YouTube Comments Sentiment Analysis

IM 5054 Introduction of Text Mining Final Project - Group 14

Brian Chen†
Dept. of Information Management
National Taiwan University
Taipei, ROC

Brandon Yeh
Dept. of Information Management
National Taiwan University
Taipei, ROC

Brian Kuo
Dept. of Information Management
National Taiwan University
Taipei, ROC

Alison Liu
Dept. of Information Management
National Taiwan University
Taipei, ROC

Dragons Lai
Dept. of Information Management
National Taiwan University
Taipei, ROC

PREFACE

Starting December 13th 2021, YouTube removed the ability to see dislikes from their API. The platform claims it's made the change to protect users from "dislike bombing" and to reduce stress and embarrassment for creators. While most people are doubtful about YouTube's change to protect creators from stress, since creators are still able to see the number of dislikes for their videos, this is unhelpful for the users. The number of dislikes has always been one of the important indicators that provide users a brief view of how good the video is, offering users a chance to avoid click bait or unhelpful tutorial videos. This removal not only removes dislikes, but also removes any ability to judge the quality of content before watching.

In order to deal with this problem, the purpose of our project rises to the surface: We hope to come up with another way to determine the quality of a YouTube video.

1 Our Solution

Our solutions can be broken down into four parts: Data Collection, Data Preprocessing, Model Building, and Application. Let's examine them one by one.

1.1 Data Collection

Obtaining comments for a YouTube video is not as complicated as it sounds. YouTube does offer an API for that purpose, like what has been addressed in [3], but since we are continuously scraping massive amounts of data from YouTube, a free tier API may not be sufficient for this purpose. Repositories like [1] show an alternative of getting the YouTube comments effectively without APIs. We modified the code to fit our needs, and all are stored within the `yt_helper/comment.py` file. Since now we do not get data from the official API, the date and time for comments might not be precise, and thus time-related methods like LSTM should be implemented with caution.

We implemented a function that returns raw comment data for any specified video, given the video ID and limit. The crawling process can be done in runtime within a couple of seconds, helping us achieve the further implementation of dynamic inputs from users.

We also used the same method to get the metadata of a video, including the title, the channel name, and the view counts. Also keep in mind that since YouTube video's URLs often come in different formats and styles, we implemented a parser with regex to handle the job for us.

Packages used here are: Pandas, Request, and re (regex).

1.2 Data Preprocessing

Comments on YouTube may not be formal or grammatically correct. Instead, there may be some slang or trendy words that are unexpected or unknown as compared with traditional articles. Other than that, data fetched from the crawler are all raw data, which must be preprocessed into an understandable format before throwing it into the model. More importantly, comments on YouTube include emojis, thus it is crucial to properly identify emojis and emoticons during data preprocessing, then transforming them into text for proceeding analysis. Thus, we need the preprocessing part between crawling and modeling to make sure data is in general form to make follow-up training more precise and easier.

The data preprocessing part consists of several steps: removing duplicate comments with the same comment ID, detecting emojis and modifying them to words, and removing brackets and special characters. We return the output in the form of DataFrame for the incoming training part.

Also please note that since our models are heavily based on English, we choose to return English comments only.

Packages used here are: Emoji, Demojize, re (regex), and Langdetect.

1.3 Model 0: Using TextBlob, a simplified text processing tool

We'd found some awesome projects for processing textual data, TextBlob being one of them. [4] states that TextBlob is based on NLTK and pattern [6]. Simply make a sentence (in our use case, a "comment") to a TextBlob object, and we can get the polarity for the particular sentence by calling the `polarity` attribute of the object. This is a rather simple implementation, so in our source code, we've put them under the Data Preprocessing part.

Packages used here are: TextBlob

1.4 Model 1: Learn Word Embeddings Jointly with the Task (Fully-Connected Network)

In this model, we adopt the fully connected network as model architecture (Figure. 1.), and then use the IMDB dataset as training data. In the training process of binary sentimental classification (positive or negative), our model learns how to transfer texts to meaningful word embeddings. Therefore, we can then use the model to predict the sentiment of incoming YouTube comments.

```
_________________________________________
 Layer (type)           Output Shape         Param #
=============================================
embedding (Embedding)  (None, 200, 300)    6000000


 flatten (Flatten)        (None, 60000)         0


 dense (Dense)            (None, 16)           960016


 dense_1 (Dense)           (None, 1)             17

=============================================
```

Figure 1: Model 1's architecture

Packages used here are: Keras and Numpy.

1.5 Model 2: Using pre-trained word embedding (BERT & SVM)

In this model, we also use the IMDB movie reviews dataset as our training data, However, instead of using context-independent embeddings like word2vec, we get context-dependent embeddings from BERT. Hence, we can train the model only through the embeddings of [CLS]. In the training process of binary sentimental classification (positive or negative), our SVM model learns how to label each training data correctly. Therefore, we can then use the model to predict the sentiment of incoming YouTube comments.

Packages used here are: keras-bert and Numpy

1.6 Streamlit

We noticed that models alone do not help facilitate the analysis process for public users with no background in coding. Thus, we built an interactive visualization tool to achieve the goal using Streamlit. We meant to show users how our models "think" about a video based on the comments we received. We take the traditional approach to visualize the ratio using pie charts.

The application will show the outcomes obtained from the aforementioned models.

Packages used here are: Streamlit and Plotly

2 System Outcomes

2.1 Model 0: Using TextBlob, a simplified text processing tool

Much to our surprise, the general implemented API does a pretty decent job in determining the polarity of comments. In terms of accuracy, TextBlob provides 60% of accuracy when compared to our manually labeled data.

2.2 Model 1: Learn Word Embeddings Jointly with the Task (Fully-Connected Network)

Notice that we make an assumption that YouTube comments are intrinsically similar with the IMDB comments. To verify the assumption, we labeled 200 YouTube comments as testing data, and the performance is about 70 percent of accuracy. However, the auc and F-score are quite low, which means that the model may be unstable sometimes. (Figure 2) Therefore, we believe either that the assumption is not solid or that we should revise the model structure to further improve the model performance.

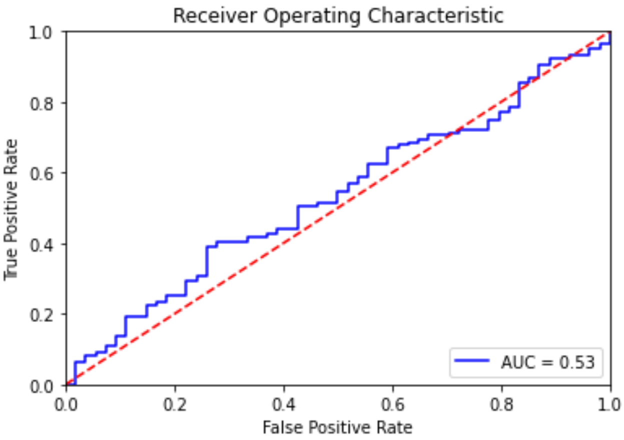


Figure 2: Model 1’s ROC curve

2.3 Model 2: Using pre-trained word embedding (BERT & SVM)

Using the same assumption and the same testing data of 200 YouTube comments we labeled. The performance between different BERT model sizes is different. When we choose the largest BERT-Base model, the accuracy can reach nearly 80 percent. However, due to the restrictions of the deployment environment, we finally chose the mini-BERT model to speed up our training, testing, and predicting process. Finally, we got 70 percent of accuracy. The auc and F-score are also not as good as we thought (Figure 3). In addition to the conclusion as in model 1, in this BERT model, we also notice that better model performance takes time.

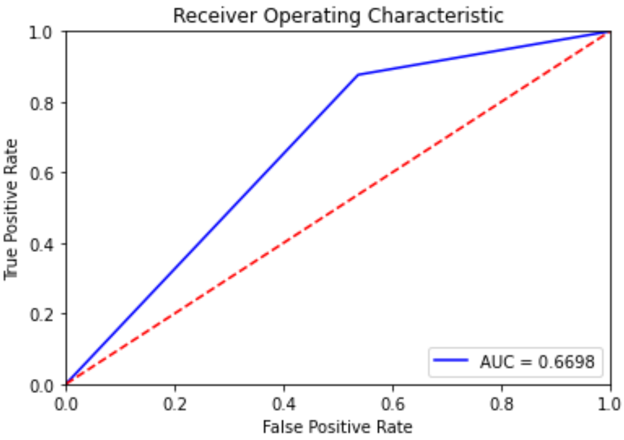
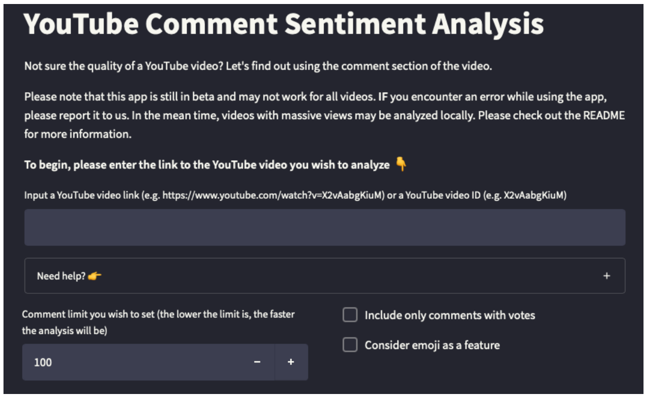


Figure 3: Model 2’s ROC curve

2.4 Streamlit

As shown in the screenshot (Figure 4), the application is relatively intuitive. A user can choose to type in a video link or a video ID. Then, the user should specify the limit of the comments downloaded (if too large, he/she should expect an extremely long wait). The user can also toggle the “Include only comments with votes” or “Consider emoji as a feature”, which will leave all emojis in the comments for training.



YouTube Comment Sentiment Analysis

Not sure the quality of a YouTube video? Let's find out using the comment section of the video.

Please note that this app is still in beta and may not work for all videos. If you encounter an error while using the app, please report it to us. In the mean time, videos with massive views may be analyzed locally. Please check out the README for more information.

To begin, please enter the link to the YouTube video you wish to analyze 👇

Input a YouTube video link (e.g. https://www.youtube.com/watch?v=X2vAabgKiuM) or a YouTube video ID (e.g. X2vAabgKiuM)

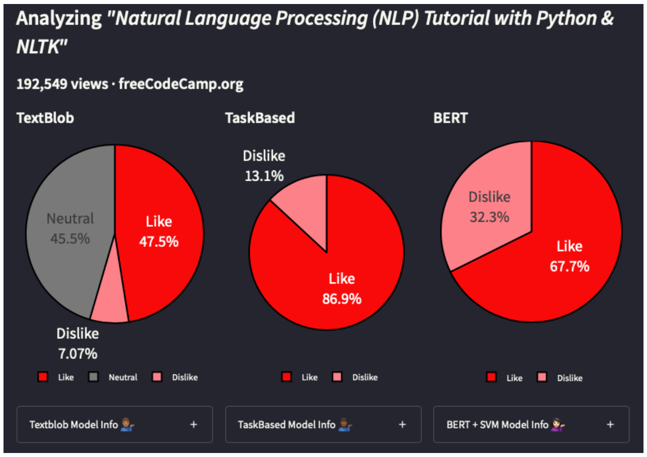
Comment limit you wish to set (the lower the limit is, the faster the analysis will be): 100

□ Include only comments with votes

□ Consider emoji as a feature

Figure 4: Our Streamlit interface

We first can notice a big difference in the runtime of our models. TextBlob (Model 0) has the fastest response time with Model 1 coming up a bit longer by no longer than five seconds. Model 2 takes longer time to generate when compared to other models. The result can be seen in the following screenshot (Figure 5).



Analyzing *"Natural Language Processing (NLP) Tutorial with Python & NLTK"*

192,549 views · freeCodeCamp.org

Figure 5: Analyzed result of “Natural Language Processing (NLP) Tutorial with Python & NLTK”

The plots are all interactive, and one will be able to expand the information section for each model to learn more about the mechanics behind.

Nonetheless, we observed that in our deployed version, for videos over 100 million views, the crawling process and

Streamlit's hosting cache system are not able to provide their services. Since we are using a free-tier version of deployment service (directly on Streamlit's server), those are the restrictions we must bear with. For now.

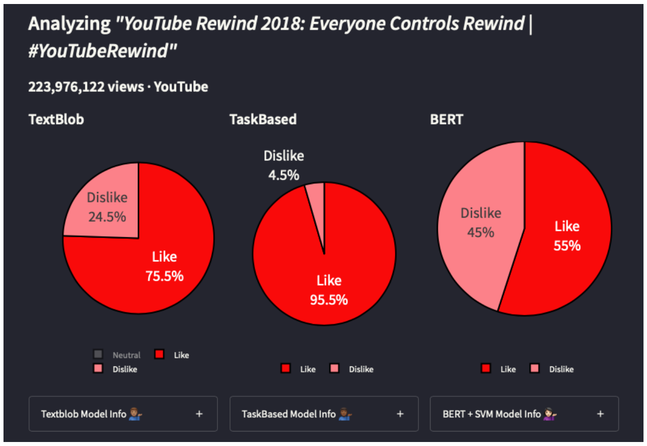
For those videos, we expect users to analyze them locally using the instructions provided in our project's repository. They all work fine in a local environment.

3 Conclusions

3.1 Discoveries

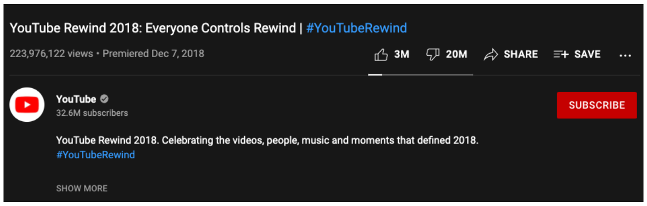
In this project, we provide users a sentimental analysis of comments to help them assess the quality of YouTube videos when or before watching; therefore, providing users with a chance to know the like-dislike ratio even when those data are hidden by YouTube itself.

However, we notice that the relationship between comments and the number of likes and dislikes seems to be complementary rather than replaceable. To be more specific, it is possible that there are more people clicking dislikes, but at the same time most comments are actually relatively positive. Take Figure 6 for example, there are 3M likes versus 20M dislikes (Figure 7), but most people leave comments that do not show dislikes regarding the video. There are, though, some circumstances that we fail to handle. We will cover that in the Visions section.


Analyzing *"YouTube Rewind 2018: Everyone Controls Rewind | #YouTubeRewind"*

223,976,122 views · YouTube

Figure 6: Analyzed Result of "YouTube Rewind 2018: Everyone Controls Rewind|#YouTubeRewind"


YouTube Rewind 2018: Everyone Controls Rewind | #YouTubeRewind

223,976,122 views • Premiered Dec 7, 2018

YouTube Rewind 2018. Celebrating the videos, people, music and moments that defined 2018. #YouTubeRewind

Figure 7: Actual Like and Dislike Ratio of "YouTube Rewind 2018: Everyone Controls Rewind|#YouTubeRewind"

Whether the number of likes and dislikes is hidden or not, our service can help users to take a look at how others think of the videos. But still, it would be better if they can have both pieces of information.

3.2 Visions

3.2.1 Turn the app into a simple Plug-In. We believe that this application can provide users who are determined to figure out whether a video is regarded as good or bad in an easier and faster way. Now, however, due to the time constraint, the application is built into an individual website, which can be improved. A user now will have to copy and paste the link from YouTube to our site, ending up being more troublesome and less user-friendly.

Therefore, we have the vision to turn this application into a simple browser plug-in or extension like [2]. A browser extension can provide instant feedback to the video while watching, thus having the similar user experience to what we were used to before the removal of the like-dislike ratio. We expect increasing usage rate and more feedback from users so that the overall performance can be improved.

3.2.2. Ironic Sentence Detection. To realize ironic sentence detection, we have to analyze video content first. However, it involves some advanced techniques beyond the scope, such as video content analysis, object detection, speech recognition, etc. But with such functionalities, the "actual" feeling or sentiment on a video can be retrieved.

3.2.3 RNN model implementation. From our last lecture, we realize that Recurrent Neural Network (RNN) performs well in analyzing sequential data such as documents, articles and comments. We ended up building two-and-a-half models (we consider Model 0 one half of a model), using W2V and BERT, and we'd been too late to recognize such an approach. In our preprocessing step, we implemented to store those time-oriented information and sequential traits, including commenting time and the type of a comment (whether it's an initialized "comment" or a "reply" to a comment). We hope that this task can be accomplished someday.

3.2.4 More Labeling Data. In this project, due to our shortage in manpower, we only manually labeled 200 data points despite the fact that we knew that this step was extremely important in training and evaluating our models. We hope to obtain those “labeled data” from all over the world like what Google did in its Crowdsource project [5]. This can be realized by either setting up an independent website designated for this job or through an extension that prompts a user to score their “comments”. More relevant data increases the probability that our models contain useful information, and we expect to see improved models with higher accuracy.

REFERENCES

[1] Egbert Bouman. 2022. youtube-comment-downloader. GitHub. Retrieved January 13, 2022 from https://github.com/egbertbouman/youtube-comment-downloader

[2] Dmitrii Selivanov. 2022. Return YouTube Dislike. GitHub. Retrieved January 13, 2022 from https://github.com/Anarios/return-youtube-dislike

[3] William Yang. 2020. How to Build Your Own Dataset of YouTube Comments. Medium. Retrieved January 13, 2022 from https://towardsdatascience.com/how-to-build-your-own-dataset-of-youtube-comments-39a1e57aade

[4] 2018. TextBlob: Simplified Text Processing — TextBlob 0.15.2 documentation. Readthedocs.io. Retrieved from https://textblob.readthedocs.io/en/dev/

[5] 2018. 玩遊戲也能貢獻智慧！到 Google Crowdsource 解任務讓 AI 更聰明 | iKala Cloud. ikala.cloud. Retrieved January 13, 2022 from https://ikala.cloud/google-ai-crowdsource/

[6] 2020. Hands-on Guide to Pattern - A Python Tool for Effective Text Processing and Data Mining. Analytics India Magazine. Retrieved January 13, 2022 from https://analyticsindiamag.com/hands-on-guide-to-pattern-a-python-tool-for-effective-text-processing-and-data-mining/