YouTube Recommendation Algorithm

Nguyen Vi Cao
CSS 486 Machine Learning
Professor: Dong Si

University of Washington Bothell
cnv3103@uw.edu

ABSTRACT

YouTube is now owning one of the most sophisticated and biggest scale Recommendation System in the social media platforms. This paper will cover a wide perspective of how the algorithm has developed from its starting point, try to describe the system on a high level as well as identify its close relationship with machine learning by having two main methods: Collaborative Filtering and Classification. Furthermore, there will be a slight insight to a smaller scale Recommendation Algorithm which can be used to have a better understanding of how YouTube Recommendation Algorithm works when it was first introduced.

APPENDIX

1. INTRODUCTION

Recommendation algorithm is a crucial aspect that almost every social media platform needs in order to compete with other competitors on the market. It is crucial because it fulfills the expectation of both the platforms and the users. Social platforms will want their users to keep using their platforms as long as possible; on the other hand, users need to be entertained, if not, they will stop using it or switch to another platform.

Recommendation system comes from a very interesting concept of basic human needs. For instance, we want to receive gifts that are either meets our expectation or our needs without saying the specific gift out loud. People would have to gather information to be able to make a guess of what that gift should be. That is basic human needs because we want others to care about us and the "gathering information" is the proof of caring.

For YouTube recommendation algorithm, it has been developed so that it can bring benefit to the platform by keep refreshing the interest of the users. The process of gathering information include multiple variables from the users such as: subscription, click, watch time, content, sharing, like, dislikes, etc. Over the development process of YouTube since its recommendation system was first introduced in 2008, the algorithm has grown significantly not only with what being fed to the algorithm but also the direction and the purpose of the algorithm has also changed over time which will be discussed later on in this paper.

2. GROWTH OF ALGORITHM

YouTube recommendation system has made some progress since it was first introduced in 2008. Back then, the algorithm is being fed with only one input which is the ranking of videos based on popularity. It was a good start since it met the majority of the users' needs. However, it had quickly became insufficient with the raise of users and variety as well as the diversity of users' interests had also increased.

Later on in 2011, the algorithm added one more variable to it other than the ranking of the videos which was the user's click. The "click" variable will store the user's behavior of what video did they click onto to add another layer of diversity to the system. However, a flaw of this "click" variable is that users can

mistakenly click to the video and that data got fed to the algorithm which makes it went off track. In 2012, "watchtime" variable was added with the purpose of fill in the flaws for its previous variable. In the YouTube Official Blog, they stated that they immediately realized a 20% drop in views after the "watchtime" was added but they believed that this was the right way to do it. Not only had they thought of increasing the number of variables to add to the algorithm, but they also went ahead and added some precautions steps to filter the videos that were published. Therefore, racy, and violent video would be prevented from being showed up as recommendation to the users.

From that point, there were more and more variables were added to make the algorithm as precise as possible. To predict the watch behavior of the users, survey responses, sharing, likes, and dislikes had made their way to the lineup. These variables will certainly help the system to improve its preciseness of predicting the users' behavior; hence, create more values to both the users and the platform. (YouTube Blog Official: On YouTube's recommendation system)

3. CHALLENGES

YouTube recommendation system is a one of a kind in its area. Unlike other recommendations from social media like Facebook, Instagram; or music recommendations from Apple Music, or Spotify, these are considered lower scale then YouTube. While other recommendations algorithms only take in around ten inputs include inputs from the users, recommendation objects, and calculate the similarities; YouTube recommendation has to deal with a corpus of millions of videos with various length. That is only one example that shows us how the YouTube recommendations algorithm has its own uniqueness and how challenge it is to handle.

However, we can trim it down to three major problems. According to "Deep Neural Networks for YouTube Recommendations" paper, those are:

- Scale: similar to what has been mentioned before, many existing recommendation algorithms doesn't work effectively with the expectation of YouTube in order to highly individualize the recommended video to the users. With that being said, the algorithm needs to be optimized to be fed with an enormous amount of information from the users' preferences inputs and the corpus of videos.
- Freshness: this is a sub problem of Scale that has to do with the video lengthy. With the variety of each video length, it can be short (YouTube Shorts), to a length of a song (Music Video), to a Vlog or even movie long, etc. The algorithm needs to be effective and sensitive enough to take in hundreds of videos per minute, filter, identify and categorize them.
- Noise: noise is an unpredictable variable appear from both the users and the videos. Videos that have the same content do not mean that it will fit perfectly with every user, some might enjoy it and others might not. There is no specific scale that can measure the satisfaction of the users with the recommended video. While video produces are trying to be creative with their videos, it accidentally creates conflict with the metadata associated with content from previous videos that we have collected; hence, create more and more noise.

4. RECOMMENDATION: COLLABERATIVE FILTERING

4.1. EXPERIMENT SETUP

Let's first tackle this problem by taking small step one at a time. Let's say we are trying to build a recommending algorithm on a YouTube platform where it has 5 videos only (we will number it from 1 to 5) and 5 users (we will call them A through E) as our collected data to work with. Each cell in this 2D array will hold a value, that is 1: if they have watched the video and hit like; -1: if they have watched the video and hit dislike the video; and 0: if they have not watched the video and therefore have not rate the video as well. For the sake of this experiment, let assume that after the users have watched a video, they will have to rate a video, they cannot watch and do not leave either a like or dislike. We have got all of the information setup, let's form a table:

Video / User	1	2	3	4	5
A	1	-1	0	0	0
B	-1	0	0	1	1
C	-1	0	1	-1	0
D	0	1	**?**	-1	-1
E	1	1	-1	0	1

Table 1

Our goal here is to go through the calculation process and try to predict whether user D after watching video number 3 will either like or dislike the video? There are two approaches that we can apply to this problem: one is User-user collaborative filtering and two is Item-item collaborative filtering.

4.2. USER-USER COLLABORATIVE FILTERING

In User-user collaborative filtering, the method will take advantage of the similarity calculations, in this case is the cosine similarity formula, between the 5 users in the data.

$$\text{Similarity } (A,B) = \cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||}$$

It is convincing that people with high similarity are more likely to either share the same interests in a specific area and vice versa. After getting those similarity values, we will use those to try to predict whether user D will like or dislike video #3.

4.2.a. SIMILARITY BETWEEN USERS

In Table 1, each user interacts with the five videos and those interactions are represented as values in each row. Each row of values is called one User Vector. The output we want to produce will be the cos(θ) which will be in the range of -1 to 1. The smaller the angle θ between the two vectors, or the closer the value to 1, the more similar the two users are and vice versa, the bigger the angle θ between the two vectors, or the closer the value to -1, the more dissimilar the two users are.

However, we also want to take in account that not every user will be ease at rating a video compared to others. Hence, we will add another layer of complexity to the cosine similarity formula by adding one more step which is to normalize the data to represent each user critics. This normalized step can be computed by calculating the mean for each user and then subtracting it from the corresponding ratings in Table 2. For example, the mean of user A is (1+(-1))/2 = 0 and then we will subtract the mean with each value of like or dislike so that value for user A interacts with video #1 is 1-0 = 1, video #2 is -1-0 = -1. We will do that for all other users and the result will be showed in Table 2. By doing this, we have brought the average rating for each user to zero. Now, with all of the normalization calculations has been clarified, we will proceed to calculating the Cosine Similarity of user D compared to other users (Cos Sim) on the last column of Table 2.

Mean	Video User	1	2	3	4	5	Cos Sim
0	**A**	1	-1	0	0	0	-0.578
0.33	**B**	-1.33	0	0	0.67	0.67	-0.33
-0.33	**C**	-0.67	0	1.33	-0.67	0	0.165
-0.33	**D**	0	1.33	**?**	-0.67	-0.67	N/A
0.5	**E**	0.5	0.5	-1.5	0	0.5	0.119

Table 2

These numbers have given us 2 important sets of information:

- How criticize each user are when they interact with each video. From the mean column, we can easily see that user B and user E are more likely to give video a like after they have watched it, that is why their mean is positive. Hence, if they tend to give any video a dislike, it should be weighted more heavily than the others. It also works the other way around with user C and user D. These two people are pretty strict with their like buttons (these are represented by their negative mean), and so if any video is convincing enough to receive a like from them needs to be weighted more heavily than the others.
- How similar and dissimilar each user is compared with user D. We can see that User A and user D have some conflict when they rate video #2, while A dislikes it, D likes it. Same situation with user B and user D, while B likes video #4 and #5, D dislikes them. From that, we are able to conclude that User C and E have some similar with user D while A and B do not.

4.2.b. PREDICTION

To make prediction on whether user D will like or dislike video #3 after watching it, we would need to take the weighted average by calculating the Predicted Rating by the following formula:

$$P_{D,3} = \bar{r}_D + \frac{(Wc,d)(Rc,3)+(We,d)(Re,3)}{|Wc,d|+|We,d|}$$

$$= -0.33 + \frac{(0.165)(1.33)+(0.119)(-1.5)}{0.165+0.119}$$

$$= 0.33 + 0.144 = -0.186$$

With video #3, there are only two data that are available for us to work with which are user C and E; hence, the value that we need to perform the Predicted Rating related to video #3 of three users C, D, and E are: the cosine similarity of user C, D; the cosine similarity of user E, D; rating of user C with video #3, rating of user E with video #3. Notice that, the average rating of each user with each video after normalization process is zero; therefore, the value that we calculated above, $P_{D,3}$ = -0.186 which is smaller than 0 predicts that user D is likely to give a dislike to video #3. If we were going to predict what would be the rating for video #1 of user D. After doing the calculation, we will have the value $P_{D,1}$ = -0.305 which indicates that user D will possibly give a dislike to video #1.

To sum up what we have done so far, if we were going to perform the same calculation for every 0 value in Table 2 which means that the video has not been watched by the users. We will have some scientific predictions based on the data collected on whether that specific user, after watching the video, will either have positive feedback or negative feedback toward that specific video. From that information, we can decide whether we should recommend the video to the user or not. In the previous example, since we have predicted that the rating of user D on video #3 is negative -0.186, and video #1 is -0.305, it is more likely that the user D will not like the videos; hence, we will not recommend those videos to user D.

The benefit of the User-User Collaborative Filtering is that the calculation is pretty straightforward and easy to perform. However, its downside is that the algorithm does not perform well on a big scale. The User-User Collaborative Filtering needs to be proceeded through 2 steps: 1. Determine the similarity between the targeted user with other users; and 2: Make prediction based on the similarities. As we can see the first step is very costly because it is heavily depended on the number of accounts, the larger the number of users is, the more time consuming and delay it will get to even compute through this first step. Therefore, we have come to a solution to the scaling problem which is to approach the problem on the Item aspect, in this case is the video object instead of the users.

4.3. ITEM-ITEM COLLABORATIVE FILTERING

4.3.a. SIMILARITIES BETWEEN ITEMS (VIDEOS)

For the Item-Item Collaborative Filtering approach, we would still need to proceed with two steps: calculating the cosine similarities (Cos Sim) between the videos and then calculating the predicted rating. Let's keep our task from the previous example, predict whether user D will like or dislike video #3. First, we need to calculate the Cosine Similarity between the other 4 videos compared to video #3 as well as the average weighted rating in table 3:

Video User	1	2	3	4	5
A	1	-1	0	0	0
B	-1.33	0	0	0.67	0.67
C	-0.67	0	1.33	-0.67	0
D	0	1.33	?	-0.67	-0.67
E	0.5	0.5	-1.5	0	0.5
Cos Sim	-0.436	-0.215		-0.383	-0.353

Table 3

4.3.b. PREDICTION

After we have computed the Cosine Similarities and the average weighted rating, we have got enough information to calculate the Predicted Rating for user D with video #3 by considering all items that has been rated by user D:

$$P_{D,3} = \frac{\sum video\ W video,3 * Rd,3}{\sum video\ W video,d}$$

$$= \frac{(-0.215)(1.33)+(-0.383)(-0.67)+(-0.353)(-0.67)}{-(0.215+0.383+0.353)}$$

$$= -0.21$$

As we could predicted, the result comes out to be the same as what we have computed with the previous example, which is a negative value -0.21. Hence, we do not likely to recommend video #3 to user D.

Until this point, we can conclude that the Item-Item Collaborative Filtering does not differ from what we have done with the User-User Collaborative Filtering from the two approaches to the calculations, we would still need to iterate through every single video that are available on the platform in order to compute the similarities among them so why do we even bother to learn about this? The only difference is that the calculation among these two methods tries to tackle different aspect of the problem, one tries to tackle on the user side and other tries to tackle on the video side which makes the Item-Item is more efficient and easier to implement. We can easily realize that people are more unpredictable and harder to categorize. For instance, one people can be interested and like videos that are about sports, science, technology, geography, politics, and there is no way we can predict what that person might be interested into. On the other hand, videos are much more easier to categorize. It is either about sports, technology, animal, funny, there might be some overlapping, but it is still a lot more easier to put them into categories. However, the Item-Item Collaborative Filtering is not perfect and still carrying some of its flaws. For instance, the algorithm still requires tons of space, and it is computationally consuming with the rate of new videos each second. Another problem that is related to what we have discussed is the matrix still has a ton of zeros which is because only a fraction of videos is watched by the users and the users only watch a fraction of a video.

5. RECOMMENDATION: CLASSIFICATION

In my introduction slide about the YouTube recommendation algorithm, I have developed a function for classification that reads in multiple variables then decide whether we should recommend the video to that user or not.

$$Recommend_{Yes/No} = f(a_{age}, b_{gender}, c_{country}, d_{subscription}, e_{search\ history}, f_{watched\ history})$$

This is only a first touch of the variables that I can think of and like I have answered to a question after the presentation, I think there are more variables that can be added to the equation such as popularity, watchtime, users' likes, dislikes, shares, and survey responses.

According to "Deep Neural Networks for YouTube Recommendations", the system overview can be illustrated by figure 1. We can see that the system has two major filters, a "candidate generation" and a "ranking" system.

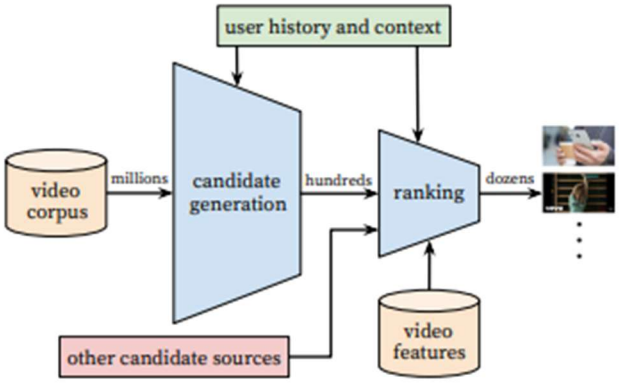

Figure 2

Since the video corpus is very large, the first filter is very crucial because it acts as the main factor to get the videos that is 80-90% suitable for the users already. This is the result after the algorithm is being fed with the users' inputs such as, personal information (obtain by signing in with google account), subscription, searched-watched history, likes, dislikes, shares.

After the first filter has been applied, only about hundreds of videos that match the users' preferences, the leftover work is to put those videos in a ranking system to see which video pop up first. This can be mostly determined by the watchtime of the user on a specific content and the algorithm should also take into consider the video length, content, the channel, etc. these are considered as video features. Furthermore, we can add a bit of spice to it by having one or two videos that we think that the user might interested in. This can be determined by some calculation to find the similarity between users with the Cosine Similarity formula.

❖ Cosine measure: If d_1 and d_2 are two vectors (e.g., term-frequency vectors), then

$$\cos(d_1, d_2) = (d_1 \bullet d_2) / ||d_1|| \, ||d_2|| ,$$

where • indicates vector dot product, $||d||$: the length of vector d

Figure 3

If the users' similarity score is pretty high (closer to 1) means that these two or this group of users share some similar interest or behavior. Hence, we can tell the algorithm to find and recommend videos that are in one person video corpus but not in the other person corpus. This would definitely add a more sophisticated, and another layer to what complicated algorithm has already been to make it more interesting for the users. Hence, keep the interest of the users to use the platform.

5.1. CANDIDATE GENERATION FILTER

After the pre-processing work with the enormous amount of data collected, the next crucial step is to come up with an efficient and effective candidate generation filter since it is the biggest filter in the model. Its purpose is to: prevent any inappropriate video that contains violent, sexuality, death, etc. to be recommended to the users; as well as narrow down the number of videos from millions to around a hundred. So, what data or features that YouTube choose in order to help them to feed this stage of the algorithm? The features are watch history (users' list of recently watched videos), search history (users' queries of what they type into the search box), location, device type, gender, age of users, and video freshness. For a shorter usage, let's call these User Context. The YouTube Recommendation algorithm will take advantage of the deep neural networks that basically is a matrix factorization. This neural network will be fed with one variable, in this case is the User Context, and then will output another variable which is the recommended videos to the users.

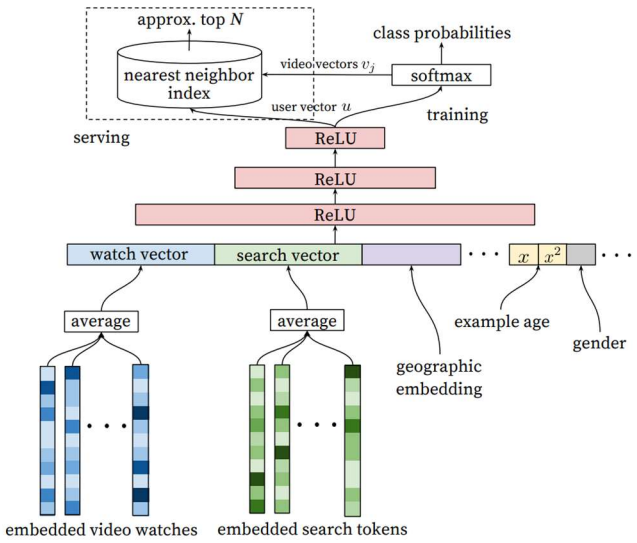

Figure 4

In order to turn these complicated features so that they can have the common unit and can be calculated later on, they will be turn into vectors and scalars. Searched history will be turned into tokens and then vectors, while watched videos is represented by a variable length sequence that is parsed with the video ID then get mapped into the watch vector (figure 3, from bottom up). Other features that are simpler such as location, age, gender is converted into scalars. Combine all of that, we will have a vector that contains all of the information needed which is the Rectified Linear Units (ReLU) that is ready to be fed to the neural network. After going through the calculations with the matrix factorization, the expected output would be the Full SoftMax, or the probability of the video being watched by the target user. However, this leads to an unwanted scenario.

Since we are considering the User Context as the input which means that every single video in the video corpus will need to go through this process and then we will choose the top 100 video that has the highest probability. This is not the ideal approach by any means. That directs us to the usage of a specific Training Data set that will be used for the Candidate Sampling.

5.1.a. TRAINING DATA

The training data is collected by monitoring every behavior of the users whenever they interact with the YouTube page. It could be a search, click to the new video, hit like, share, dislike button, etc. over a period of time. Each of the interaction will be considered as a User Context and the algorithm will input that User Context as a vector, let's call it x_i. Hence, x_i will be this part:

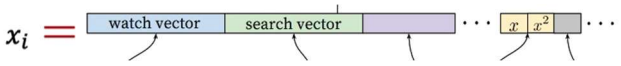

Figure 5

From that, we want to find a set of video S that this specific user might want to watch given these User Contexts: $S = P(S|x_i)$. (How does YouTube recommend videos? - AI EXPLAINED!)

5.1.b. CANDIDATE SAMPLING

Candidate Sampling, in short term, is a method that helps us to successfully keep the SoftMax properties by only sampling some number of classes from the original total number of classes. In order to calculate the $P(S|x_i)$, we want to use a method in Candidate Sampling which is called Sampled SoftMax. In this Sampled SoftMax, for each training sample x_i, we pick a small set $S_i \subset L$ of the sampled classes based on a chosen sampling function $Q(y|x)$. For each class $y \in L$ is included in S_i independent from the probability $Q(y|x_i)$:

$$P(S_i = S|x_i) = \prod_{y \in S} Q(y|x_i) \prod_{y \in (L-S)} (1 - Q(y|x_i))$$

Figure 6

In this equation, $Q(y|x_i)$ is the Bernoulli Distribution that calculates the probability of watching this set of videos y given the User Context x_i. By multiplying the set of videos that we should recommend $y \in S$ with the set of videos that we should not recommend $y \notin S$, we will get the final probability $P(S|x_i)$. We then will be able to collect the Train Time data set by summing up all of the probability. Let's call it $L(\theta)$. $L(\theta) = \prod_{i=1}^{N} P(S|xi; \theta)$. At this point, we will feed the Neural Network with the Train Time result to get approximate this optimization:

$$\theta^* = \arg . \max_{\theta} . \prod_{i=1}^{N} P(S|xi; \theta)$$

During the Test Time, we just have to get the sample from P to get a set of candidate videos given the User Context x_i.

However, there is a small drawback in this approach. The main objective of this method is to get the output of the sampled SoftMax to be as close as possible to the true SoftMax. Hence, the algorithm will minimize its scope and only focus on collecting the samples from the given distribution only which will left out the noise data. Another approach of Candidate Sampling is the Noise Contrastive Estimation or NCE loss. The difference between the two methods is that NCE loss will mainly focus on selecting the noise samples only and try to get as close as possible to the true SoftMax. This will only take one true class as input and a K noise classes. This approach becomes effective and useful when the target class approximately follow such a distribution. (Reducing the computational cost Mathematically- Candidate Sampling!)

5.2. RANKING

After narrowing down from millions of videos from the video corpus with multiple processes of calculation, we get ourselves a set of candidate videos that are around a hundred videos let's say. Our next step is to input that set of videos to the ranking algorithm which will output a relevant score accordingly with each video in that candidate set. The higher the score, the higher we place it on either the users' YouTube home page or the list of videos on the right side of the current playing video.

5.2.a. RELEVANCE SCORE

In order to set the inputs parameters for the ranking algorithm to work, YouTube used hundreds of features, some of them are already familiar with us by now as the User Context (search, watch history, like, share, dislike), some are new features that better describe the users' most recent behavior such as the number of time that the user watch a video from this channel, and when was the most recent video that the user watch on this topic, etc. In addition to that, the score of the video will change accordingly for every impression, in other words, whenever the user interacts with it. It also goes the other way around for recommended video but don't get interacted by the user, the score will decrease. With that much of information that we have known so far about the ranking algorithm, it is still a massive and computationally expensive calculations run in the background. Therefore, the method of applying Candidate Sampling can still come very handy so that instead of re-ranking thousands of videos after a click of a user, to only a hundred of videos.

Let's try our first step into computing the relevance score. YouTube first attempt is to have a direct relationship between the relevance score with expected watch time of the video. It is reasonable since it behaves heavily based on the direct interaction of the user on the video, longer watched time equal bigger interest toward the content of that video which was the right intention to start with. However, it immediately grew a conflict argument, since score = expected watch time, it straight up favors lengthy video over videos that have shorter length; hence, create incorrect output that we want in the long run. That is the reason why, YouTube has come up with a solution to that and call it Weighted Logistic Regression.

5.2.b. WEIGHTED LOGISTIC REGRESSSION

The Weighted Logistic Regression is a method which it reads in two main inputs, the User Context, and the features of video in general. The output will stay the same as the relevance score. We want the score to not only include the interactions between the video and the user (both watched and unwatched video) but to also take into account the time that the user spends on watching the video. Hence, we can come up with a simple calculation as:

$$\text{Score} = \text{Odds} = \frac{weighted\ sum\ of\ all\ videos\ watched}{weighted\ sum\ of\ all\ videos\ not\ wached}$$

The term "odds" is used here for later purpose which is to easily distinguish the serving and training result at the last step. Basically, the higher the odds is the higher the expected watchtime that the users spent on the video. So far, the feature of watched and unwatched video has been included, but we are going to need another variable that can distinguish the interest of the user on various video length. Since the interest of the user on a 5-minute video cannot be equal to a 1-hour long video so it is crucial to add a weighted variable to the formula above. Let's say each video v is assigned a weight based on its watched time T, N are the number of candidate videos, k are the videos that have been watched by the user. We can then come up with a more sophisticated formula:

$$\text{Odds} = \frac{\sum_{i=1}^{N} Ti}{N-k}$$

With this formula, we can finally have all of the features that can contribute a lot of features that are important in terms of helping us to decide the relevance score of a video. First, we want the video that is longer but still attracts the users to weight more than those that are shorter and then have a summation on the numerator. Second, on the numerator, we will divide it with the leftover videos by subtracting the number of candidate videos with the videos that have been watched by the user. That is for theoretically, but realistically, N is a lot smaller than k since out of hundreds of candidate video, the number of watched videos by the users are very small. Hence, it can just be Odds = $\frac{\sum_{i=1}^{N} Ti}{N}$ and so the odds is roughly equal to the denominator which is the expected watchtime.

That is all the calculations in the background of the math aspect. Now, in order to implement them on the neural network approach, YouTube has used a method that is similar to the Candidate Sampling in Figure 3.

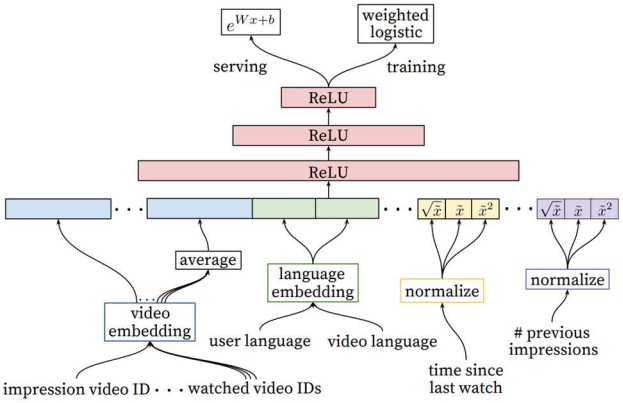

Figure 7

After computing multiple values of ReLU in training, after each layer of reducing the size, the output which is the weighted logistic will be able to learn the algorithm of the odds, or in other words, the log-odds. With that value of log-odds, while testing, we can directly exponent it to get the raw odds. That is the relevant score of the specific video to a specific user or what we have proven earlier, it is also roughly equal to the expected watch time of that video to the user.

6. SMALL SCALE RECOMMENDATION ALGORITHM

In order to implement the YouTube Recommendation Algorithm system, it is very crucial to have access to multiple types of data such as User Context data (watch history, search history, location, device type, gender, and age of users), Video Data (the whole corpus of videos, content, length, etc.), some of them will be unable to have access to such as the User Context because of privacy and some is to big of a data to be handled. Hence, I will try to analyze and learn from a small movie recommendation algorithm project to have some insight about the recommendation algorithm as a whole. I chose to look at the movie recommendation system because of two main reasons:

1. The data is accessible. I have chosen an online source that already has multiple features of movie data that I can use to serve for the algorithm
2. They share some similarities. The movie recommendation algorithm will use some overlapping features such as ratings (similar to like and dislikes), tags (similar to content).

The algorithm will work based on similarities with the User-User as well as Item-Item Candidate Filtering which the goal is to find the similarity between users simply by their ratings and movies by their tags. If they have some similar ratings for the same set of movies which show that they have some similarities in taste, then if one person have watched and rate a specific movie with high score then there is a high probability that the other person will share the same taste and like that movie. For instance, assuming that person A have watched movie M1, M2, M3 and rate them with a high score, person B also watched M1, M2, M3 and rate them with positive score; hence, their similarity might be high enough for the algorithm to pick out. Then, if person A has watched movie M4, rate it with a high score and person B has not watched that movie, based on the result after some calculations, we might want to recommend movie M4 to person B.

I also want to keep noticing a fact that this part of work will only be used as my other perspective on the User-User Candidate Filtering and Item-Item Candidate Filtering which is related to my main purpose that is to learn about the YouTube Recommendation Algorithm system. Therefore, I will try to learn, discover, and analyze the Movie Recommendation System with Pandas as a source for me to have an insight on what the recommendation algorithm might look like which can be very helpful for my study of the YouTube Recommendation System.

6.1. DATA

The data that we will use to have an insight in the Movie Recommendation Algorithm will be separated into four Comma-Separated Value (csv). These data (ratings, tags, movies) come from movielens.org – a movie recommendation service. It contains 100836 ratings and users' ratings that were selected for study purpose. The users are selected randomly regardless of location, age, demographic, etc. However, these users need to rate at least 20 movies in order to get picked and each user are represented by an id number. Below are the descriptions for each of the four files with four figures following to show the first four items in each file in Jupyter Notebook:

- Links (figure 8): stores the link between movie id with the imdbid (the Internet Movie Database) and the tmdbid (the Movie Database)

```
In [1]: import numpy as np
        import pandas as pd

In [2]: links_df = pd.read_csv('data/links.csv')
        links_df.head()
```

Out[2]:

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

Figure 8

- Movies (figure 9): stores the movie id, title (with year produced), and genre.

```
In [3]: movies_df = pd.read_csv('data/movies.csv')
        movies_df.head()
```

Out[3]:

	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

Figure 9

- Ratings (figure 10): stores the user id, movie id, rating (in scale of 1 being worst to 5 being best, increment by 0.5), and time stamp

```
In [4]: ratings_df = pd.read_csv('data/ratings.csv')
        ratings_df.head()
```

Out[4]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Figure 10

- Tags (figure 11): stores the connection between user id, movie id, tag (genre) and time stamp

```
In [5]: tags_df = pd.read_csv('data/tags.csv')
        tags_df.head()
```

Out[5]:

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200

Figure 11

6.2. GENERATE RECOMMENDED MOVIES LIST

In this setup, we will be able to get a list of recommended movies based on the similarity between users and somewhat items (movies). The algorithm is straightforward, we will try to calculate the similarity based on their ratings. If they share some similar ratings for a specific set of movies, we will consider them as similar and retrieve a set of movies that either one person has watched and the other has not or movies that share the same genre or tag.

First, we will do a merge between the ratings database with the movie database by the movie id in order to perform the calculations.

```
In [6]: df = movies_df.merge(ratings_df, on='movieId')
        df
```

Out[6]:

	movieId	title	genres	userId	rating	timestamp
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1	4.0	964982703
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0	847434962
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	7	4.5	1106635946
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	15	2.5	1510577970
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	17	4.5	1305696483
...	...	...	...	...	...	...
100831	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy	184	4.0	1537109082
100832	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy	184	3.5	1537109545
100833	193585	Flint (2017)	Drama	184	3.5	1537109805
100834	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation	184	3.5	1537110021
100835	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy	331	4.0	1537157606

100836 rows × 6 columns

Figure 12

In this joint list, let's pick The Fast and the Furious 4 (2009) as an example. We will first retrieve the movie and then sort all the ratings for this movie. Then, run a for loop to get the first five users that rates this movie highest. For each user, we want to retrieve all the ratings that this person has given and get the five highest rated movies from these users. We will add this list of movies we just got by using numpy unique to an array and print it out of the console (Figure 13).

```
In [8]: F_f = 'Fast & Furious (Fast and the Furious 4, The) (2009)' # Title as input, now it's just one movie
        recommended_movies = []

        # Retrieve the movie in the database, sort all the ratings
        movie_db = df[df['title'] == F_f]\
                    .sort_values(by='rating', ascending=False)

        # Get the first 5 users who rate this movie highest
        for user in movie_db.iloc[:5]['userId'].values:

            # Get the rated movies for this user
            rated_movies = df[df['userId'] == user]

            # Get the five biggest rated movie by this user
            rated_movies = rated_movies[rated_movies['title'] != F_f]\
                           .sort_values(by='rating', ascending=False)\
                           .iloc[:5]

            # Add these to the recommendations
            recommended_movies.extend(list(rated_movies['title'].values))

        recommended_movies = np.unique(recommended_movies)

        for movie in recommended_movies:
            print(movie)
```

```
Battle Royale (Batoru rowaiaru) (2000)
Cast Away (2000)
Catch Me If You Can (2002)
Deadpool 2 (2018)
Donnie Darko (2001)
Enemy at the Gates (2001)
Formula of Love (1984)
George Carlin: It's Bad for Ya! (2008)
George Carlin: Life Is Worth Losing (2005)
Good Will Hunting (1997)
Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Love and Pigeons (1985)
Loving Vincent (2017)
Meet the Parents (2000)
Pursuit of Happyness, The (2006)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
```

Figure 13

This first list of recommended movies was only based on the similarity between users that share some interests towards a specific set of movies. So, we will take another step by running Items-Items filtering among the list of movies that we had previously by the similarity between genres. To do that, we will create a recommended movie genre (rec_movie_genre) that splits different genres by "|" of The Fast and the Furious (2009) movie (which are action, crime, drama, thriller) as well as splits the genres for every movie in the list that we have generated above. We will also keep track of a score variables that will increment every time it matches the genre so the higher the score, the higher the matching probability (figure 14).

```
In [10]: #split each genre by |
rec_movie_genre = df[df['title'] == F_f].iloc[0]['genres'].split('|')
scores = {}  # {title: score ...}

#split each genre of each movie from the rec movie list above
for movie in recommended_movies:
    best_fit = df[df['title'] == movie].iloc[0]
    movie_genres = best_fit['genres'].split('|')
    score = 0

    #increment score by one every time a genre match
    for rec_movie_genre in rec_movie_genre:
        if rec_movie_genre in movie_genres:
            score += 1

    scores[movie] = score

#sorting the score in descending order so the biggest will be up top of the list
recommended_movies = sorted(scores, key=lambda x: scores[x])[::-1]

#print the list out console
for movie in recommended_movies:
    print(movie)
```

```
Battle Royale (Batoru rowaiaru) (2000)
Formula of Love (1984)
George Carlin: It's Bad for Ya! (2008)
George Carlin: Life Is Worth Losing (2005)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Love and Pigeons (1985)
Meet the Parents (2000)
Ted (2012)
What We Do in the Shadows (2014)
Cast Away (2000)
Deadpool 2 (2018)
Enemy at the Gates (2001)
Good Will Hunting (1997)
Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il) (1966)
Pursuit of Happyness, The (2006)
Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
Star Wars: Episode V - The Empire Strikes Back (1980)
The Interview (2014)
Catch Me If You Can (2002)
```

Figure 14

6.3. YOUTUBE VS MOVIE RECOMMENDATIONS ALGORITHM

Based on the Simple Movie Recommendations approach above, we can see that the set of movies resulted from this algorithm was heavily based just the highest rating (5.0) as well as the movies that have been rated highest from the first five users. Then re-order the list by the most match genre. This approach can easily be accepted by newbie in the Machine Learning and Deep Learning field who has just begun to learn about the concept like I do for a variety of reasons: it is easy to follow; the algorithm used was simple as it only gets the highest item in ratings; the accessibility of data; even though the data was pretty big (more than 100000 units) but it is reasonably easy to use; the data preprocessing was not necessary since there is only two features left to compare and generate from which are rating and genre. On the other hand, we can also point out a handful of limitations from this model: the data collected was too plain, only one to two dimensions to compare (rating and genre); the error would be high because we only take into consider the top rated (5.0) movie from the list and ignore the middle value (for example: 3.0-4.0).

With a slight of track from the YouTube Recommendation Algorithm to have a hands-on insight on the Movie Recommendation, I was able to have some conclusions on what can be further developed from the movie algorithm to YouTube algorithm. When YouTube was first introduced, we all know that its recommendation algorithm was very simple and plain, it only took in the most popular video as a feature and then recommend those popular videos to every user on the platform at that time which is very similar to this movie recommendation algorithm. Hence, the movie recommendations system has given not only a generic but also very valuable perspective of how YouTube recommendation algorithm in its first steps looked like. This has showed that this study is right on track. If I were able to build more from the foundation of the simple Movie Recommendation model by adding more User Context, features, applying the Collaborative Filtering as well as the Classification algorithm, I would then enter the realm of recommendation algorithms. There would still be obstacles. For instance, the data obtained would be too large that my personal computer will not be able to build and run; inaccessible Users data; time limitation to a quarter long.

7. ACKNOWLEDGMENTS

Since I consider this as an enormous project to learn about in the length of a quarter because of variety of obstacles that I have identified, I will try to take one step at a time by doing researching, reading, and receiving lots of information on how people would approach this problem in the first place. The YouTube official Blog has helped me a lot by giving a generic perspective back in day when the algorithm was first introduced. The "Deep Neural Networks for YouTube Recommendations" paper, on the other hand, has provided a lot of insights as well as reinforce my own formula that I have developed (they used a similar algorithm but with more complexity to it). I also took an additional step which was slightly off-track to have a hands-on experience on a small-scale movie recommendation algorithm that use real data thanks to the video "How to build a Movie Recommendation System" made by Open Weaver. Although this algorithm was working on a movie database, it acted very similar to the video recommendations algorithm by YouTube back in the days; hence, it has given a lot of not only reassurance but also knowledge to back-up my statement as well as my study about various approaches, algorithm that run in the back scene.

REFERENCES

[1] Covington, P., & Adams, J. (n.d.). Deep Neural Networks for YouTube Recommendations. Retrieved November 5, 2022, from https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45530.pdf.

[2] Goodrow, C. (2021, September 15). On YouTube's recommendation system. blog.youtube. Retrieved November 5, 2022, from https://blog.youtube/inside-youtube/on-youtubes-recommendation-system/

[3] YouTube. (2019). How does YouTube recommend videos? - AI EXPLAINED! Retrieved November 7, 2022, from https://www.youtube.com/watch?v=wDxTWp3KMMs.

[4] YouTube. (2021). Recommendation systems overview (Building recommendation systems with TensorFlow). Retrieved November 7, 2022, from https://www.youtube.com/watch?v=BthUPVwA59s.

[5] Mullery, S. (2021, July 1). How the YouTube algorithm works in 2021. Tinuiti. Retrieved December 6, 2022, from https://tinuiti.com/blog/paid-social/youtube-algorithm

[6] kandikits. (2022, October 20). Movie recommendation system with Pandas. kandi. Retrieved December 6, 2022, from https://kandi.openweaver.com/collections/starterkits/simple-data-analysis?utm_source=youtube&utm_medium=social&utm_campaign=organic_kandi_ie&utm_content=kandi_ie_kits&utm_term=all_devs

[7] Movielens. GroupLens. (2021, December 8). Retrieved December 6, 2022, from https://grouplens.org/datasets/movielens/

[8] YouTube. (2022). How to build a Movie Recommendation System | Source Code included | kandi tutorial. Retrieved December 11, 2022, from https://www.youtube.com/watch?v=Qkr2cAExsRI.

[9] What is candidate sampling. tensorflow. (n.d.). Retrieved December 11, 2022, from https://www.tensorflow.org/extras/candidate_sampling.pdf

[10] Siri, S. (2018, March 9). Reducing the computational cost mathematically-candidate sampling! LinkedIn. Retrieved December 11, 2022, from https://www.linkedin.com/pulse/reducing-computational-cost-mathematically-shamane-siriwardhana/