**Recommendation System for YouTube**

Submitted by: KATHIRAVAN NATARAJAN

Email ID: [sunkathirav@gmail.com](mailto:sunkathirav@gmail.com)

Contact number: +13137822951

**Insight about understanding videos**

Sense of humor changes from one person to another and from one nationality to another and perhaps from one age group to other. It is not an easy task to make a machine to understand the video even to the top most Machine Learning scientists in the world. If a machine can understand the nature of the video, then that is a huge development in Artificial Intelligence and Machine Learning.

Currently, Google can recognize objects in the video using Machine Learning. In the Keynote 2017, Google has introduced Cloud Video Intelligence API [[1]](#footnote-1) which uses the powerful deep learning models built using Tensorflow and other frameworks to understand the videos. The API is the first of its kind, enabling developers to easily search and discover video content by providing information about entities (nouns such as “dog,” “flower” or “human” or verbs such as “run,” “swim" or “fly”) inside video content. It can even provide contextual understanding of when those entities appear; for example, searching for “Tiger” would find all precise shots containing tigers across a video collection in Google Cloud Storage. So, it is possible for Google that if you give verb as "laugh" or "funny". This laughing part of the video does not necessarily make the people who are watching to laugh. There can be videos which may not have any language influence, yet it creates humor just with the bodily actions.

**Problem statement**

To generate a list of top10 songs by accessing YouTube programmatically that can make a user laugh.

**Approach to this problem**

A way to approach the problem of finding 10 videos to make people laugh based on the title of the video. To implement this, I need to use YouTube API to fetch the YouTube video information based on the keyword search.

Some prerequisites to this YouTube mining is to get the developer API key. Only with a valid API key, developers can mine the YouTube information. Created a API key in the developer console[[2]](#footnote-2). In this project, I have implemented the YouTube mining code (File name: youtubeMiner.py) which works to search the video information based on the keyword. Also, if I start to search the videos only with the keyword like "funny" or "laugh", I will miss many "prank videos" which makes people laugh in the recent times. Hence this search must contain at least few keywords related to fun or laugh.

One approach to find the synonyms of the word “funny” is by using NLTK’s WORDNET to obtain synonyms from lemma\_names() function. This approach will avoid the manual time taken to search in the dictionary which can give the synonyms of the keywords like “funny, laugh”. When I implemented this, some terms from the result of lemma\_names() like comic, laughable are really relevant. But some of the results are irrelevant the current scenario. The irrelevant terms like rummy, suspicious and singular are totally inappropriate to the keyword based search.

Unfortunately, YouTube allows developers to fetch the information of only 50 videos in single execution of the script. So, it is fine to get some 50 synonyms of the word funny and insert into a list which is in turn used to mine the YouTube API.

Keywords list includes "laugh", "prank", "funny", "humorous", "ludicrous", "ridiculous", "joking", "amusing", "fun", "for grins", "humor", "comical", "jolly", "hilarious", "witty", "comic", "droll", "facetious", "jocular", "jokey", "chuckle", "goofy", "chortle", "wacky".

If you are running the YouTube miner code in ipython/ jupyter notebook, then you will definitely come across the following issue which is not fixed by Google's team yet.

ArgumentError: argument --q: conflicting option string(s): --q

To avoid that, it is better to run the YouTube mining code in the console or any IDE’s like spyder, pycharm etc. There will be also errors like encoding, decoding issues since YouTube video titles are not always just strings. Titles do have emojis and other language

**Recommendation system design**

Various approaches for building recommendation systems have been developed by many researchers. The approached include collaborative filtering, content-based filtering (if you like an item, you will also like a similar item) or hybrid filtering. Collaborative filtering technique is the most mature and the most commonly implemented.

Collaborative filtering recommends items by identifying other users with similar taste; it uses their opinion to recommend items to the active user.

**Classic examples for collaborative filtering incorporated companies**

GroupLens is a news-based architecture which employed collaborative methods in assisting users to locate articles from massive news database. Ringo is an online social information filtering system that uses collaborative filtering to build users profile based on their ratings on music albums. Amazon uses topic diversification algorithms to improve its recommendation and Netflix uses collaborative filtering on their movie recommendations too. Even YouTube considers collaborative filtering on their video recommendation

**Recommendation system implementation**

In YouTube search, when I searched passionate, I got the results which were posted 2 years ago, 5 years ago and 2 years ago with different number of views including 824888, 195494 and 871135. So, the recommendation system cannot just depend on the number of likes, views, comments.

In our case of recommending funny videos to make people laugh, videos made in USA with enormous number of views, likes and comments which may seem nothing humorous to people in India and vice - versa. There also can be some funny videos which is made of different natural languages. At this moment, we can omit this scenario of location-based recommendations since I don't have enough information like languages on which the videos are made to recommend

Recommendation system can also be implemented based on the popularity. But all users will have the same recommendations of videos. There comes an issue of absence of personalization and only the same set of videos will be recommended to all the users.

Types of collaborative filtering are,

1. Memory based collaborative filtering

1. User - User collaborative filtering

2. Item - Item collaborative filtering

2. Model based collaborative filtering

I am planning to use collaborative filtering here in order to make best recommendations by considering the discussed scenarios since it personalize the video recommendations very well when compared to other approaches.

**Assumptions**

Introducing 6 users in the dataset and whose names are Kathir, Sundhar, Chris, Patrick, MSD and MarkZ in "Users". Along with those users, I have also introduced the assumed like count of each video to that paticular user. This may seem to be little fuzzy. But it is like normal mapping of one user to the multiple videos in the dataset and each video will have the like or dislike option in the "Liked" column. If the value is 1, then that user liked that video. If the value is 0, then it can be considered as dislike or not watched that video.

**Collaborative Filtering implemented using graphlab**

Reminder: Need to register graphlab package inorder to use it but it comes for free

I am implementing this recommendation system by making an item-item matrix in which we keep a record of the pair of items which were liked. In this case, an item is a YouTube video. Once I have the matrix, I use it to determine the best recommendations for a user based on the videos he has already liked. Note that there a few more things to take care in actual implementation which would require deeper mathematical introspection, which I’ll skip for now.

Three types of item similarity metrics supported by graphlab are

**1. Jaccard Similarity:**

Similarity is based on the number of users which have rated item A and B divided by the number of users who have rated either A or B. It is typically used where we don’t have a numeric rating but just a boolean value like a product being bought or an add being clicked

**2. Pearson Similarity**

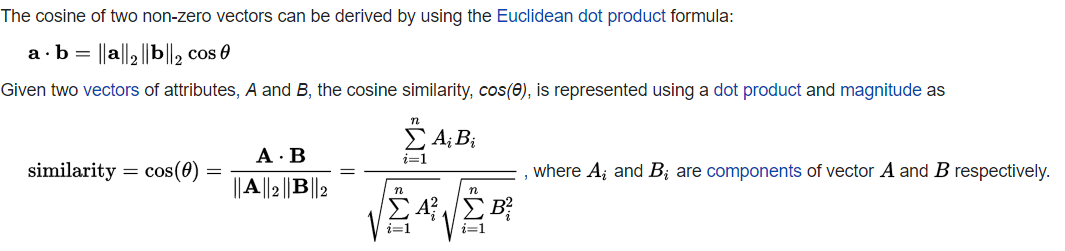
Similarity is the pearson coefficient between the two vectors

**3. Cosine Similarity:**

Similarity is the cosine of the angle between the 2 vectors of the item vectors of A and B. Closer the vectors, smaller will be the angle and larger the cosine

Though this **function comes as a black-box with different similarities**, it is always advised to **understand the algorithm better** and later they can be just imported from the libraries too.

Without using graphlab, this cosine similarity can be implemented just using numpy and pandas. But this graphlab library scales very well to the dataset that I have and another reason is that I have never tried this library so far.



#Train Model

item\_sim\_model = graphlab.item\_similarity\_recommender.create(train\_data, user\_id='users', item\_id='v\_title', target='Liked', similarity\_type='cosine')

#Make Recommendations:

item\_sim\_recomm = item\_sim\_model.recommend(users=user\_names,k=10)

item\_sim\_recomm.print\_rows(num\_rows=25

In the above code, item\_similarity\_recommender.create() function performs the cosine similarity between users column and v\_title column. After creating the similarity matrix, it is easy to recommend any number of videos. Since it is mentioned in the problem statement to recommend 10 videos, I have assigned the value of K to 10.

Now recommendations are given in the order of relevance. There should be some evaluation metric to validate these recommendations.

**Evaluation metrics for this recommendation System**

**1. Recall**

What ratio of videos that a user liked were actually recommended. If a user liked, say 5 items and the recommendation decided to show 3 of them, then the recall is 0.6

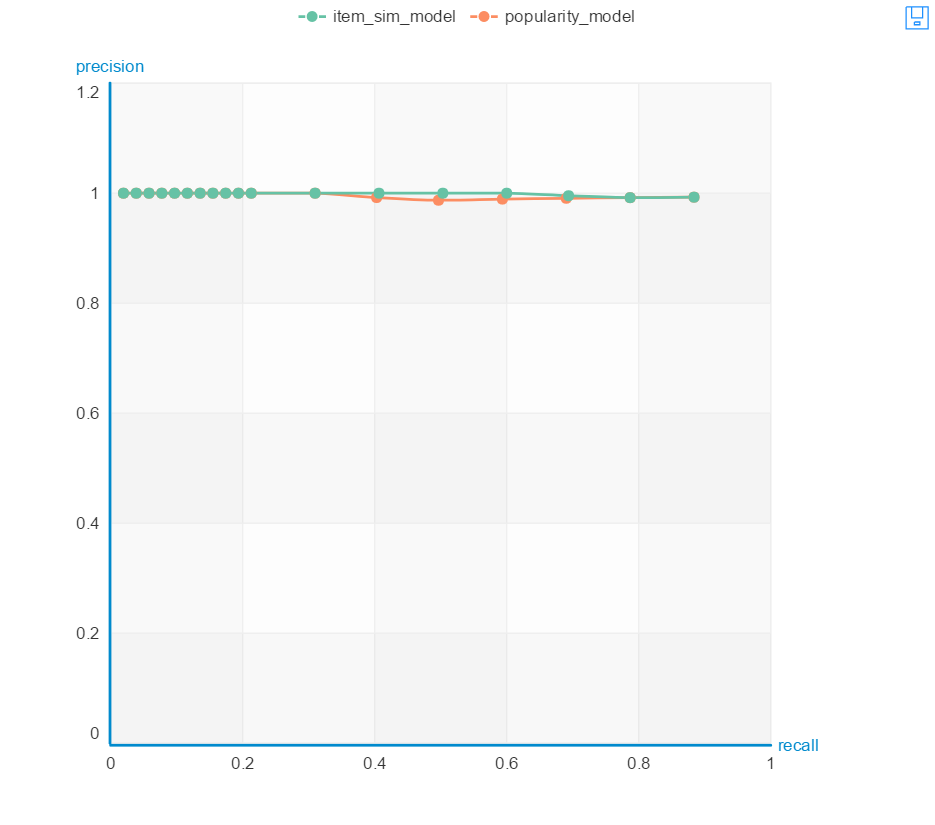
**2. Precision**

Out of all the recommended items, how many the user actually liked? If 5 items were recommended to the user out of which he liked, say 4 of them, then precision is 0.8

For further information for Recall and Precision, please refer the following link:

https://en.wikipedia.org/wiki/Precision\_and\_recall

The following figure compares the performance of item-similarity model and popularity. In this, both model appear to be performing similar. But before coming to conclusion, we must think about the assumptions that I made earlier. I sampled the user’s data and their choices(Liked) in a stochastic manner. And obviously they are not the real-world data. So, these exceptions will not be existing in the real-world YouTube recommendation system.



1. https://cloud.google.com/blog/big-data/2017/03/announcing-google-cloud-video-intelligence-api-and-more-cloud-machine-learning-updates [↑](#footnote-ref-1)
2. http://help.dimsemenov.com/kb/wordpress-royalslider-tutorials/wp-how-to-get-youtube-api-key [↑](#footnote-ref-2)