
10687	41997	Munich (2005)	Action|Crime|Drama|Thriller	4745.0	3.731718	-0.584506
9937	32587	Sin City (2005)	Action|Crime|Film-Noir|Mystery|Thriller	15481.0	3.874201	-0.524205
10109	33437	Unleashed (Danny the Dog) (2005)	Action|Crime|Drama|Thriller	1380.0	3.517029	-0.498600
10377	36517	Constant Gardener, The (2005)	Drama|Thriller	3572.0	3.747060	-0.491279
5917	6016	City of God (Cidade de Deus) (2002)	Action|Adventure|Crime|Drama|Thriller	12937.0	4.235410	-0.467036
2873	2959	Fight Club (1999)	Action|Crime|Drama|Thriller	40106.0	4.227123	-0.388892
10562	40278	Jarhead (2005)	Action|Drama|War	2987.0	3.486776	-0.369341
10432	37733	History of Violence, A (2005)	Action|Crime|Drama|Thriller	3878.0	3.636668	-0.358465
11623	50944	S.P.L.: Kill Zone (Saat po long) (2005)	Action|Crime|Drama|Thriller	27.0	3.833333	-0.322139

Conclusions

Overall, the results suggested by the similarity model is better than the results given out by the LightFM model. Given these initial results, we recommend working with the similarity model to start our movie recommendation program.

With more time to work on this project, we could have done more text analysis on the movie descriptions and use them to also improve our recommendation system. Furthermore, cleaning the tags data more deeply would also have helped us improve the identify better relationship between movies, genres, and other tags. Having tags with non-Latin characters are very difficult to clean and is one of the first things we can improve upon.

Additionally, though the LightFM model showed a poor performance when recommending movies, there were some recommendations that were intriguing to say the least. One idea we discussed and could improve our recommendation system would be combining these two models to take into account not only what the similarity to a movie is but take into account what a user has seen and how he has rated the movies he's seen.
Finally, adding data from other sources like the Internet Movie Database, Rotten Tomatoes, and MetaCritic would give us other angles to assess how movies are rated, tagged, and described. This would also increase the data on movies watched by users, given us a better reading of what movies users have seen and allowing us to be more specific when it comes to identifying potential movies that might interest users.
As to the current models, we recommend that any decisions made as to what movies might pique a user's future interest should be done considering the similarity model rather than the LightFM model, at least until we reach a point were we can combine and improve both models.

References

- Movie Lens 20M, https://grouplens.org/datasets/movielens/20m/. Date accessed 27 March, 2019
- The Movie Database, https://www.themoviedb.org/. Date accessed 27 March, 2019
- "Film Industry in India" , Statista: The Statistical Portal, https://www.statista.com/topics/2140/film-industry-in-india/. Date accessed 27 March, 2019
- "Comprehensive Guide to build a Recommendation Engine from scratch (in Python)", *Analytics Vidhya*, https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/. Date accessed 18 March, 2019
- "Convert Pandas Column to DateTime", *Stack Overflow*, https://stackoverflow.com/questions/26763344/convert-pandas-column-to-datetime. Date accessed 10 February, 2019
- "Converting Strings to datetime in Python", *Stack Abuse*, https://stackabuse.com/converting-strings-to-datetime-in-python/. Date accessed 11 February, 2019
- "Count max occurrence of character in pandas dataframe", *Stack Overflow*, https://stackoverflow.com/questions/47362355/count-max-occurence-of-character-in-pandas-dataframe. Date accessed 11 February, 2019
- "Creating a Simple Recommender System in Python using Pandas", *Stack Abuse*, https://stackabuse.com/creating-a-simple-recommender-system-in-python-using-pandas/. Date accessed 12 February, 2019
- "Embedding image in ipython notebook for distribution", *Stack Overflow*, https://stackoverflow.com/questions/26068316/embedding-image-in-ipython-notebook-for-distribution. Date accessed 24 March, 2019

- "Extracting just Month and Year from Pandas Datetime column", *Stack Overflow*, https://stackoverflow.com/questions/25146121/extracting-just-month-and-year-from-pandas-datetime-column. Date accessed 11 February, 2019
- "Hide all warnings in python", *Stack Overflow*, https://stackoverflow.com/questions/9031783/hide-all-warnings-in-ipython. Date accessed 10 February, 2019
- "How do you change the size of figures drawn with matplotlib?", *Stack Overflow*, https://stackoverflow.com/questions/332289/how-do-you-change-the-size-of-figures-drawn-with-matplotlib. Date accessed 07 February, 2019
- "How do you check in python whether a string contains only numbers?", *Stack Overflow*, https://stackoverflow.com/questions/21388541/how-do-you-check-in-python-whether-a-string-contains-only-numbers. Data accessed 11 February, 2019
- "Hot to build a Movie Recommender System in Python using LightFM", *Towards Data Science*, https://towardsdatascience.com/how-to-build-a-movie-recommender-system-in-python-using-lightfm-8fa49d7cbe3b. Date accessed 19 March, 2019
- "How to change dataframe column names in pyspark?", *Stack Overflow", https://stackoverflow.com/questions/34077353/how-to-change-dataframe-column-names-in-pyspark. Date accessed 08 February, 2019
- "How to split a column into two columns?", *Stack Overflow*, https://stackoverflow.com/questions/14745022/how-to-split-a-column-into-two-columns. Date accessed 10 February, 2019
- "I want to create a column of value_counts in my pandas dataframe", *Stack Overflow@ https://stackoverflow.com/questions/17709270/i-want-to-create-a-column-of-value-counts-in-my-pandas-dataframe. Date accessed 16 February, 2019
- "Movie Recommendation Algorithm", *Kaggle*, https://www.kaggle.com/bakostamas/movie-recommendation-algorithm. Date accessed 24 March, 2019
- "Pandas: add timedelta column to datetime column (vectorized)", *Stack Overflow*, https://stackoverflow.com/questions/38355816/pandas-add-timedelta-column-to-datetime-column-vectorized. Date accessed 11 February, 2019
- "Python For Data Science Cheat Sheet", *Data Camp*, https://s3.amazonaws.com/assets.datacamp.com/blog_assets/PySpark_SQL_Cheat_Sheet_Python.pdf. Date accessed 08 February, 2019
- "Python | Implementation of Movie Recommender System", *Geeks for Geeks*, https://www.geeksforgeeks.org/python-implementation-of-movie-recommender-system/. Date accessed 12 February, 2019
- "tmdbsimple", *PyPi*, https://pypi.org/project/tmdbsimple/. Date accessed 24 March, 2019