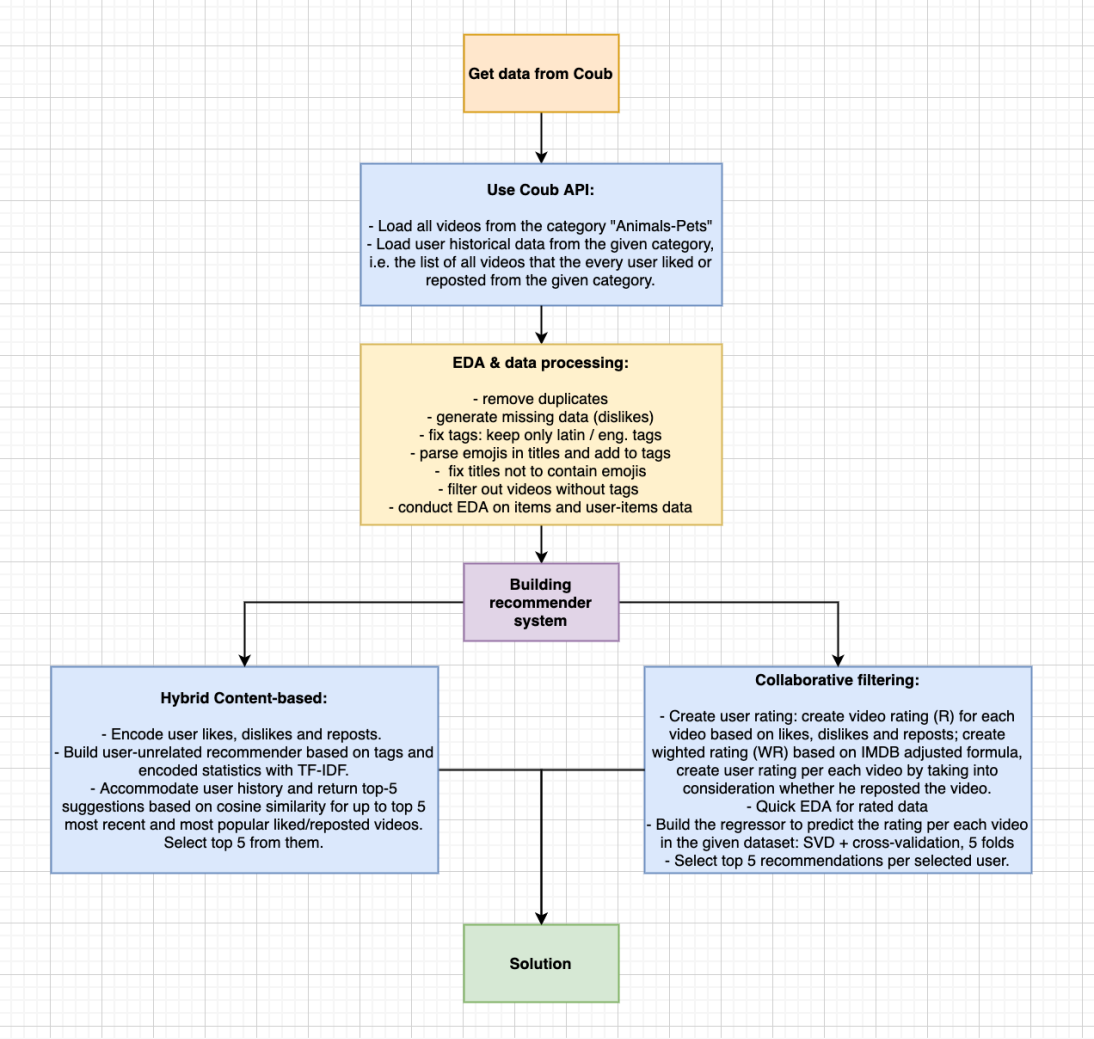
Coub recommender system demo.
Final report.

The purpose of this project has been to conduct research and build the prototype recommender system to suggest new coub videos, which are likely to fit their interests and previous history. The category chosen for the current demo is **Animals-Pets.** The flow of the project is presented below:

Figure 1 – Project flow



Data Load

In the first place we need to download the input data from Coub. This data includes two data sources:

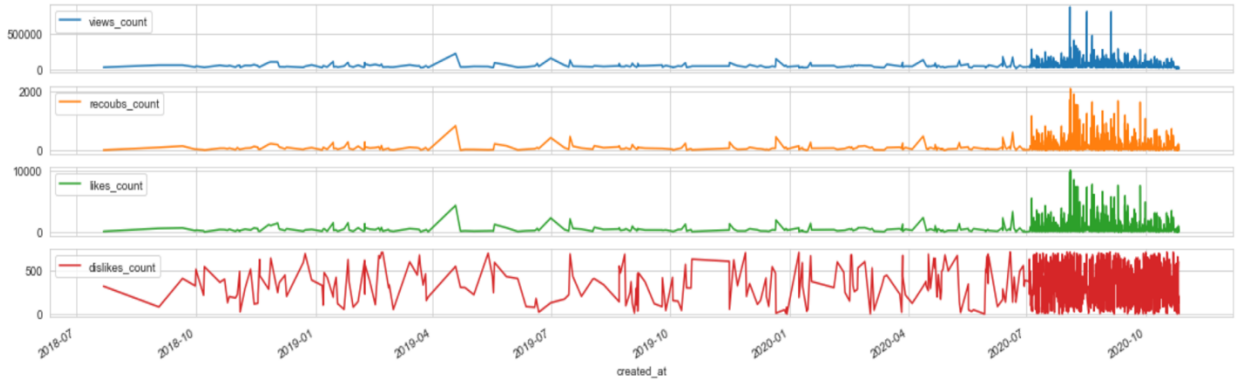
- The list of all videos from the selected category Animals-Pets along with relevant metadata, like title, date of creation, likes and dislikes, reposts, tags, URL, etc.
- The list of all user history data with respect to the given videos.

Once data is loaded, preprocessing starts. Filter out all non-Latin tags from the tag-list, parse the emojis that might be present in the title, remove them from the title along with the all non-alpha-numeric and non-Latin characters. Remove duplicates. It appeared that for the same videos we had multiple entries, which differed only by number of views. Thus, I kept only the most recent – the bigger number. The result has been that data has shrunk by 2.5 times.

Example of processing the emojis string:
'Little #kitty cat is so funny 🐱😂😂' → ['little', 'cat', 'kitty', 'funny', 'grinning', 'face', 'joy', 'tears']

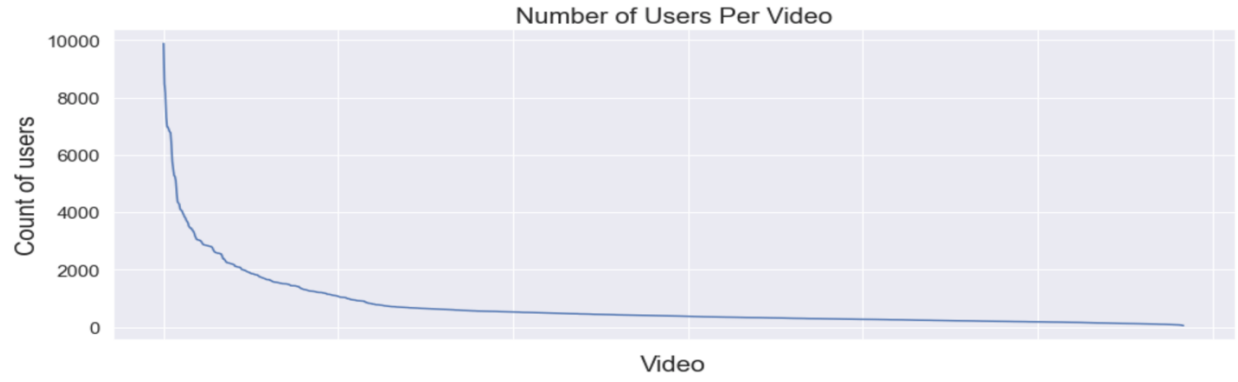
Let's see what the EDA has to say us about our data.

Figure 2 – Temporal distribution of likes, reposts and dislikes



It seems that most of the activity in the current category is quite recent and gained momentum in July 2020. Since we didn't have the real dislikes data, I randomly generated them.

Figure 3 – Number of users per video



The figure shows most of our users didn't watch too many movies, up to 500 maybe, but some videos are just top tier performers with more than 10000 unique users.

More details on the EDA results are in the research notebook.

Building Recommender for our videos

Content-Based System

At first, I went with the item content-based approach. We have abundant metadata available, so I used it to build impersonalized recommendations based on the item similarity only. I encoded the likes, dislikes and reposts columns by dividing them into low, middle and high ranges. This text representation, along with the title and tags, has been concatenated into a string to be analyzed through the TF-IDF vectorizer and comparing the cosine similarity of the given text string. But before using TF-IDF, the lemmatization of words has been performed to keep only the root of the word.

The **TD-IDF** algorithm is used to weigh a keyword in any document and assign the importance to that keyword based on the number of times it appears in the document. Put simply, the higher the TF-IDF score (weight), the rarer and more important the term, and vice versa.

Each word or term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TF-IDF weight of that term.

The **TF (term frequency)** of a word is the number of times it appears in a document. When you know it, you're able to see if you're using a term too often or too infrequently.

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

The **IDF (inverse document frequency)** of a word is the measure of how significant that term is in the whole corpus.

IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

$$w_{x,y} = tf_{x,y} * log\left(\frac{N}{df_x}\right),$$

where:

- tf_{x,y} - frequency of x in y
- df_x - number of documents containing x
- N - total number of documents

This is a good way to represent text strings and based on the cosine similarity – define most similar items.

However, at present no previous user history is taken into consideration. Thus, I built for every user I sorted his up to top 5 most recent and most popular items, and against every one of them ran the content-based recommender defined above. As a result, I ended up with up to 25 movies, of which based on the cosine similarity the algorithm returns only top-5. But if we want to specify the user and the exact video to perform recommendation for and return top-5 similar items, this option is available too. Finally if you want to visually inspect the result, open up the link specified in the url_big field.

Figure 4 – Sample output from the recommender engine

	id	title	sim_score	url_big
699	152962743	Мы пойдем с конём	0.466989	https://coubsecure-s.akamaihd.net/get/b59/p/co...
968	155151492	Это маи все все маи	0.423465	https://coubsecure-s.akamaihd.net/get/b119/p/c...
619	152458696	Big Horse	0.275839	https://coubsecure-s.akamaihd.net/get/b51/p/co...
361	150123750	birds	0.270196	https://coubsecure-s.akamaihd.net/get/b41/p/co...
11	89848750	Bird	0.209296	https://coubsecure-s.akamaihd.net/get/b201/p/c...

Collaborative filtering

Next, I decided to check out how the collaborative filtering would work on our data. But in order to proceed I needed to create the rating for each video, since it wasn't initially available to us. I had two options – randomly create it or try to integrate some logic while using the existing statistics on the very same likes, dislikes and reposts information which we already have. So, I went with the second option. Firstly, I introduced the rating per movie following the next logic: I binarize the mentioned above columns into three bins each and assign the value from 2 to 4 with respect to where specific value falls into except for dislikes, which amount to 1 to 3 - we don't want to penalize that much.

R = (views_count(score) + recoubs_count(score) + likes_count(score) - dislikes_count(score))

In this way we incorporated the statistics of interest. But it's not quite weighted yet, since different users differently assess the video, and number of assessments also varies. Therefore, I adjusted the following IMDB formula to account for this:

$$WR = \left(\frac{v}{(v+m)}\right) * R + \left(\frac{m}{(v+m)}\right) * C,$$

where:

- v - the number of votes for the movie;
- m - the minimum votes required to be listed in the chart;
- R - the average rating of the movie;
- C - the mean vote across the whole report.

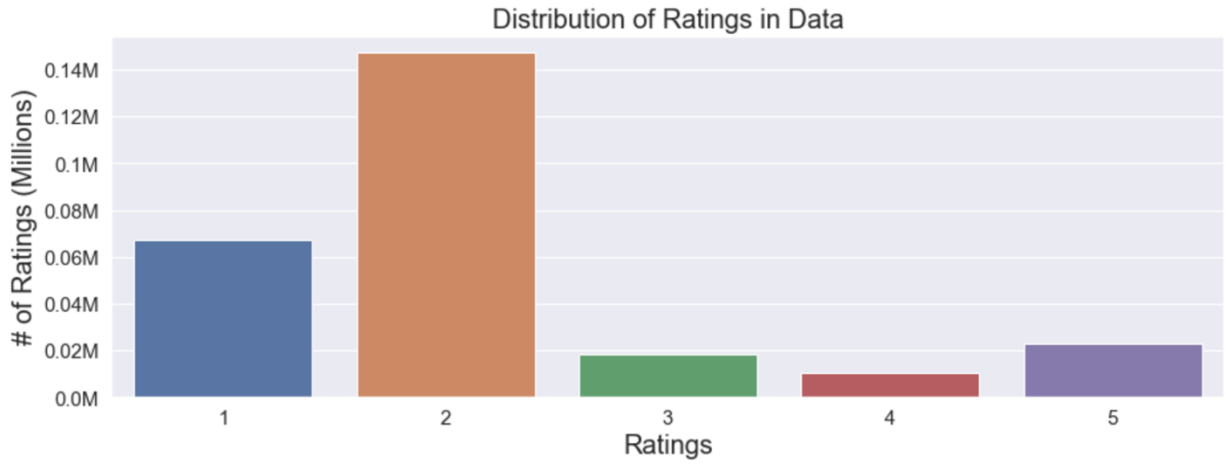
Since we don't have the number of votes, let's create them based on the assumption:
#votes = int(0.10 * #views);
The m, or minimal number of votes required to be listed is set to be 90% percentile of v;
C is easy to obtain by taking an average accross all ratings in the dataframe

Once the weighted average rating is computed, we can calculate the individual rating of the coub using the following logic:

- if the user only liked the coub, the result of WR is multiplied by 1, i.e. equals the WR;
- else if he reposted the coub onto his page, the WR is multiplied by 1.25.
- I limit the votes to 5 points at the most to avoid very imbalanced rating distribution (it appeared in our set ratings with >5 were very underrepresented, so I decided to put them into one category)

Okay, done with this. Here's how the rating distribution looks like:

Figure 5 – Distribution of ratings



As evident, most of the videos got rating of 2, followed by 1, while higher marks are less frequent.

While taking user preferences and ratings into consideration dynamically, the classical Collaborative Filtering has some issues. First, the main issue is scalability. The computation grows with both the users and the videos. The worst case complexity is O(mn) with m users and n items. In addition, sparsity is another concern.

One way to handle the scalability and sparsity issue created by collaborative filtering is to leverage a latent factor model to capture the similarity between users and items. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user. One common metric is Root Mean Square Error (RMSE). The lower the RMSE, the better the performance.

Figure 6 – Sample from the CF recommender

	coub_id	user_rating	id	channel_id	title	url_big
0	153478349	4.946647	153478349	1640331		https://coubsecure-s.akamaihd.net/get/b22/p/co...
1	151869218	4.946087	151869218	5336785	Sweet homes	https://coubsecure-s.akamaihd.net/get/b117/p/c...
2	150990021	4.938604	150990021	7185721	My nugget my nugget	https://coubsecure-s.akamaihd.net/get/b156/p/c...
3	151165731	4.937745	151165731	5609501	Rise shine	https://coubsecure-s.akamaihd.net/get/b4/p/cou...
4	153138733	4.937267	153138733	1397904	Sealed Loop	https://coubsecure-s.akamaihd.net/get/b163/p/c...

While switching to the user based rating system I used dimension reduction algorythm SVM and considered the recommendation problem as a regression problem minimizing the RMSE and MAE error. This approach has its advantages, since if some videos got rated high enough, most likely this user will rate them well too. Plus it's fast enough. Yet the disadvantage of this approach is the limited input data, essentially just the matrix of users and items with respect to their ratings. In the conclusions section I'll share some more ideas how and what could be done to improve initial results.

Conclusions

The purpose of this research notebook has been to demonstrate and try out the different approaches of the SOTA in the realm of the Recommender Systems. Given the nature of the data from *coub* I was able to achieve good results with the hybrid content-based approach, which would include the previous preferences of the given user. Once done I also provided the baseline collaborative filtering model that relies upon SVD with also pretty good results.

While researching the topic I encountered a more advanced approach, for which half of the job has been implemented in the initial version of this notebook, but had to be removed due to much time (over a day for some step and 18 hours for another) for needed to process and train the regressor. I'll briefly enlist now the steps it entailed:

- Once we have the user rating, create user-item matrix and similraty matrices: user-user, and item-item matrices.
- Instead of taking just the rating column advance it and prepare the following features: prepare Global Average of all video ratings, Average rating per User, and Average rating per Video, ratings for the given video by five similar users, ratings of five similar videos to the given one.
- Build different regressors on these data, compare and select best model.

This approach is described in the following resource:

How to Build a Recommender System(RS):

https://medium.com/datadriveninvestor/how-to-built-a-recommender-system-rs-616c988d64b2

Also, while implementing by me specified approaches, I now realize that approaches provided by me in this solution, merely provide the background for something more advanced, some much more sophisticated approach that truly goes into depth of the conent analysis and building similarity matrix based on feature maps exctrated directly from audio and video channels of the coubs. It would require paramaunt computing capabilities, but could potentially build a much more detailed profile of the user, his preferences and the profile of the videos themselves.

If I think about it, about how I would approach this challenge, the following steps come to my mind:

Simplified approach - extract key frames from video sequence

1. Switch from sequence data, which any video is, to a feature map representation of the video.
2. To do this, select certain number of key frames per video. Convolve them, use 1D convolutions to reduce the output dimensions.
3. For any coub compare each frame vector with vectors from all other coubs, select top similar videos by IoU.

The disadvantage of this approach would be the lack of context in each video, since the algorithm would compare merely the similarity of what it sees.

Thus:

More advanced approach - handle video sequence

1. Extract feature maps for the video sequence of specified length, process audio and video channhels separately. For audio you can look for videos with audio of similar MELS or magnitude. For video you can try video image captioning to extract text representation of each video. Add it to content based model.
2. Think of how you can combine the results of each channel. Add this info to coub ids.

Audio processing and recommendation based on audio:

1. **Deep Learning in Music Recommendation Systems** https://www.frontiersin.org/articles/10.3389/fams.2019.00044/full
2. **I Decoded the Spotify Recommendation Algorithm. Here's What I Found.** https://medium.com/@ericboam/i-decoded-the-spotify-recommendation-algorithm-heres-what-i-found-4b0f3654035b
3. **Content-Based Music Recommender Systems:Beyond simple Frame-Level Audio Similarity**http://www.cp.jku.at/people/seyerlehner/supervised/seyerlehner_phd.pdf
4. **Music Recommender System Based on Genre using Convolutional Recurrent Neural Networks**https://www.sciencedirect.com/science/article/pii/S1877050919310646
5. **NeuralTalk for video image captioning** https://medium.com/@samim/generating-captions-c31f00e8396e
6. **Dense Video Captioning Using Pytorch** https://towardsdatascience.com/dense-video-captioning-using-pytorch-392ca0d6971a

---

Unfortunately, we don't have internal statistics on coubs - like how often they're being watched, how long (taking into consideration it's a looped video and with right audio you may want to listen to it long enough), more details on user profile like location, gender, age, etc.

But you have an app with own video gallery, like Reface does, a lot of this data could be obtained. Which templates are being popular, how long are being watched, whether or not selected for refacing etc. Based on these preferences and applying one of the more advanced approaches above you could substantially enrich the potential database of templates and thus bring more variety into the app.

Recommender system could be applied to the reface template and user base as well, to suggest increasingly better / personalized templates based on his or her preferences, content similarity and its allignment to the picture of the person.

A fascinating source of data could also be a video emotion recognition. It seems to have been gaining popularity:

1. **Audio-video Emotion Recognition in the Wild using Deep Hybrid Networks** https://arxiv.org/pdf/2002.09023v1.pdf
2. **Video-based emotion recognition in the wild using deep transfer learning and score fusion**https://www.sciencedirect.com/science/article/abs/pii/S0262885617300367

If frontal camera use is permitted to observe users while using the app, an additional whole lot of information layer could become available.

These all tasks I would definitely love to try my hand on. I realize that there might be some major pitfalls in realizing ideas specified by me in this afterwords section, and some theoretical gaps might be present, given the starting point of not being an expert in video processing etc., but filling these gaps and making it happen could put the whole enterprise of recommending to a whole new level, on pair with Netflix or Youtube, which is an ambitious goal, yet realisable I belive.