Youtube Videos Trending Prediction and Analysis

Zheyuan Hu, Zixuan Shao, Yang Gao
New York University, New York, NY 10003
zh2095@nyu.edu, zs2167@nyu.edu, yg2410@nyu.edu

Abstract

As YouTube becomes the most popular video-sharing platforms, YouTuber is developed as a new type of career and is growing over years. The revenue of a YouTuber depends on the popularity(views) of the video he created. Therefore, it is valuable to figure out the factors that determines the popularity of a video. In this project, we are going to discuss how we analyze on the dataset we scraped from YouTube, and try to build a model to predict the popularity of a YouTube video. The qualitative result shows that our best model stacked from Random Forest, XGBoost and Gradient Boosting has a considerable performance. This implies that our model is capable of solving this business problem.

1 Business Understanding

1.1 Background Introduction

Along with the booming development of the Internet industry, people's lifestyles are already intertwined with the Internet and are changing accordingly. Today, a plethora of online entertainment options has led to the change of people's leisure and recreation, and among these options, watching online videos has become one of the most popular forms of entertainment among young people. YouTube, as the largest video-sharing platform, and the 2nd most visited site in the world, is undoubtedly the first choice of watching videos for many people. The success of YouTube is due to its wide variety of videos that everything you are interested in can be found there, and this is contributed by those who share their lives and create the contents on YouTube, so called YouTubers.

As a new type of profession, YouTuber is fun, low threshold and lucrative, so that attract more and more young people to join these years. As of 2020, there are more than 37 million YouTubers, which grew more than 23 % compared with last year. Professional YouTubers make money through advertising revenue, corporate sponsorships, and merchandise sales, and all of these are determined by the number of subscribers a YouTuber has. Currently the highest paid YouTubers are two brothers named Vlad and Nikita, 7 and 5 years old who have 70 million total subscribers, and they can earn up to $37 million per year. To get more subscribers requires a YouTuber to create videos that can be seen by more people, that is, to have more views. Therefore, the popularity of a video (views) is what YouTubers are most concerned about.

1.2 Goal and Business Value Analysis

In this project, we are going to figure out what factors are going to affect the views of a YouTube video and try to predict its future popularity. Specifically, we would like to build a model that can take in the information of an uploaded video and a future date and then give a prediction of its view count on that date. This model can be implemented as a new feature on the YouTube website/app, which can be a good reference for the video publishers to have a preliminary prediction on their videos' performance so that they might adjust their video to obtain more public attention.

As for the platform YouTube, the business value of this feature lies on its consequences that it might attract and fix more YouTubers by enhancing their user experience, and incent them to create better contents so that brings more traffic to the platform. Meanwhile, the future popularity of the video can be included as a new dimension of the recommendation system for trending videos. Currently, the algorithm for YouTube trending videos is based on the video clicks in the recent past and it does not take the prediction of future clicks into account. We believe that what is truly popular should stand the test of time, so incorporating the future popularity into the algorithm might make the trending recommendation more valuable. Although we have seen some analysis examining factors that influence video trending, none of them wants to predict video trending and turn it into a valuable business model.

2 Data Understanding

2.1 Preliminary Understanding

Our initial thought is to use a dataset on Kaggle called Trending YouTube Video Statistics, which is scraped by Mitchell J in 2018. This dataset consists of the information of YouTube daily trending videos in several months. Almost all of the video information except the video content has been recorded in this dataset, including the video title, channel title, publish time, tags, views, likes and dislikes, description, comments, etc. Intuitively, many of these features are related to the view count, so this dataset seems to be a good source to build our predictive model. However, if we are going to use this dataset to address our business problem, then three serious issues would come out.

Firstly, this dataset was scraped two years ago, whereas people's concerns and the factors that influence the popularity of video are changing over time, in other words, what was popular two years ago is obviously not the same as it is today. Thus, we are not going to build a model that keeps up with the current trends based on this obsolete dataset. Secondly, all of the videos in this dataset are trending videos, that is, they all have a decent amount of views, mostly exceeding 100k. This would result in our model being unsuitable for common videos, hence greatly affect the generalizability of the model. Besides, this dataset comes with a selection bias. With the limitation of 200 trending videos daily, the dataset only contains a few thousands of videos for each region and most trending videos appear multiple times in several days.

Therefore, in order to build a timely, generalizable and less biased model, we decided to scrape our own dataset.

2.2 Data Scraping

We built our own scraper referring to the source code provided by the creator of the Kaggle dataset. The scraper is built based on the Google Youtube v3 API, a source to scrape all kinds of information available on Youtube via Google Cloud API key. Our initial idea is to scrape random videos from Youtube, which would be the ideal balanced and general dataset for our project, but Youtube API does not yet support the feature that could extract random video information from Youtube. The only way we could obtain a large amount of data from video is using the combination of "search" and "video" result from Youtube API. In other words, "search" will return a list of video snippets such as video id and channel id from the search keywords, then we use the video id we got from "search" to obtain comprehensive video information from "video" API section. To simulate the distribution of random video from Youtube, we use some keywords of categories to generalize some specific topics such as sports, game, music and film etc. Typically, returned content from the API often contains 500 to 600 videos for each keyword, so we somehow made up nearly one hundred keywords to get around 40k distinct videos. If we want even more data and yet run out of ideas, same keywords might be used and duplicate videos could be handled during preprocessing. The features of the data we scraped is the same as that of the Kaggle data as we use the same function of the transformation.

2.3 Data Cleansing

Table of All Variables	
Variable Name	**Example**
video_id	2L6gsn7rGqI
title	Euro Coach Bus Simulator 2020: City Bus Driving Games
tags	sidhu moose wala| game official video| shooter kahlon|...
description	5911 Records Presents. Song : GameSinger /Lyrics...
publishedAt	2020-09-03T 02:30:13Z
scrapedAt	20.23.11
channelId	UCV4iYJ7wTDWr RGbGZEvQsSg
channelTitle	5911 Records
categoryId	10
thumbnail_link	https://i.ytimg.com /vi/2L6gsn7rGqI /default.jpg
view_count	57304357
likes	1615585
dislikes	78750
comment_count	4970339
ratings_disabled	False
comments_disabled	False

After scraping from YouTube for about one week, we got a raw dataset that contains 36k videos with all kinds of categories in the US region, which we considered as a good fit for our predictive model. The table above shows all the 16 variables that has been contained in this dataset with an example of each feature.

Our target variable here is view_count, and other variables are going to be potential features. Looking into the data, we noticed that some videos have been scraped repeatedly, so we first dropped those duplicate rows, and around 33k data left after duplicates removal. Then we were lucky to find that there is no null cell in our dataset, so we didn't have to apply the null replacement. After that, we decided to drop the columns that have nothing to do with the views, including thumbnail_link, video_id, channelId, and channelTitle.

2.4 Exploratory Analysis

After some basic cleansing, we started to search for the relationship between our target variable view_count and the features. Based on intuition and life experience, we proposed the following three conjectures and tried to verify them with some EDAs:

1. Those counted variables (likes, dislikes, comment_count) are related to our target variable.

2. The further back the date of the video's release, the more views the video will get.

3. Some categories of videos tend to be more likely to prevail than the others.

It is easy to check for the first conjecture since all of these variables are numerical variables, we plotted an correlation heatmap, which was shown below:

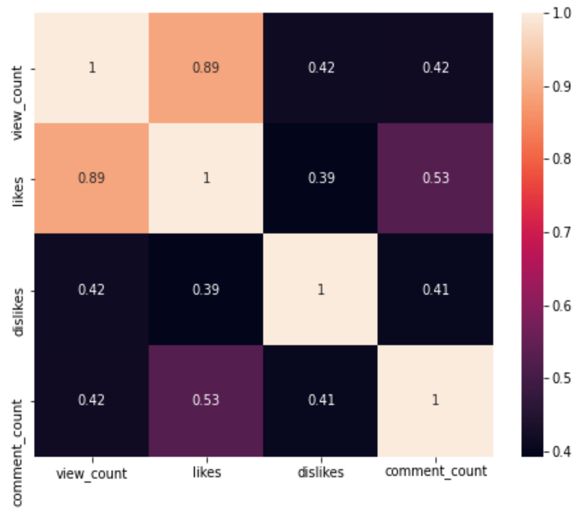

Fig 2.1. Correlation Heatmap

We can see that the like count are strongly correlated to the target variable, while dislikes and comment count have a moderate correlation with the target variable.Therefore, we believe that all of these three numerical variables need to be fed into our model.

Before verifying the second conjecture, we created a new variable called time_gap, which represents the days between the published date(publishedAt) and the scraped date(scrapedAt), so we can use that new variable to measure the relationship between times and views. We binned the time_gap into several groups and calculated the average view counts in each group. Unsurprisingly, older videos have significantly more views on usual.

Finally, we traced the category IDs back to its corresponding category names, and then created a boxplot showing the distribution of log of view_count by categories:

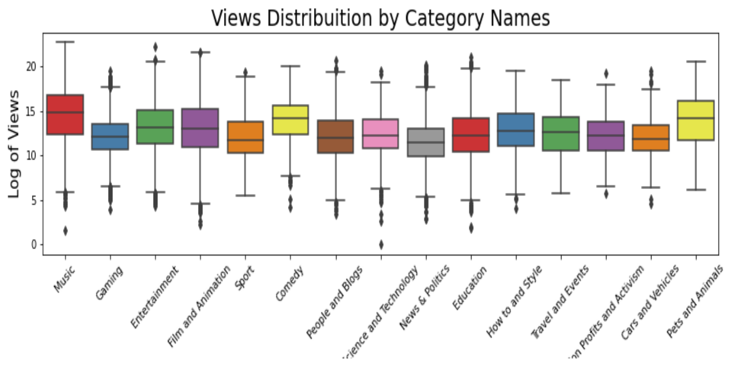

Fig 2.2. Boxplot of View vs Category

Videos in some categories like "music" and "pets and animals" clearly have more views than others, which means that categoryId is also a feature we need to take into account.

3 Data Preparation

3.1 Data Preprocessing

Besides the cleansing procedures mentioned above, we also did the some preprocessing which we considered necessary for our model to be more convenient and less noisy:

1. Normalize 'view_count', 'likes', 'dislikes' and 'comment_count' to reduce the wide ranges of these variables.

2. Convert True/False columns (e.g. 'comments_disabled') into binary variables.

3. Generate a new column called time_gap recording the days between publishing and scraping.

3.2 Define Labels

Although we already have our target variable, the log of view count, this cannot be directly used as the labels since a predictive model is not able to precisely estimate the log(view) into decimal places. What we need to do is to designate several ranges of the log(view) and define these ranges to be our labels (e.g. label video with log(view) in [0,6] as “0”), so our problems now becomes a multi-classification problem.

In order to have a sufficient and similar amount of data in each interval, we figured out to use percentiles to label the data. For example, if we want label our data into n classes, we then let the range [a,b] corresponding to the 0th to (100/n)th percentile of log(view) to be the class “0”, and class “0” can be easily converted back to the range of views, which is [e^a, e^b]. In this way, each class would have a sufficient and equal size, which eliminated the impact of uneven distribution on the performance of prediction.

3.3 Handle with Text Variables

In our dataset, we have three text variables, “title”, “tags”, and “description”. Intuitively, these three text variables are highly correlated with the popularity of a video. To incorporate them into our predictive model, we implemented some NLP techniques to convert the text into vectors. Here we used a GloVe model pre-trained on 27 billion tokens from Twitter to embed these texts. We chose the smallest embedding size to be 25 since our text is relatively short so does not need a high dimension to keep the information. To ensure that 25-d is the best fit for our problem, we also compared the models with embedding size 25 and 50 later in the training, and the 25-d has significantly shorter training time and almost identical performance as 50-d.

After embedding, each of the text variables has been converted into a 25-d vector. Considering that 25-d each is still too high, we decided to use PCA to reduce their dimensions. We set the n-component to be 0.9 so 90% of the variance would still be kept in the vectors, and now the reduced dimension for each text vector is 15.

3.4 Feature Selection

One of the most important goals for our model is to give the YouTubers a preliminary prediction on their videos’ performance so that they can adjust their video to obtain more public attention. Therefore, we decided to focus on the features that YouTubers can control, and those a YouTuber can do nothing about, like “comment_count”, will not be fed into our model, except for time_gap since a future date is treated as an input of our predictive model. To summarize, our final selection of features are as follows:

1. title(embedded), 15-d vectors
2. tags (embedded), 15-d vectors
3. description (embedded), 15-d vectors
4. whether or not enable the viewers to leave comment, binary
5. whether or not enable the viewers to rate, binary
6. number of days that have passed since the video was released, int
7. category ID, int

For future convenience, we also built a pipeline that contains all the preprocessing procedures mentioned above, so if we want to add newly scraped data, then all the pre-training preparation can be done within a single line of code.

4 Modeling and Evaluation

Now everything is ready for the modeling part. We built some model training functions to automatically split the data to 0.8 training set and 0.2 testing, and perform grid search by 5-fold cross validation on the training data given a dictionary of chosen model. At first, we used some classic classifiers such as logistic regression and decision trees as the baseline models since both models could give some decent results with the least amount of effort. After we saw some performance of baseline model, we could switch to ensemble classifiers and improve upon previous simple classifiers. Although they take much more effort to adjust and train, the result could be more desirable to fit our business goal.

4.1 Evaluation Metrics

The evaluating metric we used for hyperparameter tuning is weighted ROC AUC score because it gives a comprehensive measurement of precision and recall across all thresholds and the weighted average also avoids the possibility of imbalanced target classes. For the multiclass configuration, we compared the difference between AUC using "one vs rest" and "one vs one". One vs rest computes the AUC of each class against the rest, while one vs one computes that of all possible combinations of class. We did this analysis alongside the number of dimensions to choose during the PCA section.

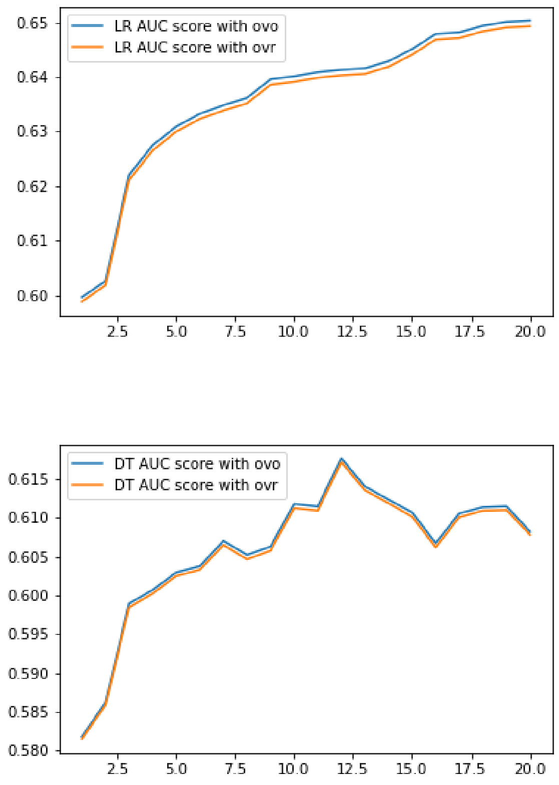

Fig 4.1. AUC Score plots with ovo and ovr for Decision Trees and Logistic Regression models

The figure shows that (Figure 4.1) two configurations have little difference when measuring the model performance. Therefore, we decided to use one vs rest for sake of time, which only has linear time complexity. This is also a crucial element if we consider the time cost of hyperparameter tuning. From the business perspective, since our purpose is to help customer to predict the videos' popularity based on video attributes, having a probability of all classes here might be more useful than classifying particular video to which range of views. This business goal makes AUC score more preferable than mean accuracy and other classification metrics according to certain threshold.

4.2 Baseline Models

4.2.1 Logistic Regression and Decision Trees

```
              precision    recall  f1-score   support

           0       0.57      0.48      0.52      1641
           1       0.39      0.24      0.30      1656
           2       0.34      0.42      0.37      1698
           3       0.28      0.30      0.29      1610
           4       0.46      0.58      0.51      1556

    accuracy                           0.40      8161
   macro avg       0.41      0.40      0.40      8161
weighted avg       0.41      0.40      0.40      8161

AUC score on test data: 0.7195691659138763
AUC score on train data: 0.7342554529171799
```

Fig 4.2. Summary for Decision Tree Model

```
              precision    recall  f1-score   support

           0       0.29      0.45      0.35      1641
           1       0.27      0.18      0.22      1656
           2       0.32      0.25      0.28      1698
           3       0.27      0.15      0.20      1610
           4       0.39      0.58      0.47      1556

    accuracy                           0.32      8161
   macro avg       0.31      0.32      0.30      8161
weighted avg       0.31      0.32      0.30      8161

AUC score on test data: 0.6440684112927532
AUC score on train data: 0.6487220373128595
```

Fig 4.3. Summary for Logistic Regression Model

We did a simple grid search to tune the hyperparameters of both logistic regression and decision trees. For logistic regression, we only considered the type of regularization and its strength, and for decision trees, we considered maximum depth, minimum samples split, minimum samples leaf and splitting criterion.

4.2.2 Other Models

AUC Score for Models on embedded features with 15 dimension	
Model Name	AUC Scores
Random Forest	0.79278
Logistic Regression	0.64407
Decision Tree	0.61012
Support Vector Machine	0.72272
AdaBoost	0.72508
XGBoost	0.78984
Gradient Boosting	0.77038

Table 4.1. AUC Scores Comparison for 7 Models as Baselines

Before we delved into hyperparameter tuning, we also constructed a baseline for all possible candidates. The table above (Table 4.1) shows that Random Forest, Gradient Boosting and XGBoost are the best three candidates. We used to consider support vector machine from its decent performance, but the running time of SVM is extremely slow given its time complexity of training is up to O($number\,of\,features * number\,of\,samples^3$). If we plan to add more data later on, the time cost is not worth compared to those tree-based classifiers with logarithmic training time.

4.3 Hyperparameter Tuning

For the hyperparameter tuning part, we designed a two stage tuning. In the first stage, we used a randomized search across the parameter grid with a maximum of one hundred iterations because the parameter grid we set up has over one thousand combinations of parameters, which may take tens of hours to tune a single classifier and we thought the subtle improvement is not comparable to the time taken. In the second stage, we manually choose the parameters that could be improved further in the first stage and update the parameter grid with much fewer iterations. In other words, the first stage helped us to narrow down the range of the hyperparameters. Then we performed a thorough grid search on the second hyperparameter grid and that may give us a combination of hyperparameters with better results. In addition, to further optimize the tuning time, we made use of joblib-spark that could distribute the tuning task on a Spark cluster. The results show that the two stage procedure indeed improves each classifier by 0.005 on the test AUC score.

Ensembled Model Performance	
Model Name	AUC Scores
Random Forest	0.80609
XGBoost	0.79963
Gradient Boosting	0.80105
Stacking Classifier	0.81346

Table 4.2. AUC Scores Comparison for Ensembled Models

Above is the classification report of 4 classifiers on our final 40k dataset (Table 4.2). Except for the regular Random Forest, XGBoost and Gradient Boosting models, we tried to combine the output of all these three classifiers into a stacking classifier, which feeds the output of three classifiers into a logistic regression, and improved the test score by another 0.01 as shown in Table 4.2. This stacking classifier wraps up our modeling part.

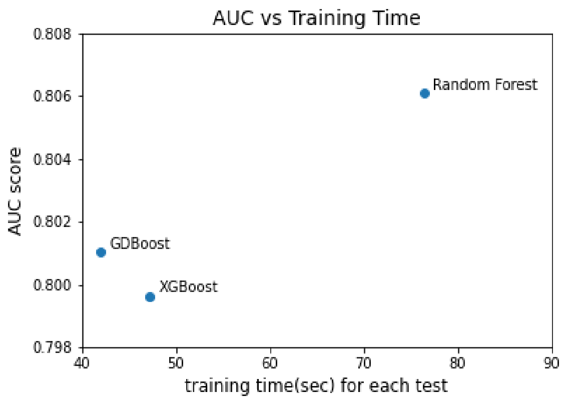

Fig 4.3. Performance vs Computational Cost

5 Deployment

5.1 Model Deployment

After model selection, we decided to move on with the model Stacking Classifier with a reasonable time and highest AUC score. Our model did a great job in the classification of YouTube trending videos' popularity. Therefore, we built a function based on our best model that takes in a video's information and a future date and will output a predicted views of the video on that date. As an example, we randomly select a video from our test set, which is a videotape on a basketball 1 on 1. It was published by YouTuber Awakened Choppa on 11/26/2020, and has a 77k views on the date we scraped it, which was 11/27/2020. Our model predicted it to have a view count between [586131, 1362537] on 12/30/2020.

When turning into the real world problem, both YouTubers and YouTube itself can be benefited from it. Our model can be added as a utility function into YouTubers' YouTube webpage /app interface. Thus YouTubers can use it to predict their videos future popularity. Besides that, YouTubers may also use it to modify their video performances or styles to capture the real-time hotspots –no matter if it is the video categories, tags, or titles etc.–in order to understand their audiences better and gain more views. YouTube itself, as the video playing platform, would be able to gain more traffic and streams.

Furthermore, our model's popularity label classification could also be used for companies who use YouTube channels as a propaganda method to promote their merchandise. Advisers can use the estimated label and the most likely future views to access the potential business value for a specific YouTuber or YouTube video to optimize their advertising budgets and product profits. AB testing in this case can be used to determine our model performance in the YouTuber's interface. And it is kind of simple to evaluate and compare. The change in the amount of traffic and streams through YouTube platform can be the feedback for further improving our model so that we could provide YouTubers better video making advice.

5.2 Considerations

Since the YouTube videos' popularity always changes fast, our model needs to update periodically by updating the training set through YouTube data scraping, for the sake of capturing any popularity concept drifts, to make sure the classification labels are correctly generated. Moreover, even thought it is mitigated, the selection bias may still exist when scraping the data based on the created keywords.

5.3 Ethical Risks

Because our function can help YouTubers to modify their current video performance to gain more views and revenues an overuse of this function should be aware and considered. Our function is built based on the video topics and tags that have already exist. Therefore, since all the YouTubers want to catch the future popularity and make money, function overuse may stifle YouTubers' creativity and even increase the probability of video content plagiarism, thus causing the vicious competition in the video market.

6 Conclusion and Future Work

6.1 Conclusion

YouTube trending video popularity prediction is a topic with huge potential business values. In this project, we transfer such topics into a supervised multiclass classification problem. Different classification machine learning algorithms such as Logistic Regression, Decision Trees, Random Forest, Support Vector Machine (SVM), XGBoost, AdaBoost, and Gradient Boosting were tried on the dataset we created by scraping data from YouTube about one week. The stacking classifier we created by combining the outputs from Random Forest, XGBoost and Gradient Boosting classifiers perform the best with reasonable running time and the highest AUC score.

6.2 Future Work

As for the future work, first of all, more data is recommended to be scrapped from the YouTube platform and be put for training the model. Secondly, in order to predict the popularity more precisely, when scrapping the data, the keywords of categories that generalize topics can be more specific. Moreover, besides Random Forest, XGBoost and Gradient Boosting classifiers, other classifiers could also be considered adding into the final stacking classifier as part of the components to make the final AUC higher. Last but not least, a more efficient algorithm could be developed based on our current model to perfect our YouTube trending video popularity label prediction model.

References

[1] Matthias Funk. How many youtube channels are there? *YouTube Marketing*, 2020. https://www.tubics.com/blog/number-of-youtube-channels/.

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. 2014. https://nlp.stanford.edu/projects/glove/.

[3] Mitchell J. Trending youtube video statistics. 2018. https://www.kaggle.com/datasnaek/youtube-new.

[4] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *eprint arXiv:1603.02754*, 2016. https://arxiv.org/pdf/1603.02754.pdf.

[5] Gábor Szabó and Bernardo A. Huberman. Predicting the popularity of online content. *Communications of the ACM*, 2010. https://dl.acm.org/doi/10.1145/1787234.1787254.

7 Appendix

7.1 Collaboration

- Zheyuan Hu:
  - Construct the proposal
  - Business understanding
  - EDA
  - Data preprocessing
- Zixuan Shao
  - Data scraping
  - Build and train models
  - Related content in the report
- Yang Gao:
  - Model Analysis and Interpretation
  - Model Deployment
  - Final write-up Formating

7.2 Links

All of the code used in our project can be found in the Github.

- Scraper Github repo
- Codes