Report

on

Movie Recommendation System

Viral Shanker
Sandeep Moparthy
Kaustubh Lad
Drashti Patel

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Introduction and Motivation

Recommendation systems are the backbone of modern entertainment consumption. In an option-rich entertainment industry - be it video games, books, music, or movies - there are far far more options available to a potential consumer than they could feasibly explore in their life. Furthermore, most users have no desire to spend time exploring choices they like and dislike: by definition they would like to consume material they are going to like. And this is where recommendation systems come in. Using information - using historical preferences of the consumer and features of the industry itself - recommendation systems predict how much a user will like an item, and recommend the ones it has the most confidence the user will rate highly. Netflix, YouTube, Steam, Spotify, Amazon - just to name a few - all heavily rely on recommendation systems to keep their consumers engaged with their product.

Motivated by the importance of these recommendation systems, we focus on one of the biggest entertainment sectors of today: movies. We leverage the MovieLens Dataset which contains ratings, tags, and various other metadata for movies.

There are two main approaches to take: content-based filtering and collaborative filtering. A content-based approach returns movies similar to a given movie. Similarity can take on many different meanings which we will explore. Collaborative filtering leverages a user's history to make recommendations on what that user may like in the future. Our project revolved around these two approaches.

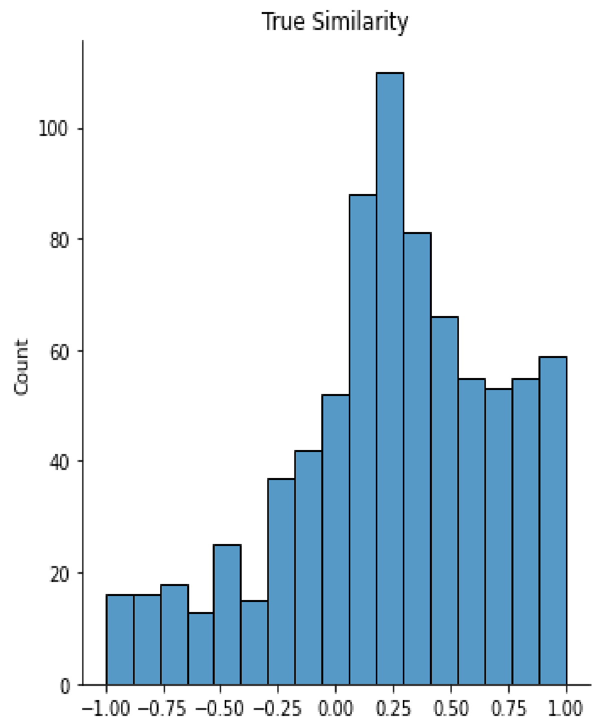
Methodology

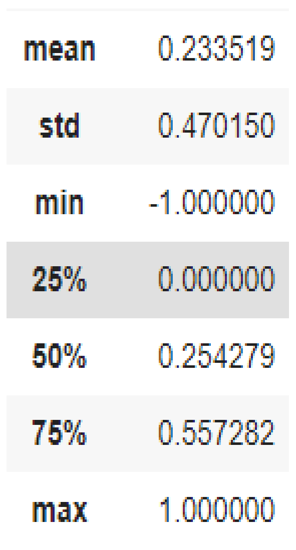
Content-Based Filtering

We took 3 different approaches to content-based filtering, with each one being over a different feature space. The biggest challenge in content-based filtering is to evaluate data since this is typically done via unsupervised means which are always challenging to evaluate. We use various different metrics, some internal, some external. One approach we can use as a standard is a custom metric called "True Similarity."

True Similarity

Keeping our goal of recommending movies people will enjoy, we can define our own metric of similarity. Given two movies we consider users who have rated both movies. Then we consider the correlation between the users' ratings of the two movies. This essentially tells us how similar the ratings of the two movies are, since our goal, after all, is to identify movies that will be rated similarly to the given movie. One drawback however is with unpopular movies. If only one user has seen both the movies, clearly our similarity metric is not of much use - so we return NaN for the similarity of 2 movies that have only been watched by less than some threshold of views (we default to 2). We sampled True Similarity to get an idea of its distribution.



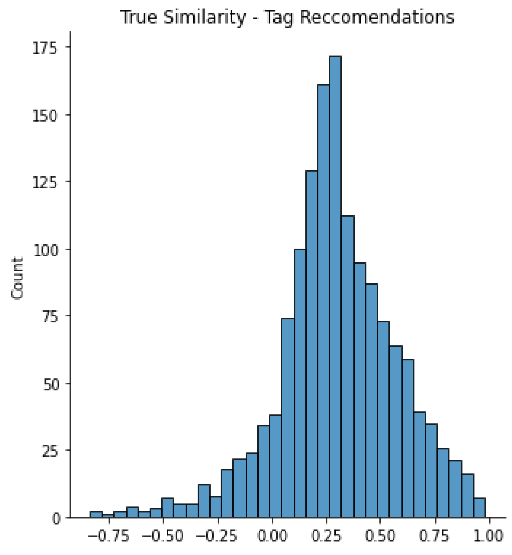


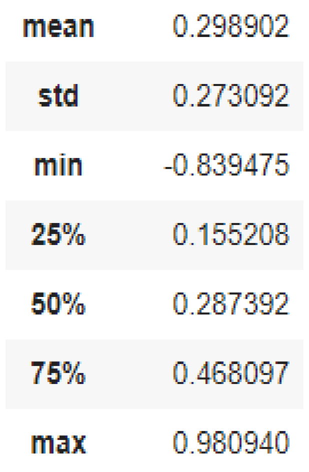
mean	0.233519
std	0.470150
min	-1.000000
25%	0.000000
50%	0.254279
75%	0.557282
max	1.000000

Tag-Based Filtering

The data comes with tags - the tag a user I gave to movie J. These tags can range from “epic” to “snakes.” A huge variety is present. We have 75000 tags, but only 16000 have been used more than 5 times. Inspired by the min_df metric in a tf_idf matrix, we only consider tags that have been more than 5 times. From here, we have essentially a tag frequency matrix - tag X was assigned to movie Y Z times. Using NLP techniques, we make a TF_IDF matrix from these tags to develop a feature space. We then use cosine similarity to find the closest movies to a given movie.

One huge drawback of this approach is that we limit ourselves to around 25000 movies only because of our aforementioned limited tags approach. And moreover, some movies may only have 1-2 tags, making them have almost no similar ones. Regardless, we recommended the most similar movies to a given movie in terms of cosine similarity over tag features and calculated their True Similarity, as mentioned above.



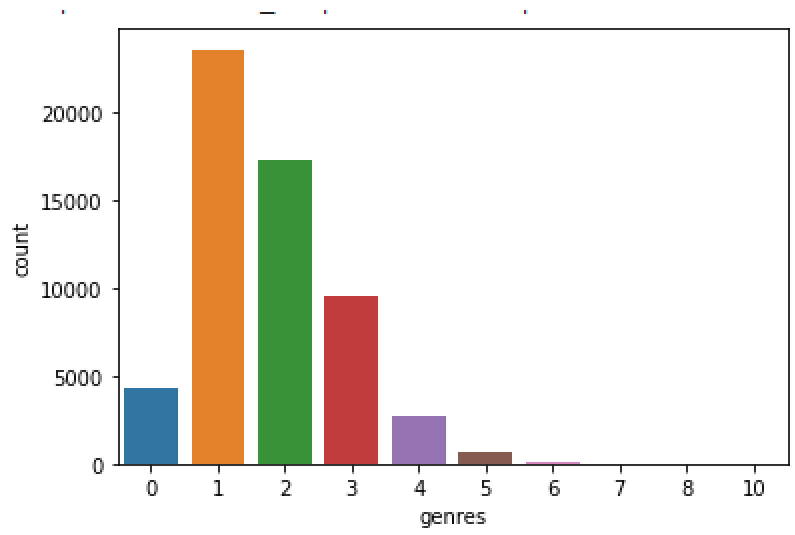


mean	0.298902
std	0.273092
min	-0.839475
25%	0.155208
50%	0.287392
75%	0.468097
max	0.980940

Note the marked improvements over random true similarity. We are barely mentioning movies that have a negative correlation almost at all, and our 25% and 50% percentiles are way up, as is our mean. Furthermore, we have dramatically reduced the variance over random recommendations.

Genre Based Filtering

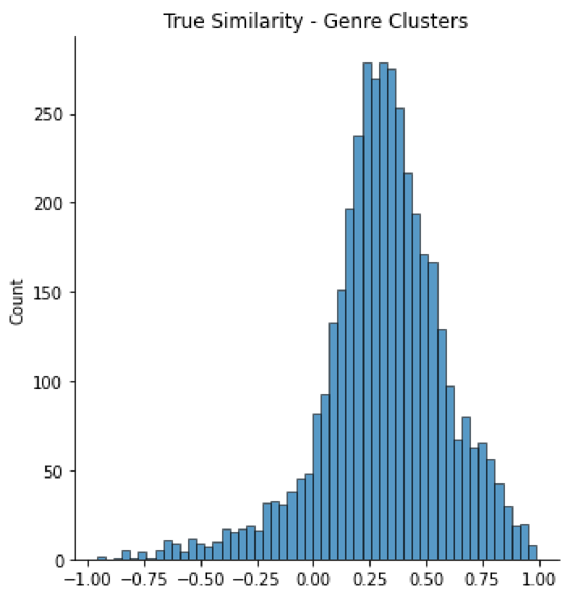
We have genres assigned to each movie. These genres unfortunately are not in a clean format at all and must be extracted from the data. Once extracted, we one-hot-encode the genres, so if a movie is classified as action and drama, the action and drama columns have a 1 in that row, while the rest are zero. This allows us to make yet another feature space. We also include the movie's average rating as well.

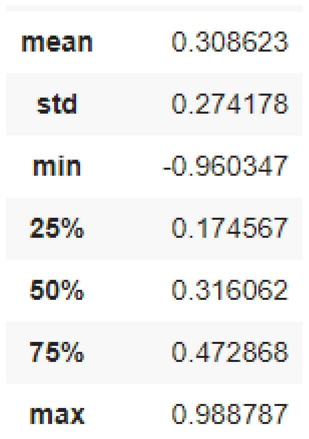


A movie can have multiple ratings, as seen on the figure to the left, but most movies only have 1 or 2. This should tell us that our feature space is not very rich, so the quality of recommendations could be questionable. Regardless, we persevere.

This time, we try out kmeans. The elbow method for selecting the number of clusters did not work, so we picked 500 clusters after some trial and error. This could definitely be tweaked. Regardless, we normalized data and applied kmeans, but with 500 clusters, the inertia was 2600. We recommended movies that happened to be in the same cluster as the given movie.

Again, we apply the True Similarity metric.





mean	0.308623
std	0.274178
min	-0.960347
25%	0.174567
50%	0.316062
75%	0.472868
max	0.988787

We get surprisingly good results. Overall the same theme as the tag clusters, but better performance. We are barely mentioning movies that have a negative correlation almost at all, and our 25% and 50% percentiles are way up, as is our mean. Furthermore, we have dramatically reduced the variance over random recommendations. This beats out tag similarity as a recommendation metric.

Review-Based Filtering

The reviews - the review a user I gave to movie J. These reviews can range from a simple experience to rage about the uneasiness of sitting through the entire movie. A huge variety is present. We have 44513 reviews, but only 16000 have been used more than 5 times. Inspired by the min_df metric in a tf_idf matrix, we only consider tags that have been more than 5 times. From here, we have essentially a word frequency matrix - word X was assigned to movie review Y Z times. Using NLP techniques, we make a TF_IDF matrix from these tags to develop a feature space. We then use cosine similarity to find the closest movies to a given movie.
Our approach ensures a user can input a particular movie of liking and get 9-28 recommendations. These are purely based on the cosine similarity. One huge drawback of this approach is that we limit ourselves to the user input and not the history of the user's usage or views. One other huge drawback is that we only see the reviews here which may potentially be unique per user and the reviews can sometimes be very different from the movie story or description. A potential use of this approach will only recommend

movies with a large number of good reviews to the user if he searches for a movie that already has good reviews and wise verses.
We have tried defining an internal metric since an external metric to evaluate reviews may not be possible. We have checked for existence of the words of genre for an input movie and mapped them through the recommended movies' genre. This gave us a true/false situation. Exploiting this output, we have found the accuracy (sklearn's accuracy_score) for the recommendations and it was observed to be **0.66 and an MAE of 0.33.**

Description-Based Filtering

The data comes with overviews - the description a user I gave to movie J. These overviews can range from a simple sentence to the entire story. A huge variety is present. We have 44513 reviews, but only 16000 have been used more than 5 times. Inspired by the min_df metric in a tf_idf matrix, we only consider tags that have been more than 5 times. From here, we have essentially a word frequency matrix - word X was assigned to movie review Y Z times. Using NLP techniques, we make a TF_IDF matrix from these tags to develop a feature space. We then use cosine similarity to find the closest movies to a given movie.
Our approach ensures a user can input a particular movie of liking and get 16-28 recommendations. These are purely based on the cosine similarity. One huge drawback of this approach is that we limit ourselves to the user input and not the history of the user's usage or views.
We have tried defining an internal metric since an external metric to evaluate reviews may not be possible. We have checked for existence of the words of genre for an input movie and mapped them through the recommended movies' genre. This gave us a true/false situation. Exploiting this output, we have found the accuracy (sklearn's accuracy_score) for the recommendations and it was observed to be **0.357 and an MAE of 0.64.**

PLEASE PUT YOUR PART HERE SANDEEP

Collaborative Filtering

This is where we leverage the true power of the dataset we have: the interaction data. We leverage the rating each user has given a movie and recommend based on that. We took two approaches using this framework.

First question is what is collaborative filtering ? Simple answer is that it is a user based recommendation system. Which means that I will recommend you one item that other users that are similar to you liked. This recommendation is used by many big companies like Google, Netflix, Amazon etc. If you see on those companies' websites they show us that you may like this product or movies because some of the users similar to you also like this. So, what does it mean by similar users like you and how they find that ? So, let's take an example of Netflix, you liked couple of movies like

'Batman' , 'Inception' .Now as per the database there is one user who also likes this movie and he likes the movie 'Prestige' but you haven't watched this movie then it will recommend you to watch that movie. They will find the similarity between two users using the rating given by users in the previously watched movie and to calculate some kind of number to compare they use Euclidean distance, Pearson Correlation, Cosine distance etc. There is no fix one computation method. Those companies also use some personal information like user's age, profession, hobbies etc not just the rating but those companies have that kind of information.

Reading the Data set

We used two tables for this:

1. Movies: to get movie title
2. Ratings: to get user ratings

Dataset is very big and it can take a lot of computation. So, taking data of first 100 users and then split out the dataset into two parts for testing and training. We will use 80% of our data for training our model. We split it randomly but try to train using all users' ratings.

We have used Pearson correlation to calculate similarity between two users as we already used cosine similarity.

- Pearson Correlation

$$sim(a,b) = \frac{\sum_{m\in M}(r_{a,m} - \bar{r_a})(r_{b,m} - \bar{r_b})}{\sqrt{\sum_{m\in M}(r_{a,m} - \bar{r_a})^2}\sqrt{\sum_{m\in M}(r_{b,m} - \bar{r_b})^2}}$$

$r_{a,m}$ = Rating of movie “m” by user “a”

r_a = Mean rating given by user “a”

Used Pearson correlation to get to know similarity between two users. To compute the predictions we will use the following formula:

$$\boxed{pred(a,m) = \frac{\sum_{b \in N} sim(a,b) * (r_{b,m})}{\sum_{b \in N} sim(a,b)}}$$

- pred(a,m) = Predicted rating of user “a” for movie “m”
- sim(a,b) = Similarity between user “a” and user “b”
- M = Set of common rated movies by user “a” and “b”
- $r_{b,m}$ = Rating of movie “m” by user “b”
- r_a = Mean rating given by user “a”

```
similarity_result[15]
```

```
{1: 0,
 2: 0,
 3: 0,
 4: 0.12747493490655398,
 5: 0.270611428779858,
 6: 0,
 7: 0,
 8: 0,
 10: 0,
 11: 0,
 12: 0,
 13: 0,
 14: 0,
 16: 0,
 17: 0,
 18: 0.7905694150420948,
 19: 0,
 20: 0,
```

We have put one filter in that if users have watched more than two similar movies then it will give similarity results. Otherwise it will give 0 Score.

Evaluation of Collaborative Filtering

- Root mean square error

- $$RMSE = \sqrt{(\frac{\sum(\hat{y} - y)^2}{n})}$$

- Mean absolute error

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Where n is total number of data points

```
[1.1814134640639267, 0.9009308218685784]
```

Result of RMSE = 1.181413

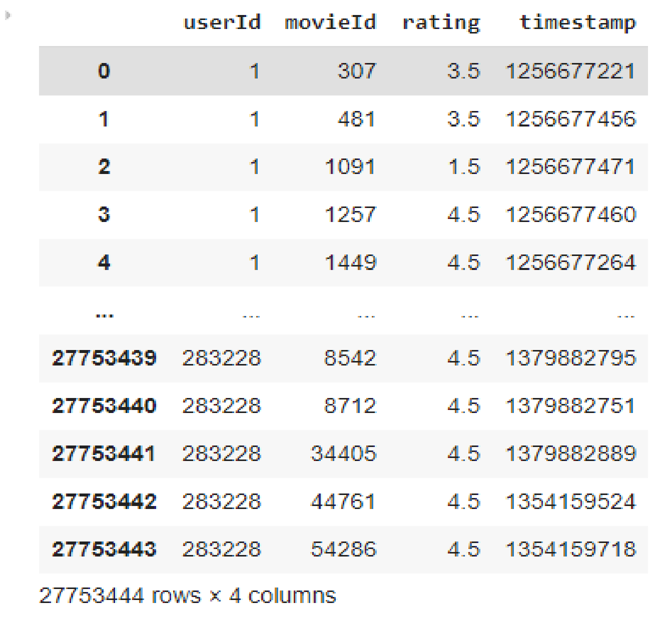
Result of MAE = 0.90093

After seeing the results of these we can conclude that results are very less deviated compared to original results.

Singular Value Decomposition

SVD is the state-of-the-art non-deep solution to the recommendation problem. Netflix uses this exact system to make recommendations, and we can leverage this powerful technique to make great recommendations.

The first thing to do is to get our interaction data in the correct format. We are given a dataFrame like so:



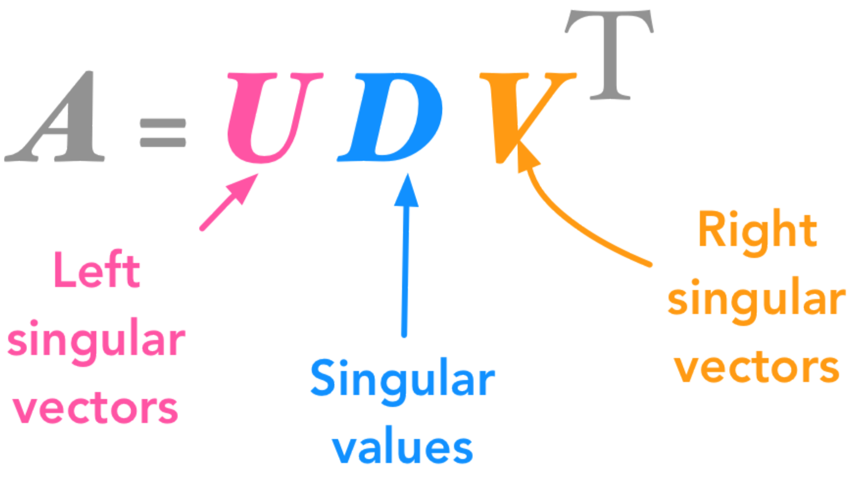
	userId	movieId	rating	timestamp
0	1	307	3.5	1256677221
1	1	481	3.5	1256677456
2	1	1091	1.5	1256677471
3	1	1257	4.5	1256677460
4	1	1449	4.5	1256677264
...	...	...	...	...
27753439	283228	8542	4.5	1379882795
27753440	283228	8712	4.5	1379882751
27753441	283228	34405	4.5	1379882889
27753442	283228	44761	4.5	1354159524
27753443	283228	54286	4.5	1354159718

27753444 rows × 4 columns

What we need is a pivot table of this data. A matrix I that MxN, where M is the number of users and N is the number of movies. The ith row and jth column tell us the rating user I gave to movie J.

Naturally, this gives us very sparse data, as the majority of users have rated a fraction of the whole universe of movies. And this is where we make an optimization of the standard SVD algorithm, inspired by Simon Funk's Netflix solution. This allows us to use the entirety of the data and process it in a reasonable time. This solution is orders of magnitude faster.

But first, we consider what SVD does. It decomposes a matrix:



A in our case is our interactions data, U would represent our user space and V represents our movie space. Obviously, a perfect one doesn't exist, so SVD aims to minimize the MSE of this operation. In standard form this looks like:

$$min(A_{ij} - \langle U_i, V_j \rangle)^2$$

Note that the right part is a dot product between row i and column of matrices U and V respectively.

With 58000 dimensions, this can take quite long. But using our sparse nature, we only consider non-zero dimensions. A simple but effective speedup.

This is a supervised learning algorithm, actually, with training, validation, and testing data. So we split our interaction data BY USER. This is critical, as we do not want part of one user to be in another sample, as this would corrupt our test set. After training, we predict, and our results are excellent. When predicting the ratings of users the machine had never seen before, we get a mean absolute error of 0.75 and an MSE of 0.94. This tells us that the machine can predict what a user will rate a movie within 0.75

points. With this, we can recommend movies quite easily, as the rating user I will give movie J is given by:

$$R_{ij} = U_i * V_i * \theta$$

Where theta is the diagonal part of D as shown above, and U and V are calculated from the SVD process.

Overview and Future Thoughts

Overall, collaborative methods far outperform content-based methods. For the future, we would want to try other collaborative methods that involve machine learning, like LSTM's and skip-gram since they also utilize timestamps. Also, deep methods may be able to better encode information than SVD - which is our current best performing model.

Content Methods likely didn't work well because we did not have good features. Different, more targeted features like the act structure or the amount of action/emotion might lead us to better features and hence, better performance.

Our SVD algorithm can be combined with content-based to make recommendations for any movie aggregator website like IMDB, or build a new website that recommends movies. Netflix and Amazon and such require a commitment - this can be used to make a third-party website. A user can either find movies similar to one (content). Or a user can choose movies and give ratings to them. And based on that historical data, our website can recommend movies.