Movie & TV Show Recommendations

Sarang Gawane, Sanjay Kalidindi, Mirac Kara, and Tyler Limbach
UIC - CS412

	NetID	Question asked to (Group #)	Question answered from (name)
Sarang Gawane	sgawan2	9	Jiangni Ren
Sanjay Kalidindi	kalidnd2	16	Pradhan Bangalore Suresh
Mirac Kara	mkara3	1	XX
Tyler Limbach	tlimba2	10	Haoxuan Zeng

1 Summary of Problem

In 2021 most individuals living in the United States have used or are currently using some sort of streaming service to watch movies and TV shows. There are a large variety of platforms on the internet that allow for consumers to do this such as Netflix, Prime Video, Hulu, and HBO Max. In fact, studies show that 70% of millennial's use a streaming service on a weekly basis and 40% confirm that they use them on a daily basis.

Due to the rapid growth in online streaming, a common problem many individuals have is determining which movie/show to watch next. Our project aims to implement a recommendation system using Machine Learning to determine similar movies/TV shows using a variety of features including genre, director, cast, and IMDB rating.

2 Approach

Content based filtering and collaborative filtering are two techniques that can be applied to help build this recommendation tool. Collaborative filter is a technique used by large streaming services such as Netflix and Amazon which makes automatic predictions for users by collecting data from the other users using their platforms. Collaborative filtering operates under the assumption that users who agreed in their assessment of a certain movie/TV show will likely also agree in the future on another movie/TV show. If you have used a streaming service, you might recall that underneath the descriptions of movies/TV shows there is usually a snippet of text that states the following "98% of people who liked this movie also liked **blank**." This is an example of collaborative filtering.

Content based filtering on the other hand is a technique which uses a user's previous history to provide a recommendation. Unlike collaborative filtering it does not use data obtained from other users, just data from a single unique user. Content based filtering will use key-words and features from a user's input to output a recommendation.

For our project we chose to use a content based filtering approach simply cause we don't have a platform which gives us access to the data of a large number of users.

In the next section we will take a look at the flow chart for our project and discuss a bit more about how we tackled the code for it.

3 Flowchart

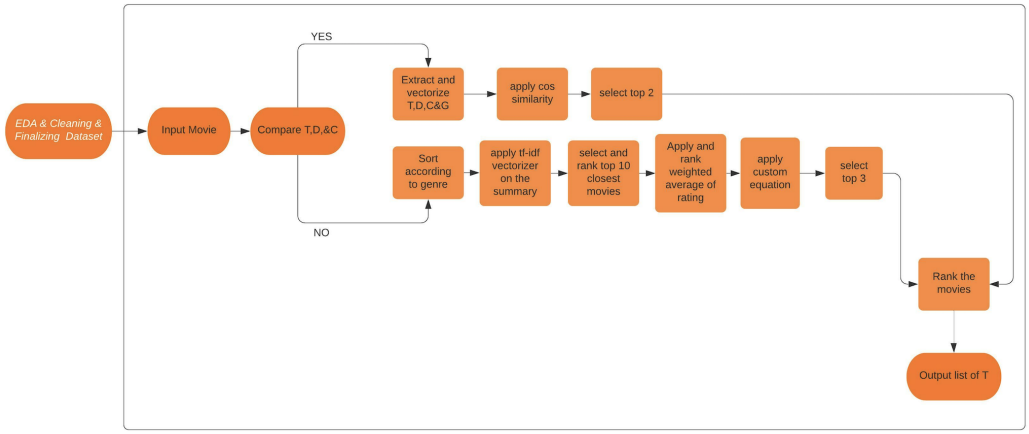

Figure 1: Flowchart of Movie/TV show recommendation system

3.1 EDA, Cleaning & Finalizing Dataset

After determining what data-set we wanted to use, the first thing we did was clean the data and perform some Exploratory Data Analysis(EDA) to better understand our data and get some clean visuals of their characteristics. The data-set we ended up using for this project originally had approximately 43,500 data-points, but after our cleaning it had approximately 10,000 data-points. The reason for the large reduction in the number of points we used was because many of these data-points had features with missing values (NaN). And because these are largely categorical features, it is not something we can estimate based on the remaining data. In addition to this, the large number of data-points was making it difficult for us to scale, so as a result we decided to remove the movies/shows with a rating below 5.0. We will discuss this a bit more in detail later on in this section.

When performing the EDA, the first thing we wanted to take a look at was the genre split of our data-set. As you can see below, our data-points can be grouped into 20 different categories based on genre. The most popular being drama, comedy, and thriller. The complete split of the genre within our data-set can be seen in Figure 2 below.

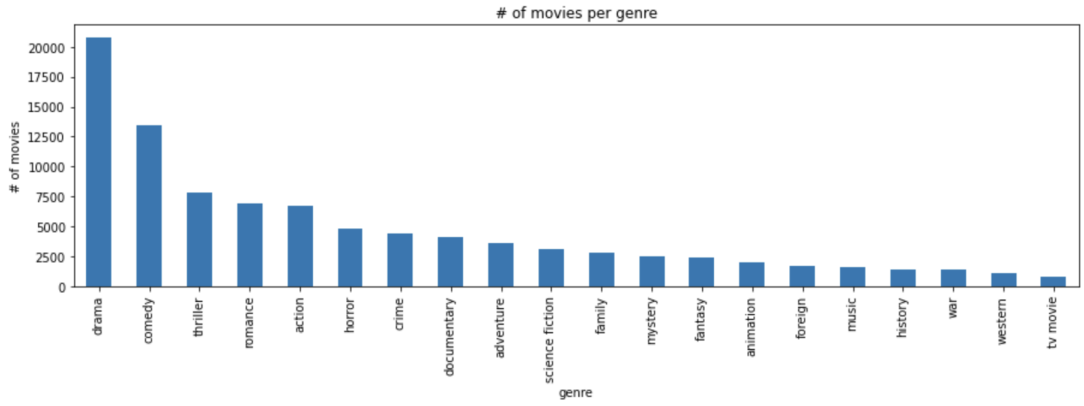

Figure 2: # of movies per genre in data-set

Next, we took a look at the IMDB ratings characteristics of all our data-points. Of the 43,503 data-points in our data-set, the mean IMDB rating was 6.014, with a standard deviation of 1.257. The median IMDB rating in the data-set was 6.100. Figure 3 below illustrates some other characteristics of the data-set including the min, max, 25% rating, and 75% rating. As a part of the cleaning of the data, we decided to remove all movies below a rating of 5.0. We did this for two reasons. Firstly because the large data-points we had of 43,503 was giving us problems when trying to scale when we used the CountVectorizer() function. To mitigate that issue we believed it would be appropriate to remove the movies with less than a rating of 5.0. Additionally, movies with a rating of less than 5.0 indicates that the audience typically didn't find the movie to be good, which therefor means that the likelihood of them wanting to recommend it would also be lower.

```
count    43503.000000
mean         6.014325
std          1.256845
min          0.500000
25%          5.300000
50%          6.100000
75%          6.800000
max         10.000000
Name: average_vote, dtype: float64
```

Figure 3: Average IMDB ratings for data-points

In figure 4 we have a plot of the number of movies from each year dating back to 1874 up until 2020. Due to the large quantity of values we have located on the axis, the visual is a bit difficult to clearly read. The visual is primarily illustrating that within our dataset, the bulk of the movies/shows are from the 21st century, and greatly outnumber those from the 19th and 20th centuries in this dataset. Initially we had considered this feature to be used as a filter to reduce the size of the data-set after we realized that the CountVectorizer() wasn't scaling higher than the linear rate and there was a lot of redundant data. So therefore we chose the top 10 years with the max count of movies per year.

However, this led to another problem, which was that it skipped the consecutive years, which meant that the obvious prediction of sequels and prequels for a series/franchise would not be included in the recommendation being outputted. This is clearly not a good sign for our system. As a result, we decided to use the popularity filter instead. This was a combination of both critical reviews and public reception. This was our final step to cut down on the size of the data set, and it had the benefit of trimming entries that less likely to be provided as user input, or to be recommended as output. Using the rating of the movie as a filter we were able to cut down the number of data-points in the data-set significantly, while still maintaining the qualitative nature of the predictions the same.

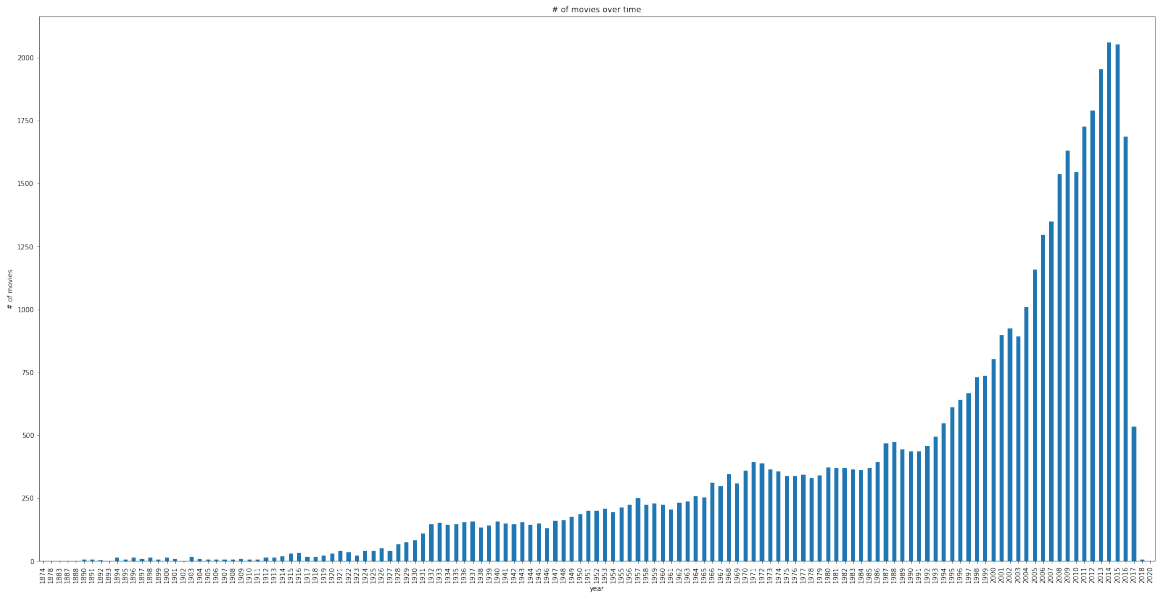

Figure 4: Movies produced by year

3.2 Compare T, D, C & G (Yes)

This portion makes up the top part of the flowchart in figure 1 and is simply just an easier way for us to track similar movies or shows from the same franchises as it saves us the effort of comparing the entire synopsis for the movie/show. We believe that there is an element of subjective similarity it captures due to ideas like cinematographic signatures unique to certain filmmakers. This portion produces a sparse similarity matrix, using cosine similarity as described below. This provided us with a base similarity score between each pair of movies.

3.3 Cosine Similarity

Cosine measurement is a technique that is used to quantify the similarity between two or more vectors and is widely used in recommendation systems. It can be simply defined as the measure of the cosine of the angle between two non-zero vectors. Cosine similarity is a value between 0-1 and a value of 0 occurs when $\cos(\theta) = \cos(90)$, which indicates that the two vectors are perpendicular and are not similar at all. A value of 1 can be obtained when the two vectors are both on the same axis and $\cos(\theta) = \cos(0)$, indicating that the two vectors are identical. For all values in between, as the value gets closer to 1 it indicates that the vectors are more and more similar. The visual below illustrates how cosine similarity measures the similarity between two vectors.

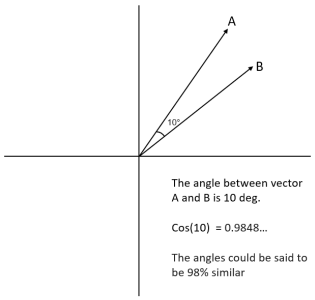

Figure 5: Example of cosine similarity[7]

The formula we use to compute the cosine similarity between two vectors can be found below.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Figure 6: Formula to compute cosine similarity[6]

For our data-set, we extracted the features Director, Star1, Star2, Star3, Star4, and Genre for each movie/show from our data-set and transformed them into vectors using CountVectorizer(). Once we had a vector for each movie/show and each feature we used the cos similarity to compare the similarity between each of these vectors.

3.4 Compare T, D, C & G (No)

For the other branch, we computed multiple sparse matrices to be combined with the base similarity matrix we obtained above. Following similar methodology, we obtained sparse matrices to measure the similarity of genres across movies. This used a CountVectorizer and cosine similarity as well. We also obtained a sparse similarity matrix for the descriptions of each movie pair, using a TfidfVectorizer and cosine similarity. These matrices were added together element-wise, with genre weighted twice as heavily as the description matrix. Finally, the similarity matrix obtained from 3.2-3.3 was multiplied element-wise with the weighted matrix obtained in this step.

3.5 Release Year

We obtained a normalized matrix of the distances between the release years for each pair of movies. This matrix was subtracted from the result of 3.4, applying a minor penalty based on how far apart the movies were released.

3.6 Popularity & Ratings

We obtained normalized vectors for both the popularity and ratings of the movies in the data set. Due to the popularity vector's large range and the presence of outliers on the high-end, we added a ceiling of 50.0 popularity to the normalization. Any value above 50.0 was given a 1. The ratings vector was normalized by simply dividing the score by 10, to get each score between 0 and 1.

These vectors were multiplied together to obtain an opinion vector for each movie. The matrix resulting from 3.5 was multiplied row-wise with the opinion vector to weigh recommendations based on their ratings and popularity. This result is our finalized similarity matrix.

3.7 Final Recommendation

All of the steps prior to this are computed in advance, so the recommendation process is nearly instantaneous. Our final recommendation takes a title as input and finds the corresponding row in the similarity matrix for that movie. The scores in that row are sorted from ascending to descending order. Finally, we iterate through the sorted list and obtain the corresponding movie title for the top 10 recommendations. The top result is always excluded because it will always be the same movie as the input.

4 Evaluations

Evaluating a movie recommendation is somewhat difficult due to the result being opinionated. There is no correct answer to a recommendation being good or bad. What one individual might consider an excellent recommendation, another individual might consider a poor recommendation. With that being said, we can still measure the quality of our recommendations by looking to see whether certain

movie inputs return prequels or sequels and whether we recommend the accompanying entries in the series. Another good measure would be to inspect if the recommended movies are off the same genre. For example, if we were to recommend *Interstellar*, which falls under the genre's adventure, drama, and science fiction, we expect the output/results to all be within similar genre's. This tends to be a natural similarity we can detect intuitively as well.

Due to the fact that our algorithm already measures objective similarities between movies, another way we evaluated performance was subjective test cases. We encountered the same limitation that prevented us from using collaborative filtering: we didn't have access to data that mapped user ratings across multiple movies. Our next option was to ask ourselves and others to provide a movie they like, and to judge the top recommendations themselves if they had already seen them. This is one way we determined our initial algorithm using only sections 3.2 and 3.3 was unsatisfactory. After adjusting weights and adding in section 3.4, we did the same manual testing and had much higher approval among testers.

5 Results

In our original testing, we weighted the directors and cast more so then the title of the movie itself. This was causing movies that were not a part of the given "universe" of movies to be returned. We used two famous "universe" movies as test cases. The first being *Marvel's Avengers* and the second being *Star Wars*. In both scenarios, after weighing the titles of the movies higher than the director, our recommendation returned included either sequels or prequels to the searched movie. After recommending the accompanying set to the series of movies, we recommend similar movies based upon genre and cast. Another test case we used was *Interstellar*. When we searched for this title, we received results of movies created by the same director and in the same genre. Searching a movie like "Monsters, Inc." provided other animated kids movies as output.

There were still some cases left where recommendations seemed off when testing the algorithm after section 3.4. For example, movies released 50 years prior to the input would be recommended because they closely matched the highest weighted features of the input. These movies have many dissimilarities that aren't captured in the other features, such as graphics and colloquialisms. This was remedied by creating a normalized distance matrix between release dates for all movies, which was subtracted from the similarity score matrix. Movies more recent to each other weren't impacted much, and more distant movies were pushed out of the top recommendations.

An example of the output we obtained for an input of *Avengers: Age of Ultron* can be seen in figure 7 below.

```
Recommendations similar to Avengers: Age of Ultron :
1. The Avengers
2. Iron Man 3
3. Iron Man 2
4. Captain America: Civil War
5. Iron Man
6. Guardians of the Galaxy
7. Star Trek Beyond
8. Transformers: Age of Extinction
9. Star Trek Into Darkness
10. Captain America: The Winter Soldier
```

Figure 7: Example output of recommendation system

6 What We Learned

This final project taught us that finding a proper data-set and appropriately cleaning it for our use case is crucial. Originally for this project we used a Kaggle data-set that contained the 1000 most popular movies on IMDB. We thought that 1000 movies would be sufficient to provide good recommendations, but as we continued without our project and began testing, we realized that 1000

movies was a pretty tiny sample of the overall collection of movies out there. For example, in 2019 alone, approximately 792 movies were released in Hollywood alone. That is for just one year, and doesn't even take into account all the different film industries around the world. So as you can see, 1000 movies is not a large enough sample size to make accurate predictions. After doing some more searching for a new and larger data-set we came upon one that contained a collection of 46,000 movies. Though this movie contained more movies, we realized that this particular one would not be an appropriate fit for our project because it was just a random collection of movies, and not the 46,000 most popular movies from a particular source like IMDB. This was an issue because it did not contain the data for many widely known and popular movies such as Star Wars or The Godfather. This clearly illustrates the importance of having an appropriate data-set for what you are trying to achieve, and that the cleanest one you find during your search may not always be the best option.

In addition to having a good data-set, we also learned the importance of properly weighing certain features over others. If we over-rely on certain features compared to others, we could end up with sub optimal results. For example, originally we were weighing the directors higher than the movie name. When we did this the results returned were not movie focused but director focused. When we searched for "Star Wars Episode VII", because the director of the movie was J.J. Abrams, our output was focused on his work. When we give a recommendation based on a movie with follow-up movies, want to make sure that those are the movies that would take precedent over the directors' filmography. We also weighed the cast too heavily initially, because the data set had an extremely large cast list for each movie. Using the whole cast without weighing it less was making other features insignificant. This also created an over-fitting problem because cast members that are less significant were being weighed equally to the stars. This was fixed by trimming the cast list to the first 4 members for each movie, which were the most prominent members of the cast.

7 References

[1] https://www.kaggle.com/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows

[2] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

[4] https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421

[5] https://analyticsindiamag.com/collaborative-filtering-vs-content-based-filtering-for-recommender-systems/

[6] https://en.wikipedia.org/wiki/Cosine_similarity

[7] https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a

[8] https://scikit-learn.org/stable/modules/generated/sklearn.feature$_e xtraction.text.TfidfVectorizer.html$