Youtube Trending
CS4563
Final Project

Sofiia Barysheva &&
Sam Zhang

Table of contents

Introduction

Through this project we wanted to learn how Youtube's algorithm determines which videos go on trending and when they do. To achieve our goals we used this dataset: https://www.kaggle.com/datasnaek/youtube-new#MXvideos.csv . It is a detailed collection of features such as likes, dislikes, comments, trend dates, etc., which can be used to predict certain trends. The dataset consists of many subsets categorized by country. In our project, we will mainly focus on the data from the US region.

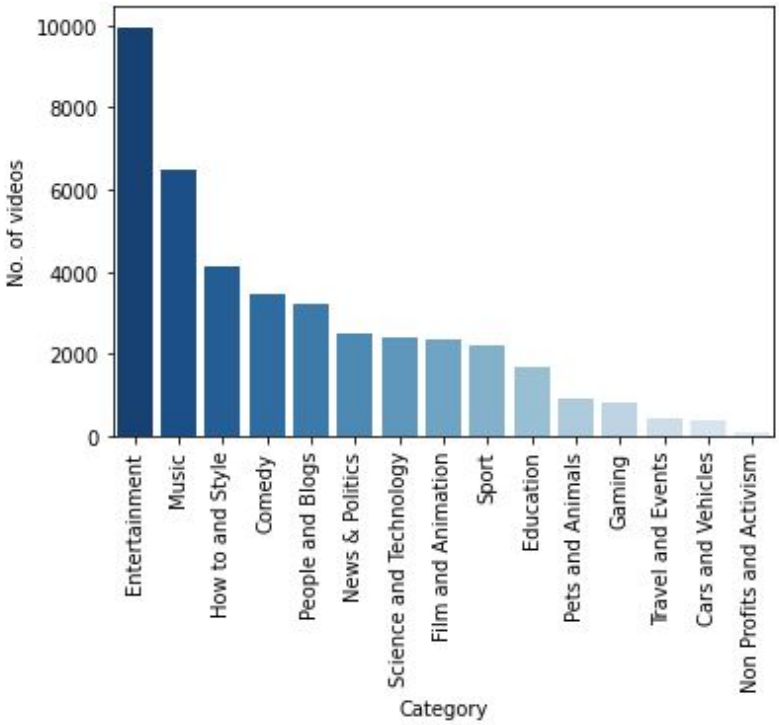
We mainly focused on the following problems: predict the number of likes on a trending video given the numerical features from the dataset, predict the category of the video using both numerical features as well as words from titles, tags, and video description. View our code on github using: https://github.com/Samalam98/introml-project

Data Visualization

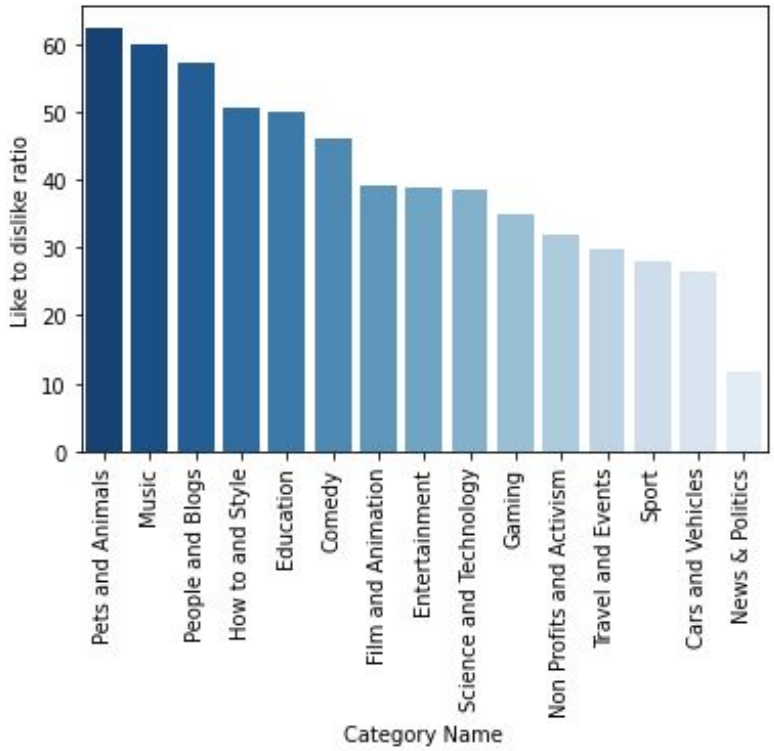
One of the most important things of Machine Learning is the ability to visualize data trends to get a sense of characteristics that might be useful exploring into. We had 16 columns from the raw dataset: video_id, trending_date, title, channel_title, categori_id, publish_time, tags, views, likes, dislikes, comment_count, thumbnail_link, comments_disabled, ratings_disabled, video_error_or_removed, and description,. The first 5 rows are shown in the screenshot below.

	video_id	trending_date	title	channel_title	category_id	publish_time	tags	views	likes	dislikes	comment_count	thumbnail_link	comments_disabled	ratings_disabled	video_error_or_removed	description
0	2kyS6S...	17.14.11	WE WAN...	CaseyN...	22	2017-1...	SHANte...	748374	57527	2966	15954	https:...	False	False	False	SHANTE...
1	1ZAPwf...	17.14.11	The Tr...	LastWe...	24	2017-1...	last w...	2418783	97185	6146	12703	https:...	False	False	False	One ye...
2	5qpjK5...	17.14.11	Racist...	Rudy M...	23	2017-1...	racist...	3191434	146033	5339	8181	https:...	False	False	False	WATCH ...
3	puqaWr...	17.14.11	Nickel...	Good M...	24	2017-1...	rhett ...	343168	10172	666	2146	https:...	False	False	False	Today ...
4	d380me...	17.14.11	I Dare...	nigahiga	24	2017-1...	ryan|"...	2095731	132235	1989	17518	https:...	False	False	False	I know...

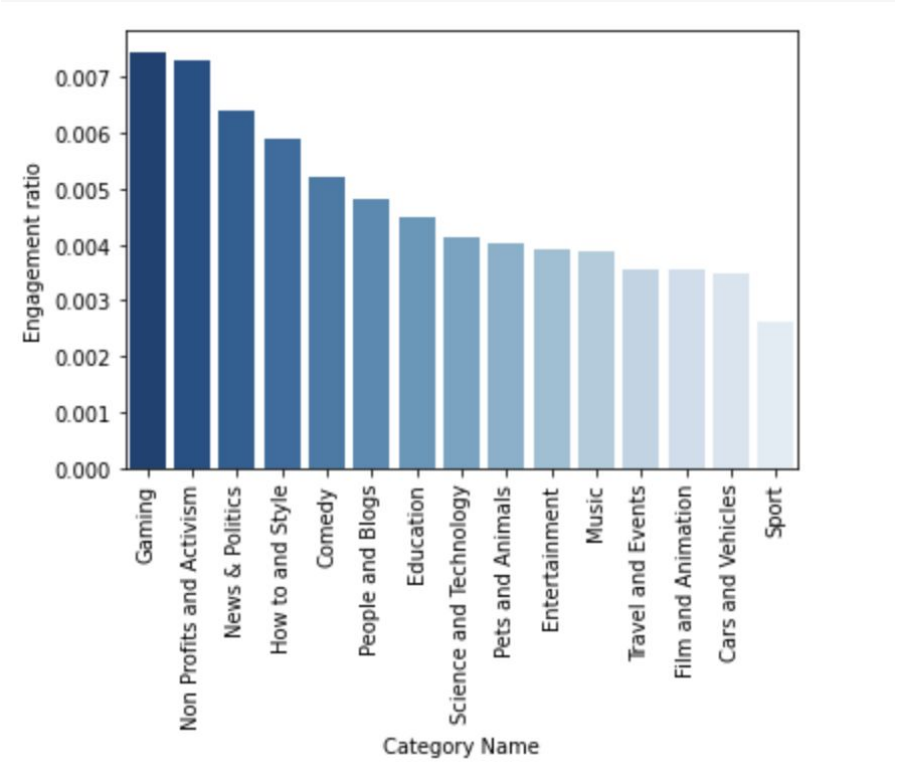
So here, we started to plot some graphs and organized the data mainly by categories. There are 16 total categories in total, such as Entertainment, music, comedy, etc. The graph belows shows the number of videos in each category.



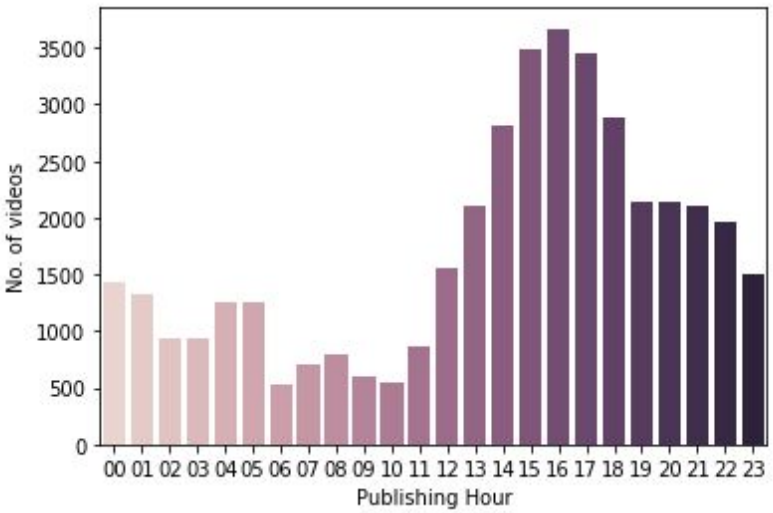
Out of the 40,000 plus videos in the data set, the top three categories contains over half, with Entertainment having the most. By taking the fraction of no. of likes over no. of dislikes, we add an approval ratio to the dataset.



It is clear that more people liked Pets and Animals, while News & Politics tend to get the more dislikes. By calculating the ratio of comment counts and no. of views, we can see how well the audience is engaging with the videos by checking how much they comment if they watched the video. This might not be an accurate reflection of people's approval of a subject, as heated discussion usually causes more comments.



By observing the engagement rate, people tend to comment more on categories like gaming, NonProfit and Activism and News & Politics. This is well expected as these are somewhat more controversial than the others.

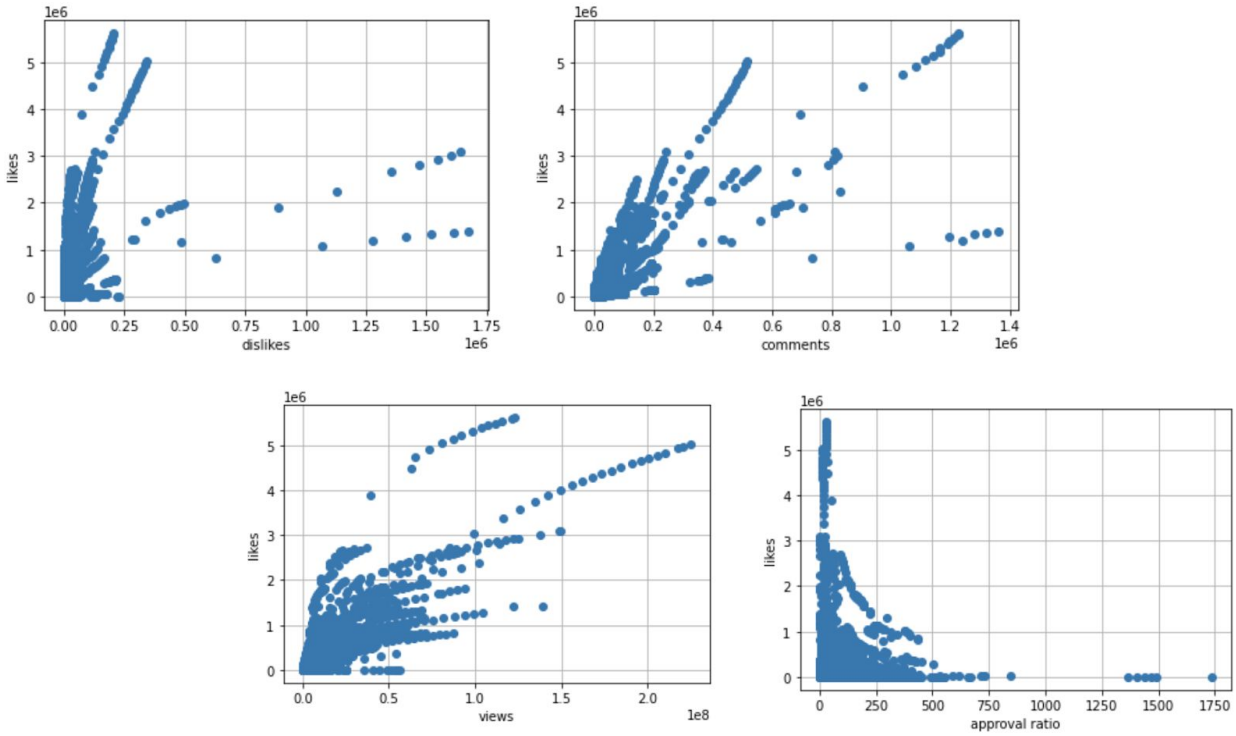


By looking at the publishing hours, we can conclude that trending videos get published the most around 3pm to 5 pm.

To get started with the project, data preprocessing was important to filter out some unusable rows from the dataset, such as when the comment_count column is needed, but a certain video had comments disabled. Also, when deriving some columns we used division. It was necessary to replace values with positive/negative infinity to NaN and drop the rows that way.

Likes Count Prediction

One of the goals was to be able to predict the number of likes a video will get based on other fields. By drawing some scatter plots as shown below, we can see that no. of likes increases as dislikes, comment counts and no. of views increase, while the approval ratio seems to be inversely proportional to likes. So here we chose likes as the target, and the above mentioned features as our variables as we sought to find a correlation.



Simple Linear Regression

A straightforward approach was to try linear regression on the features selected. We started off by doing single linear regression on these 4 columns. Here are the results for each individual loss as well as the best R^2 value from

```
0 loss=5.78e+14 beta0=27591.933366179866 beta1=5.708298055294201
1 loss=1.56e+15 beta0=57309.19648984731 beta1=5.095892649578032
2 loss=5.77e+14 beta0=12359.514775295182 beta1=0.026509719336592474
3 loss=2.11e+15 beta0=60504.10699891664 beta1=336.22160265605356
```

```
syy = np.mean((y-ym)**2)
rsqr = 1 - np.min(losses)/(nsamp*syy)
print("Rsqr="+"{:.3}".format(rsqr))
```

```
Rsqr=0.728
```

Multivariate Regression

As we can see these losses are very large, and the highest R^2 value is about 0.73, which has room to improve. Therefore we combined these columns to train a multivariate regression model on our target.

```
multiple variable loss=2.15e+14
R^2 value =0.898829415627762
```

This model yielded much better results as the loss has increased by at least 60%, and the R^2 value increased to about 0.90. We wanted to push the results even further by using feature transformation on our columns. We attempted to add columns by taking the square root of views, square of dislikes, etc.

```
multiple variable loss after feature transformation =1.98e+14
R^2 value =0.9066794505335459
```

We can clearly see that the loss has dropped even further, and we successfully increased the R^2 value to 0.91. There are definitely better ways to do feature transformation on the columns which would take more time and trials to compute. Overall, this shows that these fields combined are correlated to the target, and can be used to predict the no. of likes of a certain trending video.

Future & Improvements

Algorithms to try:

- Random forest regressor
- Decision tree
- Logistic Regression with regularization

Category prediction

We wanted to explore the possibility of predicting video categories using words available in our data set. First, to illustrate our point we made word clouds for several categories to show the most used words in each category. Keep in mind, this is 2017, so the words are pretty odd.

You can easily tell from these word clouds which categories they belong to. So, we thought we could try to teach a machine to do that for us.

K-Nearest-Neighbors

Here we used K-Nearest-Neighbors from the sklearn library to try to classify categories of videos.

First, a train-test split was made to split the dataset into 80:20. When we first tried to do a one-vs-all using all the categories without any cross-validation or parameter tuning, the accuracy achieved was horribly low, at 0.26. This can be accounted for as there are 16 categories, and many of them share very similar traits in their views, likes, comment counts, making it more difficult to predict.

```
#check accuracy of our model on the test data
knn.score(X_test, y_test)
```

```
0.2602687626774848
```

Therefore, it made more sense to us to only focus on the top categories first to predict the labels. We took the sub dataframe with only videos in categories Entertainment and Music. We also ran Cross Validation on the result, and took the average of the 5 scores achieved. This gave us a 0.67 score as accuracy.

```
0.6659354434346599
[0.63365716 0.66593544 0.6750235  0.6884989  0.66750235]
cv_scores mean:0.6661234722657474
```

The default number of neighbors used was 5. This can be optimized to further increase the score. Therefore, we ran GridSearchCV to search for the best number of neighbors. As shown below, the best n_neighbors was 19 and we got a better accuracy of 0.69.

```
best params: {'n_neighbors': 19}
best score: 0.6881228455029771
```

We then tried the same algorithm for the top 3 categories, this time including How to and Style. This decreased the score to about 0.56. To summarize, numerical features are possibly not the optimal way to predict a category. Therefore, other methods are used below.

```
best params: {'n_neighbors': 23}
best score: 0.566443553308559
```

Bag-of-words

First, we chose bag-of-words as a model for dealing with words in the titles/tags/description from our dataset. Here for the train/test split we permuted our dataset and assigned the first quarter to train and the second quarter to test.

In terms of prediction our first goal was to determine if we can predict a single category versus all the rest of the categories. The challenge of this problem is that each category takes about 15 to 5% of the entire data. So, it was really easy to fall into the trap of having 95% accuracy for a category that is only 5% of the entire data. This happened while using Bayes rule as it gave great accuracy, but predicted all 0s.

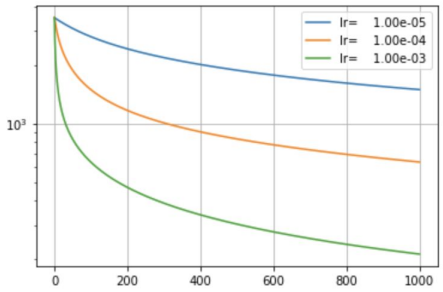
Next, we tried using gradient and using logistic loss for category predictions. Using simple gradient optimizer we got pretty good results on the training data:

```
Train accuracy = 0.952645
Recall: 0.7463672391017173
Precision: 0.923202614379085
```

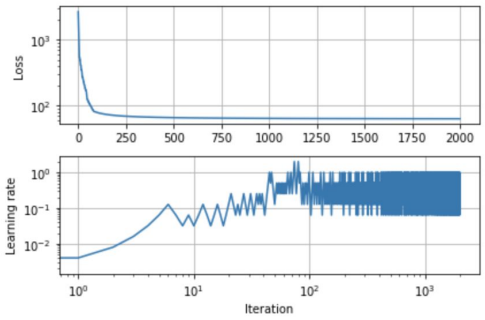
Then, we tried using different learning rates in the optimizer with the best one being (here you can see a big increase in recall and a slight increase in both accuracy and precision):

```
lr=    1.00e-03  Train accuracy = 0.994254
Recall: 0.9749009247027741
Precision: 0.9866310160427807
```

With the overall graph of learning rates like this.

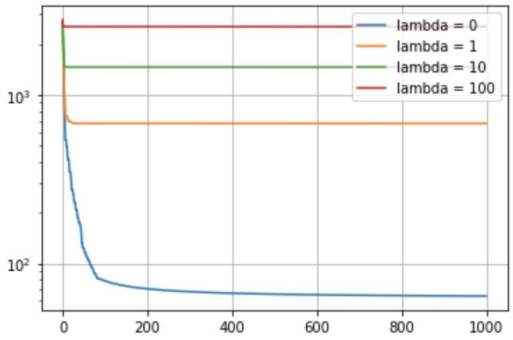


Next, we played around with the step size:




```
Train accuracy = 0.997622
Recall: 0.9841479524438573
Precision: 1.0
```

Then, we used L2 regularization that proved to do more harm than good with the best results at lambda equal to zero.




```
lambda = 0  Train accuracy = 0.997622
Recall: 0.9841479524438573
Precision: 1.0

lambda = 1  Train accuracy = 0.991084
Recall: 0.9630118890356671
Precision: 0.9772117962466488

lambda = 10  Train accuracy = 0.971864
Recall: 0.8665785997357992
Precision: 0.9411764705882353

lambda = 100  Test accuracy = 0.962758
Recall: 0.7784431137724551
Precision: 0.9285714285714286

lambda = 100  Train accuracy = 0.946503
Recall: 0.7014531043593131
Precision: 0.9234782608695652
```

Then we tried to apply kernel logistic regression using linear kernel, and that took a really long time to calculate given the size of the training data - 10000 entries with each entry of 4344 length. It never finished (on my watch), so I gave up on the idea (that is I tried both scipy and without scipy - it still took a very long time and never finished).

We also used sklearn for linear model of logistic regression and got:

```
Logistic Regression Accuaracy = 0.995246
Recall: 0.9640718562874252
Precision: 1.0
```

Then, we used a logistic regression classifier on a non-linear kernel, and it gave us:

```
Test accuracy = 0.732171
Recall: 1.0
Precision: 0.3306930693069307
```

Kernel Support Vector Machine gave us:

```
Test accuracy = 0.998415
Recall: 0.9880239520958084
Precision: 1.0
```

This became the best test result that we've gotten for one vs all on the category 10.

For the **actual category prediction** we used gradient with logistic loss that worked well on the one vs all and ran faster than the other methods. The speed was the basis of its appeal since we would need to calculate 16 betas and that takes a while. Seriously, the calculation of 16 betas took the entire Booksmart movie and even around 10 minutes more. The results couldn't have been more disappointing with the train accuracy of 0.479394 and test accuracy of 0.012779. Although, for top 3 the stats went a little up, but still were very mediocre:

```
Train accuracy = 0.978007  Test accuracy = 0.339937
```

For top 5 (although that would be a third of the categories):

```
Train accuracy = 0.987121  Test accuracy = 0.504754
```

Word Embedding

Another method I used for category prediction was **word embedding** using Keras Python library. For that I used 10000 entries of the data and a K-fold testing to ensure the model trained was a good fit. I tried to create my own word embedding as well as used GloVe (https://nlp.stanford.edu/projects/glove/) pre-trained word vectors. I used a baseline neural network with no hidden layers and an embedding layer from Keras Python library.

This approach gave me:

```
Baseline: 95.22% (0.21%)
```

Which is the mean and standard deviation of the model accuracy on the dataset. Result!

Future & Improvements

Algorithms to try:

- More variations of word embedding.
- K-Nearest Neighbors algorithm with word-embedding.
- Mixed word-embedding and bag of words.
- Neural networks using hidden layers.

Conclusion

We managed to somewhat successfully predict likes, views and categories using several machine learning techniques. Not without setbacks, tags proved to give a pretty good description of the video category if used in a word-embedded baseline neural network model. Linear regression showed that it is possible to predict the amount of likes a trending video received using numerical features and feature transformations. KNN algorithm gave us an understanding that not only words can predict the category of a certain video, but views, likes, and other numerical features can accomplish that as well.

The data however did not allow us to do exactly what we wanted. One of the challenges we met was that although we sought to predict how fast videos would trend, the dataset itself did not include the necessary information. The column trending_date did not include the time it

started to trend but only the date, while publish_time had the time included. This made it impossible to calculate the exact difference between the publish time and the trending time. Most videos ended up trending within 0 days when the calculation was performed. Therefore, we had to give up this initial goal we had to predict how fast a video would trend. One thing that is related but we did not have time to explore was how long (how many days) a video stays on trending. This could guide us to better understand how characteristics determine whether a video is just temporarily popular, or it is gaining continuous attention.

Initially we wanted to determine how videos go from normal videos to trending. Since the data that we found had only trending videos we had nothing to compare them with. In addition, it would be good to know other features of the videos like, for example, the amount of subscribers of the creator as well as the average number of views received by that creator for the past month/year, the total number of views that channel received before the video went viral. So, in the future, it would be useful to get that data and get more accurate and interesting predictions. In the future, we would also be interested to run the same models on different regions to see how the conclusions compare horizontally. This would potentially give us insights on whether people from different regions prefer different types of content.