Netflix Recommendation System

RUAN Yuyan
[redacted]

AHMED Ramzy
[redacted]

ÖZGÜNAY Eray
[redacted]

SENTHIL KUMAR Nikil
[redacted]

ASANATHAM Chayutpol
[redacted]

GOYAL Payas
[redacted]

LEUNG Chi Hoi Chris
[redacted]

Abstract—Big data has become a driving factor in many industries including streaming platforms such as Netflix to offer the best user experience. As the largest streaming services in the world, Netflix utilizes movies and TV shows' data as well as user's data to make personalized recommendations to its users. This ensures that users stay on their platform and use their services. In this paper we attempt to create a content-based recommendation engine for Netflix using a movie dataset from Kaggle. A user can search for a movie or a sentence and movies with similar themes will be recommended. We use NLP data preprocessing to prepare the data to be used effectively in our model. We then use vectors to represent a word that such that vectors reflect the meaning of the word. We then measure the similarity of the vectors to create fuzzy clustering model. We evaluate the effectiveness and quality of our recommendation engine using users' rating data. Our results show that the more movies a user watches, the more likely they are to like our recommendations. Users who watched more than 20 movies/TV shows would watch our recommendations. more than 55% of the time.

Keywords—fuzzy clustering, cosine distance, Word embedding

I. Introduction

Platforms like Netflix, Amazon Prime, Disney+ and more allow subscribed users to watch videos on-demand from an enormous library of TV Shows, movies and other forms of video that the platform owns. As a result, these platforms contain meta data on the videos they own (e.g IMdB rating, date of release, description, genre, name of actors, etc) and more importantly, collect large amounts of data on user viewing history. To enhance user experience and keep users engaged, technology teams at these companies have built powerful recommendation systems (and predictive models) that predict the videos that individual users would watch next based on the videos the platform currently stores and user viewing history.

Based on above, the team leveraged its knowledge to build a recommendation system that consists of two components. First, content-based filtering which recommends additional movies to watch based on the current movie selected by the user and second, a search engine which recommends movies to watch based on user's string input.

II. Implementation

The data from Netflix Kaggle challenge [1], is mostly text and word data. To be able to understand the meaning and context of the data, we need to a solution to translate the words into a form that could be understand by computer. In this project, we use word vectors to be able to determine the similarity between word and data from each movie.

Text classification is the process to classify the word by meaning and context using Natural Language Processing (NLP), text classifier can assign pre-defined tags or category to the input word based on the content and meaning of the word.

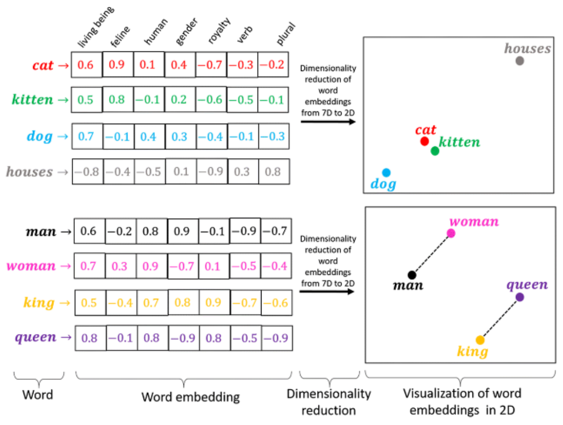
Fuzzy Clustering will be used during the classification process by measuring the cosine similarity distance and the process will be a hybrid of fuzzy clustering and collaborative filtering [2]. The utilization of fuzzy clustering in this project is to allow movies to be belong to different categories.

A. Data Preprocessing

Data Preprocessing is a crucial step to prepare the data to be used in the machine learning models. This includes cleaning the data and removing outliers. In our dataset, the raw data has features such as Movie Title, Cast, Director, Description, Genre, Rating and Playtime. Since, we are only concerned with the meaning and themes of the movie we consider only Movie Title, Description and Genre and drop the other features. The description is structured as a sentence. We start by word tokenization such that we get individual words from the sentence to be able to interpret the meaning and identify duplicates and outliers. We then remove stop words, which are words with no influence on the meaning of a sentences such as "with" and "for" that can be considered outliers. We use a technique known as word lemmatization to normalize the input and retrieve the origin of a word so different representations of the same word are not interpreted separately. For example, 'troubling', 'troubled' and 'trouble' will be interpreted as 'trouble'. After this step, we remove duplicate words to remove overfitting of our model.

The code for this project is available at https://github.com/RYY0722/Netflix-Kaggle.git

Our final step in preparing the data is to use Word Embedding, which is a text classification technique to transform a word into a vector based on the meaning, the similar the meaning of the word, the closer the vector to each other. Each word will be mapped to one vector and the values assigned to the vector are learned in a neural network. Most of the time the values of the vector are represent by real-valued with tens to hundreds of dimensions.



living being
feline
human
gender
royalty
verb
plural
cat → 0.6 0.9 0.1 0.4 −0.7 −0.3 −0.2
kitten → 0.5 0.8 −0.1 0.2 −0.6 −0.5 −0.1
dog → 0.7 −0.1 0.4 0.3 −0.4 −0.1 −0.3
houses → −0.8 −0.4 −0.5 0.1 −0.9 0.3 0.8
Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
man → 0.6 −0.2 0.8 0.9 −0.1 −0.9 −0.7
woman → 0.7 0.3 0.9 −0.7 0.1 −0.5 −0.4
king → 0.5 −0.4 0.7 0.8 0.9 −0.7 −0.6
queen → 0.8 −0.1 0.8 −0.9 0.8 −0.5 −0.9
Dimensionality reduction of word embeddings from 7D to 2D
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

Fig 1. sample demonstration of word embedding

B. *Word2vec*

Word2vec is a sub-technique of word embedding, using two-layer neural network that are trained to reconstruct context of words in form of vectors. Only the context of word is left inside the vector space, the explicit meaning of words is mostly not available but for our application, Word2vec is a suitable method because the explicit meaning of the description of the movies is not necessary for recommendation. The most important part is the contextual meaning of the description which will show the direction and topic of the movies.

In this paper, text classification started by training the Word2vec model using the large corpus of text that are open data set "text8" in addition to the data gained from Netflix Kaggle challenge. For the search engine, only the description of the movies is used and for the content-based filtering, title of the movie in the training set is added additionally. The dimension is set to 100 dimensions, to be followed up by One-hot encoding country and rating for TV shows and movie into our vector space because of the limitation of Word2vec that mostly contain only context meaning of the data, some information that have explicit meaning might be affected.

The model now ready to vectorize words data into embedding format that specified earlier and currently in the format that could be used to calculate the similarity of words.

C. *One-hot Encoding discrete features*

As mentioned, the Word2vec make the explicit meaning of word become hardly visible to the model, but some of the feature that come with the movie need the explicit meaning of particular word such as rating, country, genre and cast.

For country and rating data, it is possible to limit the number of features by setting the minimum frequency for the country data which in our project, the minimum frequency is set to be 100, and the rating of the movies is merged into 3 main categories which is 'G' stands for general audience, 'Y' stands for young audience and 'M' stands for mature audience. And with these, the amount of the features to be included is limited in the vector space. And the result will not be oriented only to these features, as one movie will consist only one or two records of country and rating.

Genre is also included in one-hot encoding, as it has explicit meaning and important when it comes to categorizing movie. Each distinct genre will be included into the vector space distinctly as there are limited number of genres for movie.

As for cast, the data for cast is very hard to set the limitation on, as there are many distinct actors and actress name in the movie. And a movie might consist of different number of casts. Prioritizing and setting the number of main cast for each movie is also hard to be implemented, lead to conclusion to not included the feature into the vector space to prevent the prior issue. These also applied to director data.

After the vectorization of word data, next important step is to come to a representation of each movie vector. In this paper, the average value of each features on every word is used in the movie's data to represent the overall context of each movie.

D. *Similarity comparison*

Cosine Similarity measures the cosine of the angle between two non-zero vectors of an inner product space. This similarity measurement is particularly concerned with orientation, rather than magnitude. In short two cosine vectors that are aligned in the same orientation will have a similarity measurement of 1, whereas two vectors aligned perpendicularly will have a similarity of 0. Cosine Similarity is used in positive space, between the bounds 0 and 1. This similarity does not measure differences in magnitude and is only a representation of similarities in orientation [3].

The cosine similarity is described mathematically as the division between the dot product of vectors and the product of the Euclidean norms or magnitude of each vector. The cosine similarity is advantageous because even if the two similar documents are far apart by the Euclidean distance because of the size they could still have a smaller angle between them. Smaller the angle, higher the similarity [4].

In this paper, we start by calculating the cosine similarity distance matrix and then we calculate the index of the movie that matches the title and then calculate the pairwise similarity scores of all movies with that movie and then finally, we use sorting to get the movie indices.

E. *Fuzzy Clustering*

Fuzzy clustering and K-Means clustering are two clustering algorithms that attempt to classify data based on their similarity. The distinction is that in Fuzzy clustering, each point has a probabilistic rather than a binary membership to clusters defined by the distance to the cluster's center. Each point is evaluated in relation to each cluster, and each evaluation requires additional steps. Just a distance estimate is required for K-Means, while total inverse-distance weighting is required for fuzzy clustering means. For our content-based recommendation system, a fuzzy clustering approach has been proposed, and its success against various clustering methods has been evaluated.

Calculating the distances among 7787 movies is very computationally intensive and costs lot of memory and time. Clustering have been used to break the data into several blocks. To be noticed, using fuzzy clustering instead of standard k-means to allow one movie belong to several groups because that makes more sense. In the Figure 1 below, the data set is not classified. When applying the fuzzy clustering to the data set, it can allow one movie belong to several groups and it reduce the calculation to only the related cluster instead of all the data. After the fuzzy clustering algorithm is applied, we can see in Figure 2 that the data is spitted out to several evenly distributed blocks, which is satisfactory for us to reduce the searching area and cost less memories when execute the program.

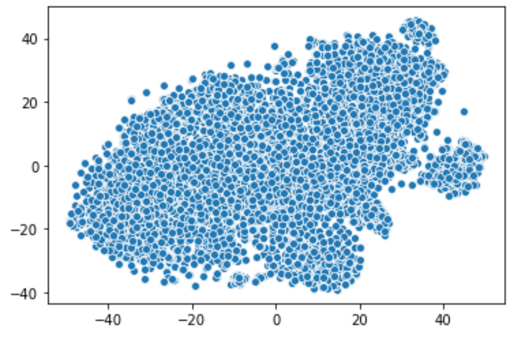


Fig. 2 Data set before fuzzy clustering

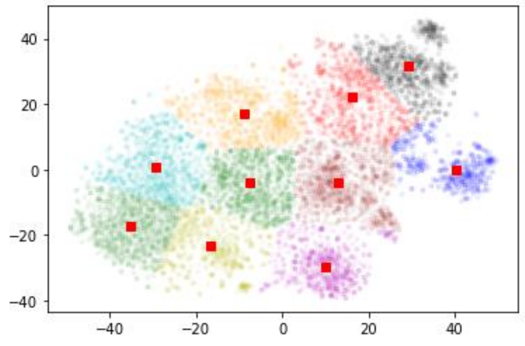


Fig. 3 Data set after applied fuzzy clustering

III. Result and Evaluation

A. Functionality

The project presents a recommendation system and a simple search engine. The recommendation system will select and recommend the most related movies to what the user is watching. The search engine will take in a string input by the user and return some recommendations.

B. Dataset and Procedure

In this project, cosine similarity is the metric for the similarity between users. However, the score, i.e., the cosine similarity, is not the best to evaluate the performance when used alone. This is because the features are extracted and represented numerically, and the similarity calculated from such features may be different from human judgments. It means that even the cosine similarity is high, the similarity from human perspective is not guaranteed. As a result, users may not be satisfied by the recommendation thus not clicking into the recommended movies or not giving high marks to them.

Therefore, another dataset is included to evaluate the model, which is the Netflix Prize Dataset [5], an open dataset for developers to design algorithms to predict user ratings for films based on previous ratings. Three features of the original data are included: userID, itemID, rating.

The detailed procedure is as follows.

1. Create like_list for every user. Movies rated over 3 by the users are added into the like_list.
2. Input half of the like_list and get the recommendations.
3. Evaluate the recommendations based of users' watching and rating histories.

C. Results

After preprocessing, it is found that there are 635 movies are rated by the users, and the watch history of 296000 users is used to do the evaluation. To be noticed, since total number of movies that each user has watched differs greatly. Some of them have only several entries while others have hundreds of records. The importance of this is that intrinsically, the larger the input size is, the better the recommendation system will work. Therefore, the users are divided into 4 groups based on this, and the corresponding intervals are [1,20), [20,50), [50,100) and [100, inf). Evaluation of these groups is done separately and compared later.

To further investigate the performance of the model, several additional metrics are introduced.

1. hit rate: A hit is defined as a user watches a recommended movie. The hit rate, or click through rate, is thereby the number of hit divided by the total number of watched movies. This figure shows the effectiveness of the recommendations, alternatively, how well the model can predict the users' behavior. This can be of commercial value since this indicates how heated the item is, which may attract commercial advertising.
2. hit like rate: a hit like is defined as a recommended movie is rated over 3. The hit like rate is the number of hit like divided by the total number of watched movies, which is a good indicator of the quality of the recommendation.
3. average input and output ratings: as mentioned, half of the like_list is fed into the system, average input rating is calculated by the average rating of these actual input movies. The output rating is then calculated by the average output rating. If the recommended movie is not watched by the user, the rating for it will be set to 0.

Since the part of the like_list is the actual input of the system, the performance is plotted and compared versus the number of the liked movies instead of the total number of watched movies.

The trend of hit rate and hit is shown in Figure4, it is noticeable that the hit rate and hit like rate increase greatly after the number of movies liked reaches 20. Referring to TABLE I, hit rates is round 60% and hit like rates reaches more than 20% when the size input is relatively large.

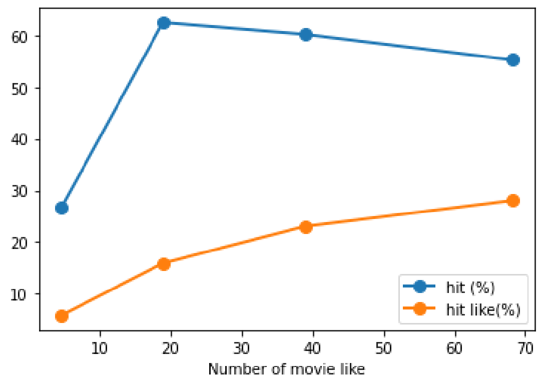


Fig. 4. Hit rate and Hit like Rate versus Number of Liked Movies

TABLE I. Hit rate and Hit like Rate versus Number of Liked Movies

total_ watched	*total_ watch_avg*	*No_like*	*hit(%)*	*hit like (%)*
[1,20)	7.61	4.62	26.64	5.80
[20,50)	31.90	18.93	62.48	15.88
[50,100)	69.52	38.92	60.19	23.02
[100,inf)	132.97	68.14	55.25	28.01

The average rating versus number of liked movies is shown in TABLE II. Note that the average input rating is calculated from the like_list, all of which are rated 4 or 5, so the input rating is relatively quite higher compared to the average output rating. When the user does not have many history records, the recommendation result is bad: the average output rating is only 1.55. However, when the input size is relatively large, the average output rating becomes satisfying and stable, around 3.6 to 3.7.

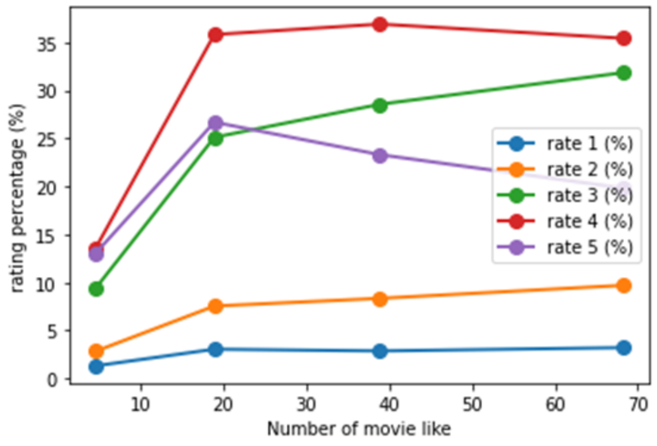


Fig. 5. Detailed Rating Distribution versus Number of Like

TABLE II. Average Rating versus Number of Liked Movies

total_ watched	*total_ watch_avg*	*No_like*	*avg_in_ rating*	*avg_out_ rating*
[1,20)	7.61	4.62	4.10	1.55
[20,50)	31.90	18.93	4.37	3.70
[50,100)	69.52	38.92	4.34	3.69
[100,inf)	132.97	68.14	4.31	3.59

The detailed distribution of the rating is shown in TABLE III and the trend is visualized in Figure 5. One significant feature is that the percentage of movies getting rate 1 and 2 is low, less than 10%. Rate 4 takes up the greatest part among the 5 ratings, around 35%. Plus, there are around 20 – 30 % rate 5 and 3 as well. In conclusion, the overall recommendation result is satisfying with a lower proportion of unsatisfying recommendations. The numeric values are displayed in TABLE III.

TABLE III. Detailed Rating Distribution versus Number of Like

total_ watch	*total_ watch_avg*	*No_like*	*rate 5 (%)*	*rate 4 (%)*	*rate 3 (%)*	*rate 2 (%)*	*rate 1 (%)*
[1,20)	7.61	4.62	13.05	13.59	9.41	2.84	1.32
[20,50)	31.90	18.93	26.68	35.80	25.11	7.55	3.06
[50,100)	69.52	38.92	23.30	36.89	28.55	8.35	2.87
[100,inf)	132.97	68.14	19.83	35.42	31.82	9.70	3.22

IV. Further Improvements

So, as can be seen in the Evaluation part, our model gives pretty good movie recommendations. But still, some further improvements can be made.

A. Tuning the Weightings of the Features

Firstly, we can tune the features' weightings because we mixed categorical features and numeric features together, and each feature has a different meaning, we may try to give the features different weightings to optimize the model.

B. Use Bigger Data From Netflix

Secondly, we can train the model with much bigger data from Netflix. Since the data is huge from Netflix, this is computationally intensive, and it will take a lot of time and computational power.

C. Use Personalized Data for User-based Recommendation

Thirdly, recommendations for specific users can be made using the user's watching history and the ratings he or she gave. It makes the recommendations personalized, and that leads to personalized results.

V. Conclusion

To conclude, we created a content-based recommendation engine for Netflix using a dataset from Kaggle to recommend to users a movie given a movie name or a sentence. In doing so, after applying preprocessing to the data set, we have utilized word embeddings, including the state-of-the-art model word2vec. Then, we used fuzzy clustering to the word embeddings of movies by measuring the similarity between vectors. Finally, we have found that user response and rating to such content-based filtering recommendations can be considered good because we achieve a 50-60 % hit rate and around 3.6-3.7 average output rating. In the future, we would like to try improving our solution to get better recommendations and rates by using three methods: tuning the weightings of the features, using more extensive data from Netflix, and using personalized data for user-based recommendations.

VI. References

[1] Retrieved from https://www.kaggle.com/aalhendi/netflix-movies-and-tv-shows-ratings/data

[2] J. Wang, N. Zhang and J. Yin, "Collaborative filtering recommendation based on fuzzy clustering of user preferences," 2010 Seventh International Conference on Fuzzy Systems and Knowledge Discovery, Yantai, China, 2010, pp. 1946-1950, doi: 10.1109/FSKD.2010.5569467.J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] DeepAI. (2019, May 17). Cosine similarity. Retrieved April 17, 2021, from https://deepai.org/machine-learning-glossary-and-terms/cosine-similarity

[4] Prabhakaran, S. (2020, October 11). Cosine similarity - understanding the math and how it works? (with python). Retrieved April 17, 2021, from https://www.machinelearningplus.com/nlp/cosine-similarity/.

[5] Retrieved from https://www.kaggle.com/netflix-inc/netflix-prize-data