YouTube Trending Category Classification

Kyle Thao  
 Computer Science  
 California University State Sacramento   
 Sacramento State United States  
 Kylenthao@csus.edu

ABSTRACT

Everyone these days are trying to become “YouTube Famous” to make a lot of easy money. In recent years, people seem to be frustrated about how YouTube handles what becomes the trending front page and what gets recommended to viewers. Some claim that the “algorithm” is a bit biased, which causes them to lose revenue and publicity. One way to tackle this problem is by classifying and identifying what types of videos become trending. By determining what types of videos become trending, based on its description and tags, we can hope to understand what type of videos are more likely to trend. This could hopefully help others reach a broader audience to influence and grow.

My approach to this problem had me using two different machine-learning models, a 1D Convolution Neural Network, and a Fully Connected Neural Network. To handle my data, I used TFI-DF vectorization to encode all of my text values to feed into as my x values. Using the categories names provided by the dataset, I was able to encode the category ids from each video as well to feed in my models as my y values.

In the end, my model was able to predict at an accuracy of 40% for my CNN model, which underperformed my baseline approach. However, I was able to achieve an accuracy of about 71% for my fully connected model, which outperformed my baseline approach. These results were achieved by tweaking the parameters of my TFIDF vectorizer

1 Introduction

Content creators are always looking for a broader audience to boost their views and revenue. By hitting the trending page, these videos are seen by the larger part of YouTube from essentially being on the front page. However, knowing what classifies a video as trending and knowing what type of videos that frequently trend is ambiguous.

To approach this problem, I decided to classify the category or genre of trending videos based on the tags and descriptions of each video. To handle the text data, I implemented the use of the Text Frequency - Inverse Document Frequency, TF-IDF. This was then classified on the different categories as mentioned in the dataset. When partitioning data however, parameters were set to not include words whose frequencies were over a certain threshold, which helped avoid words that were less meaningful and showed up frequently in the dataset. Two different models were used to test and train the data, Convolutional Neural

Networks and Fully Connected Neural Networks. Then I would be able to compare the results between the two.

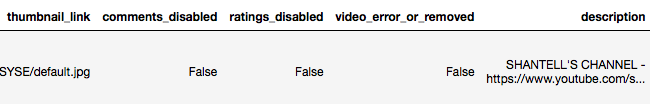
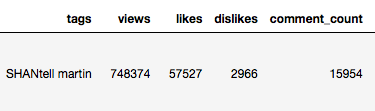
By attempting to solve this issue, I was able to contribute by:

* Classifying different categories of videos
* Predicting the category of each video based on descriptions and tags
* Determining what type of videos frequent the trending page
* Trying to determine what words affect this classification

For the rest of this paper, I will be going over several of topics of my project, starting with my problem formulation, system and algorithm design, experimental evaluations of my problem and outcome, and lastly other work related to mine and how they differ.

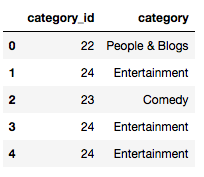
2  Problem Formulation

For my problem, I used a dataset of trending videos from Kaggle created by Mitchell **[1]**. This dataset contained csv files of trending videos sorted out by different countries such as the United States, Japan, and Germany. It also included Json files containing the names for each category and other metrics for that particular country/region.



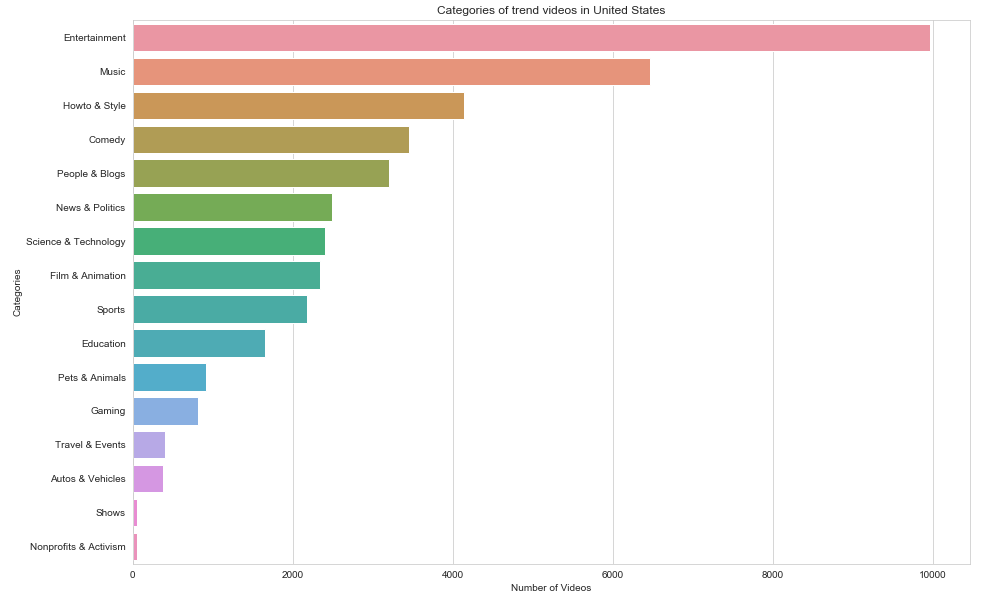
**Figure 1: One entry of my data set, visualizing the different attributes**

Each video contained these different attributes as shown above. Most of this data were not meaningful to my project such as the thumbnail link, and the disabling of comments or ratings. Because of that, I chose to only pull out and encode the necessary attributes for my project, which including the title, tags, description, and category ids. To prep my y data, I utilized the Json file containing the different category names and the id that each category name respectively belonged to.



**Figure 2: The first 5 entries of ids and their respective categories**

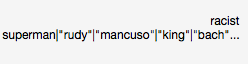
I was then able to visualize my data in a bar graph to see what categories were more frequent in my dataset.



**Figure 3: Chart showcasing the amount of trending US videos that each category held**

From **Figure 3**, we can see that entertainment, music, and how-to videos made up the majority of the trending video categories. Now with the data aligned according to their labels, I was able to one-hot encode my data to fit the y data that would be fed into my models.

For my x data, I took the description, title, and tag columns from the original dataset. The text data was full of undesirable formatting, unwanted punctuations, and unnecessary capitalization that could affect the normalized output.



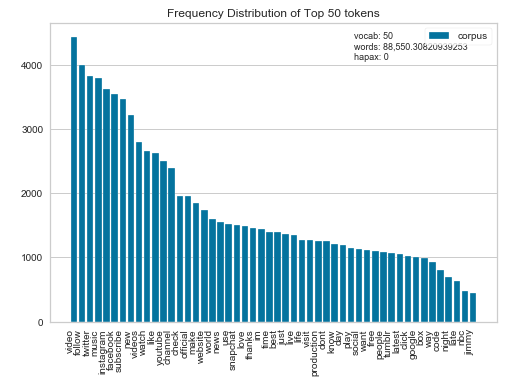
**Figure 4: An example how the tag looked like before normalization**

Shown above in **Figure 4**, is an example of a video’s tags before formatting. You can see that characters such as “|” littered the entry. This would be removed by the usage of Python’s regular expression parser, which formatted the data as shown below.Macintosh HD:Users:kyle:Desktop:Screen Shot 2019-11-24 at 4.42.48 PM.png

**Figure 5: Tag mentioned in Figure 4 after normalization**

Along with filter out unwanted characters, I decided that words that consisted of links or websites were to be removed, as they were arbitrary values that skewed the results. These links usually included links back to the poster’s own channel, social media pages, or accounts for donations.

This allowed for the TF-IDF vectorizer to identify words more easily, creating more accurate data. After normalizing the text data, I conjoined all of the results into a set to be able to input it into the TF-IDF. The results of the vectorizer were in the form of a sparse array denoting the TF-IDF values for feature words.



**Figure 6: The top 50 frequent words in the dataset**

After running through the TF-IDF vectorizer, I was able to get the most reoccurring words throughout the dataset. In **Figure 6**, the top 50 reoccurring words were visually shown, with words such as videos, follow, and twitter being the most occurring.

By encoding the categories, categorically, and using the process of TF-IDF, the data was primed for a classification problem.

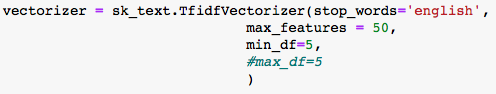
3  Project/Algorithm Design

3.1 Project Architecture

For my project, I designed it to be structured in way where I could visualize some of my data first and then encode and train my data afterwards. I mainly focused on seeing what attributes laid in my dataset and some charts to help visualize how some of these data entries compared against each other. Then I began to encode the data to fit my models. I started by encoding my categorical data, and then finished off by mutating and vectorizing my textual data. Lastly, I trained the data on a Fully-Connected Neural Network and a Convolution Neural Network, to receive two different confusion matrices that displayed precision, accuracy and f1-scores.

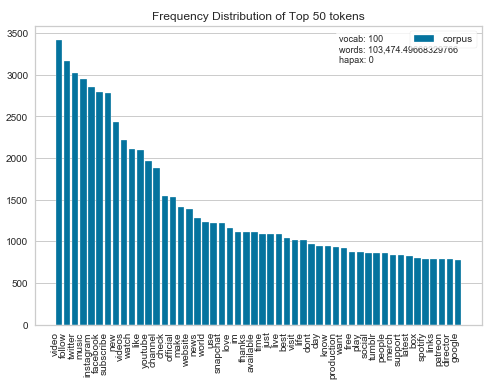
**3.2 Vectorizer Parameters**

When encoding my x data using the TF-IDF vectorizer, there were some parameters that affect the outcome of my models.



**Figure 7: The TF-IDF vectorizer and its parameters**

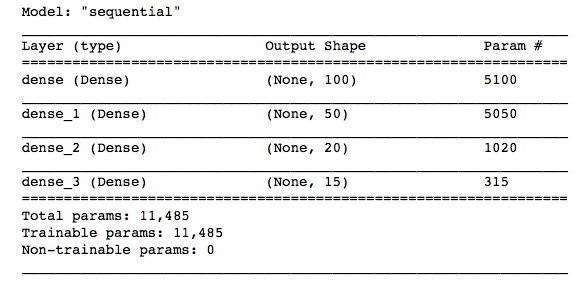
If you take a look at **Figure 7** above, you see that I have set the max features to 50 and the min to 5.



**Figure 8: Top 50 feature words using the vectorizer with the max parameters of 100**

An earlier trial had the max set to 100 features, which made my fully connected model’s accuracy lower, but my CNN model’s accuracy a lot higher. By looking at **Figure 8** compared to **Figure 6**, the different words near the tail end seem to different drastically, almost replacing the tail of  **Figure 6** with a different set of words that would skew my predictions.

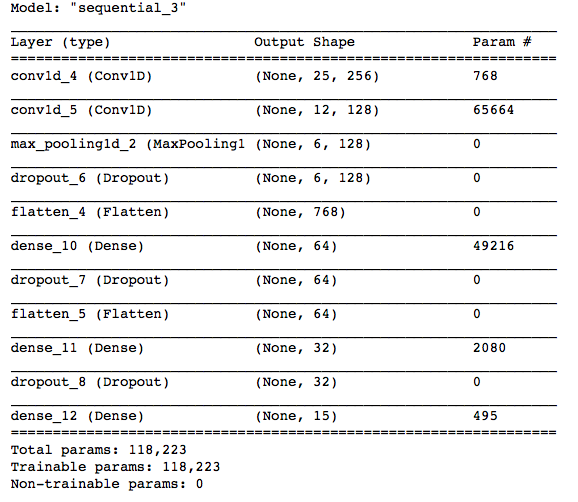
**3.3 Model 1: Fully Connected Neural Network**



**Figure 9: The Fully Connected Neural Network**

Looking at **Figure 8,** you can see that my first model contains four different dense layers that had output shapes that gradually went lower and lower. During each layer, I also added in Relu activation. When compiling my model, I set the loss to categorical crossentrophy, and my optimizer to adam. My total parameters become to be over 11 thousand.

**3.4 Model 2: Convolutional Neural Network**



**Figure 10: The Convolutional Neural Network model (CNN) and its layers**

Looking at my second model from **Figure 9,** I included two one-dimensional convolutional layers, one max pooling layer, and two sequences of dropout, dense, and flatten layers. In both my convolutional layers and my dense layers, I added Relu as my activation with the last dense containing a soft max to create the correct output shape. For this model, I once again use categorical crossentrophy as my loss and adam as my optimizer.

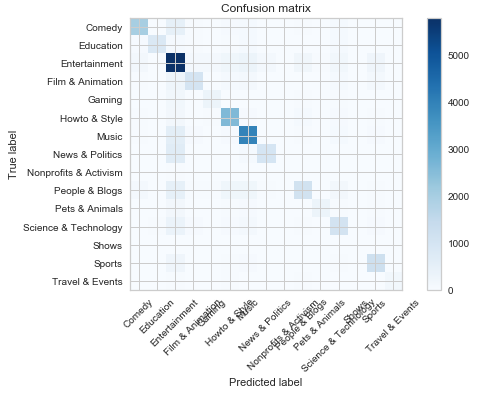
4.  Experimental Evaluation

**4.1 Methodology**

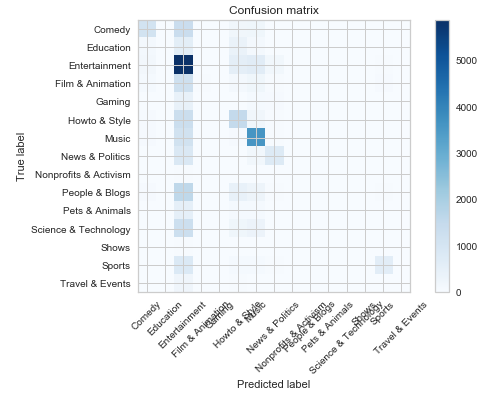
Referring back to **Figure 1**, my dataset has a lot of attributes that are unwanted. I mainly focused on data that was necessary to classify my data, which included all of the textual data, and the categorical data. The text data was fed through a TF-IDF vectorizer and then trained on the categorical encoded, category data. From there I was able to split the data collected into testing and training with training size being a quarter of the data.

To compare different results, I used some visual aids to help. Most notably, I kept track of how long it took for the models to train the data set, and used confusion matrices to help illustrate the accuracy of each model. If you take a look back at **Figure 6**, that bar chart helped visualized the more frequent words in the dataset. When I changed the vectorizer parameters, this subsequently changed the list of feature words, which **Figure 6**, helped me identify and compare.

**4.2 Results**

These metrics stated in **4.1,** helped me compare my overall f1-score, accuracy, precision, and feature words among different fine tunings. 

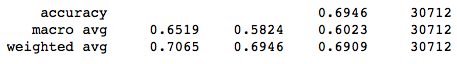
**Figure 11: Confusion matrix detailing the accuracy of the Full Connected Model**

****

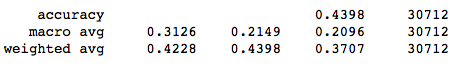
**Figure 12: Confusion Matrix for the CNN**

Looking at both **Figure 10** and **Figure 11**, we can see that the fully connected model had a better accuracy than the CNN. A majority of the classification was entertainment as seen in the raw data, and it is reflected here in the confusion matrix as well. The CNN model seems to have trouble identifying videos that are supposed to be classified as entertainment however, the Fully Connected model seemed to have the same issues but with fewer incorrect classifications.

If we take a look at the accuracy and precision gained from the metrics, **Figure 12** and **Figure 13**, we can conclude that the accuracy was way higher for the Fully Connected model compared to the CNN model.



**Figure 12: Accuracy and precision of the Fully Connected model**

****

**Figure 13: Accuracy and precision of the CNN model**

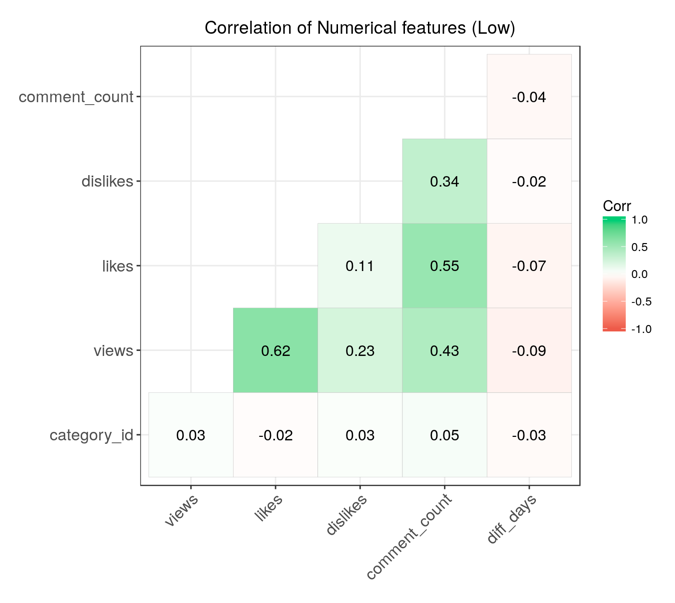
The accuracy for the Fully Connected model is 70% while the CNN model sits at almost half the accuracy at 43%.

The loss for the fully connected model, 1.0265, was about .7 points lower than the CNN model’s loss of 1.7955. This also supports that the Fully Connected model performed better compared to the CNN model.

5.  Related Work

**5.1 Related work and its problem and method**

Another person’s related work was found on Kaggle with a different approach to this problem. Mohit, the author of this publication, who also wanted to classify the different US trending video categories, used a different approach from mine [2]. Instead of classifying based on text data, he classified based on numerical data.



**Figure 14: Mohit’s correlation chart between numerical data and the category ids [2]**

Mohit was able to find a correlation between some metrics such as likes, dislikes, comment counts, and views to help classify and predict. Along with that, he was able to train his data on models not yet well known to me to gain different results. The models he decided to use in his project included:

* kNN
* kNN with decision trees
* Decision trees
* Tuned decision trees
* Random forest

**5.2 Differences between the related work and my project**

Between my project and the related work, we essentially have the same problem. However, Mohit’s approach seems to be vastly different, as he uses different models and different types of data to train on. While I use a TF-IDF vectorizer to gather feature words, Mohit was able to take the numerical values and find a correlation with the category ids.

In the end of his publication, Mohit states the accuracies of the models. The accuracies seem to hover around 40-45%, which does match up with the accuracy of my CNN model. However, my Fully Connected Neural Network seems to have a higher accuracy than his at 69%. My project seems to be slightly better in some areas however; Mohit uses different models and data to classify his problem, which makes his results more varied and analyzable.

6.  Conclusion

Overall, after analyzing my data and tuning parameters, I have come to the conclusion that my Fully Connected Neural Network performs a lot better than my CNN. By changing some of the ways I added layers and parameters, I was able to get vastly greater results with my Fully Connected model, however it tanked for my CNN model the further I went. This could possibly be the factor of becoming more accurate when the model was not classifying correctly in the first place. If I was able to analyze this longer and possibly come up with other ways to tune my model and its parameters, I possibly would be able to create a more accurate CNN model. Another improvement to this project would be to use more than region of data to compare how they differ to the US.

After looking at some related works, I feel like I could also have tested some metrics based on numerical data as well. Alongside that, as Mohit has done, more than two different models should’ve been used to demonstrate different techniques and outcomes, which could affect the way the results are interpreted. It does seem that I either do better or about the same accuracy as his project however.

7.  Work division

This project was done solo, with me handling all the work. I processed, analyzed, and trained the data on my own. I also solely did the model creation and tweaks.

8. Learning Experience

Throughout this project, I learned a lot about how to visualize data and handle real-world examples. The first hurdle I faced happened when creating the project and learning how much of a limited dataset I had. With only having a trending dataset I had to rework my problem and identify different data metrics on my own. This further pushed me into dealing with text data that I have not personally dealt with before and required me to learn about TF-IDF. Some of the data handling was troublesome as well, which caused me to discover different ways of manipulating it for visuals and data processing.

On the non-technical side, I was able to understand what type of videos usually hit the YouTube trending page and how to classify these videos. Given more time, I would have been able to process other region’s data for comparisons.

REFERENCES

[1] Mitchell J. *”Trending YouTube Statistics“* [*https://www.kaggle.com/datasnack/youtube-new*](https://www.kaggle.com/datasnack/youtube-new) *(2019)*

#### [2]Mohit Bansal “Us YouTube EDA & Category Predictions

#### <https://www.kaggle.com/bansmohit/us-youtube-eda-category-predictions> (2018)