Youtube Video Recommendation System

Abstract

We discuss the video recommendation system in use at YouTube, the world's most popular online video community. The system recommends personalized sets of videos to users based on their activity on the site. We discuss some of the unique challenges that the system faces and how we address them.

Introduction

Personalized recommendations are a key method for information retrieval and content discovery in today's informationrich environment. Combined with pure search (querying) and browsing (directed or non-directed), they allow users facing a huge amount of information to navigate that information in an efficient and satisfying way. As the largest and most-popular online video community with vast amounts of user-generated content, YouTube presents some unique opportunities and challenges for content discovery and recommendations.

The goal of the system is to provide personalized recommendations that help users find high quality videos relevant to their interests. In order to keep users entertained and engaged, it is imperative that these recommendations are updated regularly and reflect a user's recent activity on the site.

System Design

The set of recommended videos videos is generated by using a user's personal activity (watched, favorited, liked videos) as seeds and expanding the set of videos by traversing a co-visitation based graph of videos. The set of videos is then ranked using a variety of signals for relevance

Input

During the generation of personalized video recommendations we consider a number of data sources. In general, there are two broad classes of data to consider: 1) content data, such as the raw video streams and video metadata such as title, description, etc, and 2) user activity data, which can further be divided into explicit and implicit categories. Explicit activities include rating a video, favoriting/liking a video, or subscribing to an uploader. Implicit activities are datum generated as a result of users watching and interacting with videos, e.g., user started to watch a video and user watched a large portion of the video (long watch).

Related videos

The creation of a mapping from a video vi to a group of comparable or related videos Ri is one of the foundational elements of the recommendation system. We define similar movies in this context as ones that a user is likely to watch after seeing the provided seed video v. We utilize a well-known method called association rule mining or co-visitation counts to calculate the mapping.

Consider sessions of user watch activities on the site. For a given time period (usually 24 hours), we count for each pair of videos (v_i, v_j) how often they were co-watched within sessions. Denoting this co-visitation count by c_{ij}, we define the relatedness score of video v_j to base video v_i as: $r(v_i, v_j) =$ cij

One of the simplest normalization functions is to simply divide by the product of the videos' global popularity: $f(v_i, v_j) = c_i \cdot c_j$.

We then pick the set of related videos R_i for a given seed video v_i as the top N candidate videos ranked by their scores $r(v_i, v_j)$. Note that in addition to only picking the top N videos, we also impose a minimum score threshold.

The related videos can be seen as inducing a directed graph over the set of videos: For each pair of videos (v_i, v_j), there is an edge e_{ij} from v_i to v_j iff $v_j \in R_i$, with the weight of this edge given by associaction rule mining.

Generating Recommendation Candidates

To compute personalized recommendations we combine the related videos association rules with a user's personal activity on the site: This can include both videos that were watched (potentially beyond a certain threshold), as well as videos that were explicitly favorited, "liked", rated, or added to playlists. We call the union of these videos the seed set.
In order to obtain candidate recommendations for a given seed set S, we expand it along the edges of the related videos graph: For each video vi in the seed set consider its related videos Ri. We denote the union of these related video sets as C1

$$C_1(S) = \bigcup_{v_i \in S} R_i$$

In many cases, computing C1 is sufficient for generating a set of candidate recommendations that is large and diverse enough to yield interesting recommendations. However, in practice the related videos for any videos tend to be quite narrow, often highlighting other videos that are very similar to the seed video. This can lead to equally narrow recommendations, which do achieve the goal of recommending content close to the user's interest, but fail to recommend videos which are truly new to the user. In order to broaden the span of recommendations, we expand the candidate set by taking a limited transitive closure over the related videos graph. Let Cn be defined as the set of videos reachable within a distance of n from any video in the seed set:

$$C_n(S) = \bigcup_{v_i \in C_{n-1}} R_i$$

Due to the high branching factor of the related videos graph we found that expanding over a small distance yielded a broad and diverse set of recommendations even for users with a small seed set. Note that each video in the candidate set is associated with one or more videos in the seed set. We keep track of these seed to candidate associations for ranking purposes and to provide explanations of the recommendations to the user.

$$C_{final} = (\bigcup_{i=0}^{N} C_i) \setminus S$$

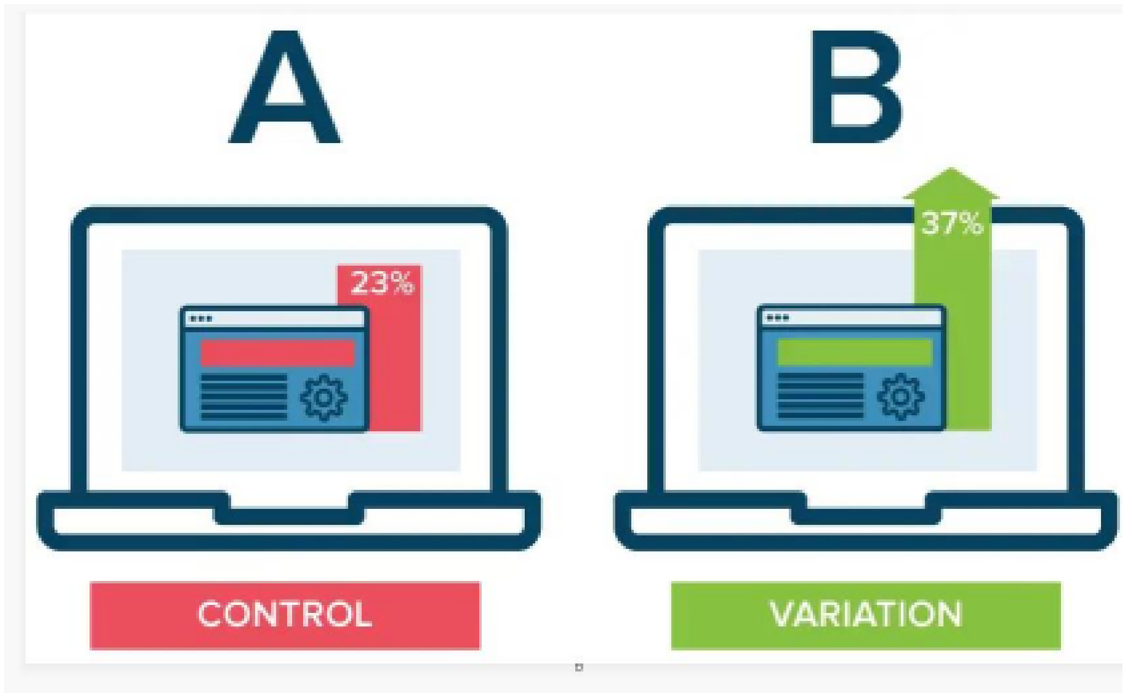
Ranking

After the generation step has produced a set of candidate videos they are scored and ranked using a variety of signals. The signals can be broadly categorized into three groups corresponding to three different stages of ranking:
1) video quality - Video quality signals are those signals that we use to judge the likelihood that the video will be appreciated irrespective s include view count (the total number of times a video has been watched), the ratings of the video, commenting, favoriting and sharing activity around the video, and upload time
2) user specificity - User specificity signals are used to boost videos that are closely matched with a user's unique taste and preferences. To this end, we consider properties of the seed video in the user's watch history, such as view count and time of watch.
3) diversification - Using a linear combination of these signals we generate a ranked list of the candidate videos. Because we display only a small number of recommendations (between 4 and 60), we have to choose a subset of the list. Instead of choosing just the most relevant videos we optimize for a balance between relevancy and diversity across categories. Since a user generally has interest in multiple different topics at differing times, videos that are too similar to each other are removed at this stage to further increase diversity.

Evaluation

In our production system we use live evaluation via A/B testing as the main method for evaluating the performance of the recommendation system. In this method, live traffic is diverted into distinct groups where one group acts as the control or baseline and the other group is exposed to a new feature, data, or UI. The two groups are then compared against one another over a set of predefined metrics and possibly swapped for another period of time to eliminate other factors. The advantage of this approach is that evaluation takes place in the context of the actual website UI. It's also possible to run multiple experiments in parallel and get quick feedback on all of them. The downsides are that not all experiments have reasonable controls that can be used for comparison, the groups of users must have sufficient traffic to achieve statistically significant results in a timely manner and evaluation of subjective goals is limited to the interpretation of a relatively small set of pre-defined metrics.

To evaluate recommendation quality we use a combination of different metrics. The primary metrics we consider include click through rate (CTR), long CTR (only counting clicks that led to watches of a substantial fraction of the video), session length, time until first long watch, and recommendation coverage (the fraction of logged in users with recommendations). We use these metrics to both track performance of the system at an ongoing basis as well as for evaluating system changes on live traffic.



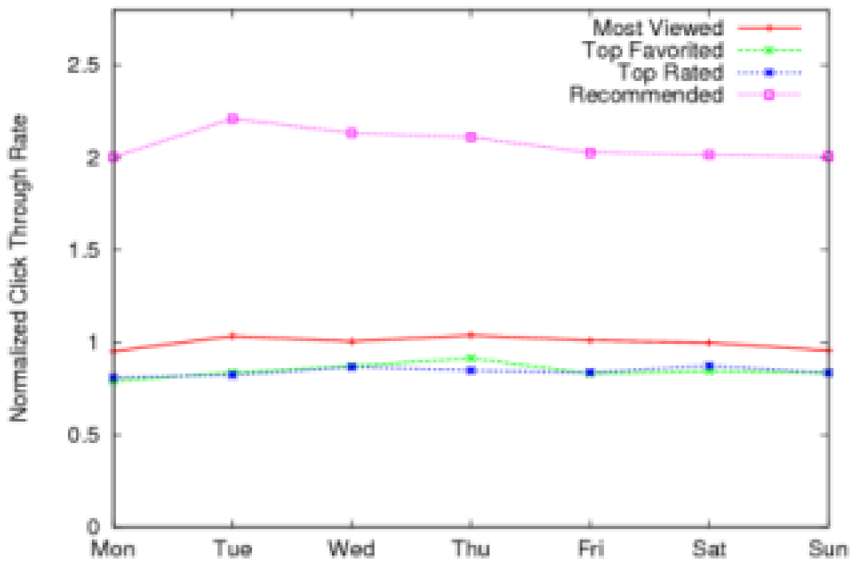
Result

Click rate is the percentage of people who were delivered your email that clicked through. Click through rate is the percentage of people who opened your email that clicked through.

The recommendations feature has been part of the YouTube homepage for more than a year and has been very successful in context of our stated goals. For example, recommendations account for about 60% of all video clicks from the home page. Having a better recommendation system than your competitor can be like having a leverage over everything. Big tech companies loose billions of dollars because of this.

Comparing the performance of recommendations with other modules on the homepage suffers from presentation bias (recommendations are placed at

the top by default). To adjust for this, we look at CTR metrics from the "browse" pages and compare recommendations to other algorithmically generated video sets:
a) Most Viewed - Videos that have received most number of views in a day.
b) Top Favourite - Videos that the viewers have added to their collection of favourites.
c) Top Rated - Videos receiving most like ratings in a day.



We measured CTR for these sections over a period of 21 days. Overall we find that co-visitation based recommendation performs at 207% of the baseline Most Viewed page when averaged over the entire period, while Top Favourite and Top Rated perform at similar levels or below the Most Viewed baseline.

This is based on the time frame of 3 weeks

Implementation

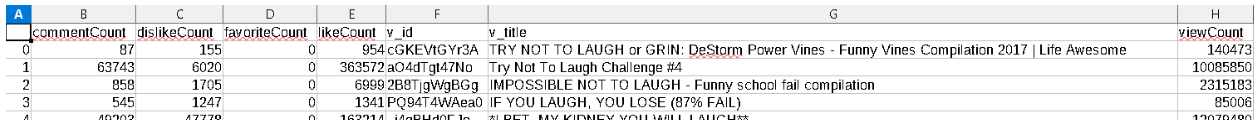
YoutubeRecommender_Group8.ipynb
https://colab.research.google.com/drive/1Ycn4oBvwdorkAKLge1mRLFkc4jHPFo1L?usp=sharing

DataScrapper.py

```
from apiclient.discovery import build #pip install google-api-python-client
from apiclient.errors import HttpError #pip install google-api-python-client
import pandas as pd #pip install pandas
import oauth2client.tools as oauthtools #pip install oauth2client
import importlib

DEVELOPER_KEY = "AIzaSyBsVyzUuSlxNxhTgVN6KR7R294-IG-5LuY"
YOUTUBE_API_SERVICE_NAME = "youtube"
YOUTUBE_API_VERSION = "v3"

def youtube_search(words):
        importlib.reload(oauthtools)
        oauthtools.argparser.add_argument("--q", help="Search term", default=words)
        oauthtools.argparser.add_argument("--max-results", help="Max results", default=50)
        args = oauthtools.argparser.parse_args()
        options = args
        youtube = build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION, developerKey=DEVELOPER_KEY)
        search_response = youtube.search().list(
         q=options.q,
         type="video",
         part="id,snippet",
         maxResults=options.max_results
        ).execute()
        videos = {}
        for search_result in search_response.get("items", []):
         if search_result["id"]["kind"] == "youtube#video":
            videos[search_result["id"]["videoId"]] = search_result["snippet"]["title"]
        s = ','.join(videos.keys())
        videos_list_response = youtube.videos().list(
         id=s,
         part='id,statistics'
        ).execute()
        res = []
        for i in videos_list_response['items']:
         temp_res = dict(v_id = i['id'], v_title = videos[i['id']])
         temp_res.update(i['statistics'])
         res.append(temp_res)
        df = pd.DataFrame.from_dict(res)
        df.to_csv('YTScrapedVideos.csv', mode='a',encoding='utf-8')


terms = ["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", "ligma"]
for i in terms:
        youtube_search(i)
```

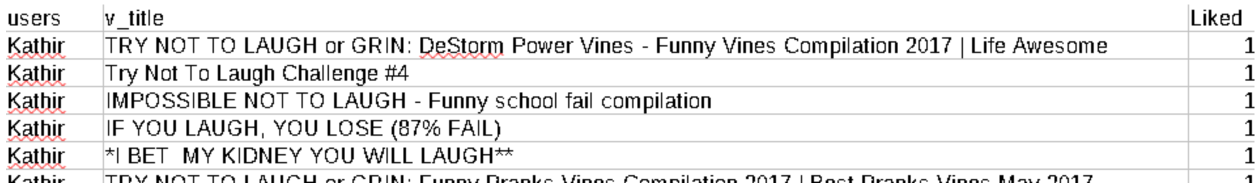
Datasets:

YTScraped.csv



A	B	C	D	E	F	G	H
	commentCount	dislikeCount	favoriteCount	likeCount	v_id	v_title	viewCount
0	87	155	0	954	cGKEVtGYr3A	TRY NOT TO LAUGH or GRIN: DeStorm Power Vines - Funny Vines Compilation 2017 | Life Awesome	140473
1	63743	6020	0	363572	aO4dTgt47No	Try Not To Laugh Challenge #4	10085850
2	858	1705	0	6999	2B8TjgWgBGg	IMPOSSIBLE NOT TO LAUGH - Funny school fail compilation	2315183
3	545	1247	0	1341	PQ94T4WAea0	IF YOU LAUGH, YOU LOSE (87% FAIL)	85006
4	49203	47778	0	163214	i4gBHd0FJo	*I BET MY KIDNEY YOU WILL LAUGH**	12079480

UsersHistory.csv



users	v_title	Liked
Kathir	TRY NOT TO LAUGH or GRIN: DeStorm Power Vines - Funny Vines Compilation 2017 | Life Awesome	1
Kathir	Try Not To Laugh Challenge #4	1
Kathir	IMPOSSIBLE NOT TO LAUGH - Funny school fail compilation	1
Kathir	IF YOU LAUGH, YOU LOSE (87% FAIL)	1
Kathir	*I BET MY KIDNEY YOU WILL LAUGH**	1
Kathir	TRY NOT TO LAUGH or GRIN: Funny Pranks Vines Compilation 2017 | Best Pranks Vines May 2017	1

References

1. The YouTube video recommendation system - James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet. [https://www.researchgate.net/publication/221140967_The_YouTube_video_recommendation_system]
2. YouTube Data API v3 [https://developers.google.com/youtube/?hl=en_US]
3. Turi Create API [https://apple.github.io/turicreate/docs/api/index.html]