Information Retrieval: Personal Recommendation based on YouTube comments

Tim Apers, Morgan Logghe & Sien Nuyens

October 2022

1 Definition

The goal of this project is to implement a system that, trained on YouTube comments, makes personal recommendations. Moreover, when presented with any piece of text, this could be another comment, a letter, an email or something else, the system will output video recommendations based on the textual information. We do not necessarily tackle any problem, but we are researching if the course "Information Retreival" can achieve interesting results in recommender systems based on textual information.

1.1 Scope

To train our model first we needed to generate a large amount of data consisting of YouTube comments within certain categories, which are listed in the 'Data' section. At first, we generated our own data-set using the YouTube Data API v3 [6]. However, later it turned out to be more useful to also work with a more complete Kaggle data-set.

Then, the classification model has the task to predict the categories using embedded comments. It is constructed out of the encoder followed by a Deep Neural Network. At last, the actual recommendation is made. This procedure exists out of a combination of two methods, which we will look at both individually and combined. At the end, we will conclude that the course can have a significant performance on recommender systems and that the processing of textual information can indeed achieve good results in terms of hit rate.

1.2 Resources

For this project, we used numerous libraries and platforms to help us make the implementation process a bit easier. Specifics regarding the usage can be found later on in the report where their contribution will become clear. Our code can

be accessed in the following Github repository: "https://github.com/timapers/project-mainf-IR" and the packages used are listed below.

- NumPy
- Pandas
- SciPy
- Sklearn
- Seaborn
- Tensorflow
- Tensorflow Hub
- langdetect
- Google Colab
- Keras
- Youtube Data API v3 [6]
- Kaggle data-set [5]

1.3 Workload

This project was a group project so we divided the work amongst the three group members. In the beginning of the project there were some struggles in the co-operation of one of the group members, Sien. But afterwards she gave it a 100% and the project was finished together.

Sien was responsible for generating the data that was needed to train the system. Tim and Morgan worked on the actual recommendation and classification model and also the evaluation metrics of that model. The final report was written together where the group wrote about their tasks individually and common sections were written together.

2 Data

The first step of our implementation is to generate the data we need as an input for our system. As explained before, we want to obtain a large data-set of comments from videos within specified categories on YouTube. Thus, to fetch all that data we chose to use the YouTube Data API v3 [6]. However because the Data API doesn't allow us to request such specific data in one go, we divided the generation in to different parts.

2.1 Category

First we specified in which categories we would be interested. As a start, we generated all categories but here we found not only different subjects but also subcategories of one specific subject like for example movie and multiple movie genres. We chose to leave that out and generated only the categories that are fundamentally different. Afterwards we also filtered out a few categories which were not applicable in Belgium meaning those categories didn't retrieve any videos. The resulting categories are the following:

- Film & Animation
- Music
- Sports
- People & Blogs
- Comedy
- Entertainment
- News & Politics
- How-to & Style
- Education
- Science & Technology

2.2 Video and Comment generation

After we obtained the categories we were interested in, we could begin with the generation of videos within those categories. However, unfortunately the YouTube Data API [6] does not allow us to specify that, the videos we are interested in, have to have at least a certain number of comments. This caused for some issues regarding the number of requests per day. So the retrieval of our data set took multiple days with multiple keys.

The YouTube Data API [6] allows per search request to retrieve a maximum of 50 videos. So, we decided we wanted to find 50 videos for each category. However, because we can only check after the request if the video is acceptable for our purpose, we had to keep track of the remaining amount of videos we still needed to search for. This resulted in multiple search requests per category to retrieve 50 acceptable videos for each category.
Because we did not want to have duplicates in our acceptable videos, we added a query to the search request. We took the longest word of the title of every video we had checked, and excluded it in this query.

As said before, after a search request was executed, we checked every retrieved

video on multiple bases. First we compared the video to all videos we already had checked to see if it was a duplicate. We do this check because the query we mentioned before didn't always work. Second we checked whether the title of the video was in English, since we only were interested in English videos and English comments.

Next, we had to request the commentThread of the video. The YouTube Data API [6] allows a maximum of 100 comments per request, which is the amount we wanted to retrieve. The retrieved comments were put in a list. First we checked if the list was empty, which would mean that the video in question didn't have any comments. Then we checked if the list contained at least 90 comments. This was a minimum threshold we chose ourselves because we wanted as many comments as possible. If the video passed all those checks, it meant that the video was acceptable for our purpose and so it was written to a file named videos.csv. The retrieved comments of that video were written to a file named comments.csv.

Because we had to generate all the data with multiple runs we had to introduce a few extra checks. We added a boolean value in the videos.csv file that was True when the video was accepted, but when a video was not accepted this was also written to the videos.csv file but with that value set on False. And we also check in the beginning if for that category we already completed the search and retrieved enough videos.

2.3 Resulting Data

Our generated data contained around 75,000 comments but unfortunately after the implementation was complete and using our data to train it, we still found that the data was not enough. So we searched for a better solution and found the Kaggle Dataset [5] and implementation. We compared to see how they were able to fix the limited amount of requests per day issue and found that instead of searching videos for specific categories, they searched for any video and then retrieved the information which also contained the category id. In the end we combined both datasets which would result in around 440,000 comments, to train our model.

3 Implementation

When implementing our model, we noticed that choosing a good encoder and using a simple cosine similarity was sufficient to achieve very good performance with our recommender. Therefore, the challenge of this project was to improve this simple model with the addition of a Deep Neural Network that classifies comments into categories. The general idea was clear, we had to accurately predict the category of the video from which comments originated such that videos of that category are more likely to be recommended. Training a good

classifier would then result in a better hit-rate and performance. Sadly, this was easier said than done.

After trying many different approaches, the final recommendation is an assembly of two methods. The first is a simple cosine similarity metric without the use of category classification. The second is a category biased recommendation that was calculated based on the prediction of our classifier. Using these two together resulted in a small increase in performance which will be evaluated more thoroughly in the Results section.

In general, the recommender model is divided into 3 parts. At the very beginning, an encoder transforms every comment to a 512-dimensional embedding making use of a Deep Averaging Network [3]. Following this, a Deep Neural Network learns to categorize these embeddings into one of the 10 video categories. Finally, when given a comment or other text fragment (query), a recommendation is made based on the cosine similarity between the embedding of the query and the embedding of the document, which is the average embedding of all comments from that video. For the second method of recommendation that also takes the classifier into account, the same similarity is calculated, but only for videos of the predicted category.

3.1 Comment Embedding

In this course, we have seen multiple methods to transform sentences into a vector of real numbers. We started with simple algorithms like TF-IDF and Bag-of-Words and saw more advanced topics later on in lecture 7 such as Word2Vec and Transformers. Because we want to use a Neural Network to classify categories, simple methods will not work as the dimensionality will get to large for the model to interpret. Thus, we are required to explore the options from lecture 7. This way we can also capture the semantics and context of a comment besides important keywords.

When trying out Word2Vec we noticed we could achieve sufficient results for classifying comments, but we wanted to go a bit further and see if we could make any further improvements. We came across the paper from Cer et al. which describes 2 approaches for generating embeddings which are specifically targeted for transfer-learning purposes. This meant that we could easily follow up the encoder with a Neural Network as a down stream task. From the 2 approaches that were mentioned, we ended up implementing the Deep Average Network [2] approach which generates a 512-dimensional embedding of a comment. The Transformer based sentence encoder achieved inferior results and in addition the paper also mentioned that DANs achieve strong base-line performance on text-classification tasks.

More specifically, for the implementation of the encoder we used a DAN embedding model that was made available on Tensorflow hub, a platform that provides

numerous pre-trained models. [4]

3.2 Category Classification

Building our classification model was mostly based on the contents of the "Artificial Neural Networks" course given by Prof. Oramas. As mentioned before, this is a Deep Neural Network that exists out of the encoder followed by four fully connected layers. The first three dense layers have a ReLU activation function whereas last will use a softmax activation function such that we can output a probability for each of the categories.

After extensive experimentation we concluded that a configuration with a first dense layer of 4096 neurons followed by one with 2048, then 1024 and finally 10 output neurons, which represent the categories, yields the best results. To deal with overfitting, we also introduce a drop out layer after every dense layer that has a drop out chance of 50%. Lastly, to train this model, we froze all the layers of the encoder as making these weights trainable would result in extreme overfitting and we would lose the generalisation that the encoder represents. Furthermore, we made use of an ADAM optimizer with a learning rate of 0.001 and categorical cross entropy loss function.

For the training procedure of our classifier, we ran our code on a Google Colab GPU where we combined Python's deep learning API, Keras, and the Tensorflow machine learning package.

3.3 Recommendations

For the final part of the recommendation system we simply used the following similarity metric which was also mentioned in the lectures:

$$sim(embed_c, embed_v) = 1 - \frac{arccos(\frac{embed_c \cdot embed_v}{||embed_c|| * ||embed_v||})}{\pi}$$

This is a cosine based similarity metric where $embed_c$ represents the embedding of the query (comment) and $embed_v$ is the average embedding of all the comments belonging to the video (document). We use the angular distance rather than the raw cosine similarity as it better distinguishes between nearly identical embeddings[2][1]. For this part we did not experiment much with other metrics. This means there is a good possibility that better metrics exist for our case.

Now, we mentioned earlier that to make the actual recommendation we use an combination of two different recommendations. The first recommendations are the highest scoring videos that are closest to the query according to the similarity score. The second follows the exact same procedure, but in this case we only compute the similarity scores for videos that are of the predicted category of our classifier, the rest of the scores are set to 0. To then assemble the

final recommendation we alternately choose the highest recommendation from the first recommender and one from the second until the maximal number of recommendations have been reached.

The data and scores were processed and calculated using the pandas and NumPy packages.

4 Evaluation

For the evaluation of our model we can evaluate 2 parts. The first will be the classifier model where we can use unranked evaluation methods and list the confusion matrix from which we can obtain the precision and recall for each category. For the second part, which is the recommender itself, we sadly cannot make use of ranked evaluation methods as every comment only corresponds to one single video. This means that we do not know if multiple videos are relevant to the search query such that we can not construct a PR-curve or similar metrics. Additionally, we do not have the infrastructure to perform user-centric methods as we are not affiliated with YouTube. As a result, we ended up using the hit-rate as an evaluation method by determining the percentage of times the video from which we picked a test query appeared in the recommendation for that query.

5 Results

In the results section we will extensively go over the training process of our classifier and our final recommendation system. As mentioned earlier, we will also list the confusion matrix of our classifier and calculate the precision and recall for each category on a test set of 10000 comments. We end this section by comparing 3 recommendation systems on their hit rate. These will be the recommender without utilisation of the classifier (DAN+COS), with the classifier (DAN+DNN+COS) and the combination of the two (Combined).

5.1 Classifier: DNN

We let the Deep Neural Network train for 10 epochs. However, from the data below we can conclude that around 5 epochs suffice as the validation loss begins to increase again meaning the model starts overfitting. A training output of the Deep Neural network looked as follows:

- loss: 1.5421 - accuracy: 0.4272 - val loss: 1.4223 - val accuracy: 0.4728
- loss: 1.3726 - accuracy: 0.4958 - val loss: 1.3454 - val accuracy: 0.5070
- loss: 1.2681 - accuracy: 0.5391 - val loss: 1.3166 - val accuracy: 0.5187
- loss: 1.1761 - accuracy: 0.5752 - val loss: 1.3071 - val accuracy: 0.5259

- loss: 1.0914 - accuracy: 0.6064 - val loss: 1.3114 - val accuracy: 0.5228
- loss: 1.0155 - accuracy: 0.6348 - val loss: 1.3143 - val accuracy: 0.5286
- loss: 0.9411 - accuracy: 0.6612 - val loss: 1.3176 - val accuracy: 0.5310
- loss: 0.8756 - accuracy: 0.6853 - val loss: 1.3439 - val accuracy: 0.5329
- loss: 0.8135 - accuracy: 0.7071 - val loss: 1.3650 - val accuracy: 0.5311
- loss: 0.7535 - accuracy: 0.7281 - val loss: 1.4098 - val accuracy: 0.5300

Now, to evaluate the performance of the classifier a bit more in depth we compose the confusion matrix based on 10000 test comments. The x and y axis correspond to categories (an overview can be found in Table 1). From the confusion matrix below we can see that the diagonal is clearly outlined which is a good thing as it represents that our classifier makes accurate predictions. Additionally, we notice that not all categories have an equally distributed number of comments. For example, category 24 - Entertainment has a lot more comments than 1 - Film & Animation. In order to easily generate the heatmap we used the seaborn package.

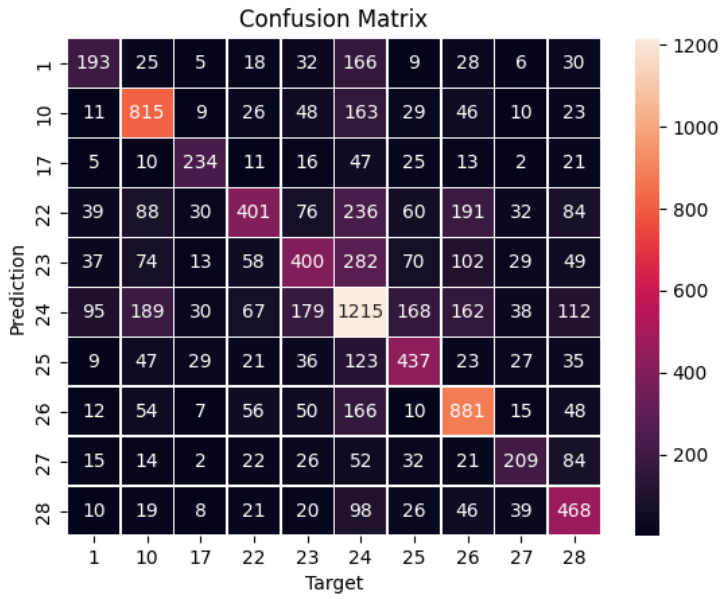

Figure 1: Confusion Matrix

We are not really interested in the confusion matrix, but rather the metrics we can derive from it. Hence, we can now also compute the precision, recall and f1-score. These values can automatically be calculated by the classification_report

function from sklearn. When looking at the recall column, we clearly see that our classifier is good at some categories such as 10, 17, 25, ... but rather bad at others 1, 22, 23. We assume that these will have a lot of similarities that makes it hard for our model to distinguish between them. Furthermore, from the precision we do not really notice that it underperforms heavily on certain categories, that is why it is always to important to look at both of the metrics since they are heavily correlated. This is the reason we also computed the F1-scores which capture this underperformance quite nicely in that it produces some low scores around 40% and some higher scores 60% that clearly indicate our model is a bit lost in classifying those categories. The course mentioned to not use accuracy as a metric in Information Retrieval, but we still think we must mention that our classifier achieved a 53% accuracy.

Table 1: Evaluation metrics of the classification model for the different categories

	Precision	Recall	F1-score
1: Film & Animation	0.45	0.38	0.41
10: Music	0.61	0.69	0.65
17: Sports	0.64	0.61	0.62
22: People & Blogs	0.57	0.32	0.41
23: Comedy	0.45	0.36	0.40
24: Entertainment	0.48	0.54	0.51
25: News & Politics	0.50	0.56	0.53
26: How-to & Style	0.58	0.68	0.63
27: Education	0.51	0.44	0.47
28: Science & Technology	0.49	0.62	0.55

5.2 Recommenders

In the table below we describe the hitrates of the two recommendation approaches we used. That is, for how many of the comments can the video from which it originated be found again in the recommendation. As mentioned before, the first recommendation refers to just the angular distance between the embeddings (DAN + COS) and the second will be after introduction of the classifier (DAN + DNN + COS). In the last row the hitrate of their combination is listed. We ran the system for k=2,4,6,8,10,20 referring to the amount of videos in our recommendations.

As for the interpretation, we can see that the simpler solution, not using a category classifier, yields better results than leveraging this Deep Neural Network for all k. Additionally, even though our classifier achieved an accuracy of 53%, the final recommendation only scored a hit rate of around 40% for high k. This can imply that we need to improve our scoring method to achieve the same accuracy that the classifier achieved. Furthermore, notice that the combination

of the two did not improve the results of the simplest recommender. We thus did not succeed in improving the recommendation system by using an extra classifier on top of the DAN encoder and angular distance metric. Nevertheless, we still want to make the case that, if we had a better performing classifier, then it has a fair shot of improving the simple recommender when we combine the two.

Table 2: Hit rates of our 3 recommendations systems for different k

Recommender	k=2	k=4	k=6	k=8	k=10	k=20
DAN+COS	0.337	0.412	0.451	0.480	0.501	0.574
DAN+DNN+COS	0.281	0.329	0.353	0.369	0.381	0.418
Combined	0.335	0.410	0.454	0.4803	0.503	0.572

6 Limitations & Challenges

In theory the project idea started out simple enough. However, when actually implementing it, we started hitting some roadblocks. In this section we will go over a few important challenges we faced and the limitations of the project we encountered. It also explains why our Deep Neural Network approach did not obtain the results we expected.

6.1 Data Quality

Data Quality was a first difficulty we encountered when working on the project. As said before, we first started by generating our own data using the YouTube Data API v3 [6]. Initially, this seemed a good option for acquiring data as this provided us with relatively recent videos and comments. However, the usage quota of the API were a first restriction in generating a sufficient amount of data. Nevertheless, we gathered around 75 000 comments to train our model on.

Upon closer inspection though, we noticed a great deal of non-English comments still emerging in the results despite specifying the language region in the API-call. As a consequence, more pre-processing of the data was required. After filtering out the unusable comments and also disregarding punctuation and emoji's, the model gradually improved but the overall results remained the same. That being the case, we had to search for another manner to obtain more data.

We then came across the Kaggle data-set [5]. This data-set contained exactly all of the information we needed which made it easy to adapt our code to work with the new data. With 440 000 comments the data better represents the comments for the different categories. However, in this data-set as well, the pre-processing mentioned earlier was still a necessity. After this step, we had approximately 160 000 comments. What we can take away from all this is that,

if we want to build a well performing text classification model, a large amount of data needs to be available and must be thoroughly processed to improve data quality. We think we would have achieved better results if we had access to more and qualitative data.

6.2 Scaling

More data comes at a cost though, it is important to note that these kind of prediction models are extremely data hungry. We saw the performance of the classification model dramatically improve when provided with more data. However, because of the limited amount of GPU available on our own computers and google colab's machines, it was just not feasible to execute multiple runs with all data. Therefore, since we had limited access to data we made the train and validation set as big as possible in comparison to the test set. Moreover, the train and validation set made up around $> 90\%$ of our total data, while the test set contained a maximum of only 10000 comments.

This caused the effect that, working with large data quantities is prone to long computation times. This project is not an exception. To save valuable time, we worked with pandas DataFrames and NumPy arrays to easily manipulate the data and introduce matrix multiplication in an efficient way. Nevertheless, pre-processing and the evaluation of our model formed a bottle-neck regarding the runtime. Hence, instead of taking into account every single comment, we sampled only a small fraction of comments to be in the test set. As a result, the code was able to run much faster.

6.3 General Comments

Another problem we encountered were general comments. We see that many comment sections on YouTube video's exist out of comments such as 'Wow, such a great video!' or 'You are my favorite youtuber'. Comments like this can be found in each of the categories and do not characterize one specific category. Since we did not make any effort to filter out these kind of comments, we can assume that they are partially responsible for the lower performance of our classifier. We thus think that if general comments were filtered out of the data set we could also obtain better results.

6.4 Overfitting

The last challenge to overcome, was overfitting. We noticed that if we did not freeze the layers of the encoder, our classifier would overfit almost instantly. Adding the drop out was also an important measure to reduce overfitting. In addition, we increased the batch size to train the model at a more steady pace.

References

[1] Nominal Animal (https://math.stackexchange.com/users/318422/nominal-animal). *Cosine similarity vs angular distance*. Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/2876675 (version: 2018-08-13). eprint: https://math.stackexchange.com/q/2876675. URL: https://math.stackexchange.com/q/2876675.

[2] Daniel Cer et al. *Universal Sentence Encoder*. DOI: 10.48550/ARXIV.1803.11175. URL: https://arxiv.org/abs/1803.11175.

[3] Mohit Iyyer et al. "Deep Unordered Composition Rivals Syntactic Methods for Text Classification". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1681–1691. DOI: 10.3115/v1/P15-1162. URL: https://aclanthology.org/P15-1162.

[4] *TensorFlow Hub*. URL: https://tfhub.dev/google/universal-sentence-encoder/4.

[5] *Trending YouTube Video Statistics and Comments*. Oct. 2017. URL: https://www.kaggle.com/datasets/datasnaek/youtube.

[6] *YouTube Data API —*. URL: https://developers.google.com/youtube/v3.