Predicting YouTube View Counts

Jon Braatz & Lance Lamore

November 2018

Abstract

The goal of this project is to use machine learning techniques to predict the popularity of a hypothetical YouTube video about video games that a content creator is considering posting to their channel. The problem is formulated as a regression problem that attempts to minimize the mean relative squared error between the log of the predicted view count and the log of the actual view count. The input to the model is natural language metadata such as title, description, and tags as well as the channel identifier of the content creator. Using these inputs, we search our dataset for the most similar existing videos using a search algorithm we developed that considers similarity of channel statistics and the similarity of semantic content between videos and channels. The statistics of the videos that were found to be most similar were then input into a regression algorithm. Several regression algorithms were tried, and with the most performant one being xgboost, which achieved a mean squared error of 2.231, and the extent to which natural language metadata provides predictability over just the identity of the content creator is quantified.

1 Introduction

YouTube content creators need to understand the factors affecting the popularity of their videos so that they can make better informed decisions about their uploads, leading to greater popularity and advertising revenue. For example, a content creator might have created a new video and have a list of potential title and description ideas, but without the help of tools for prediction and analytics, they have only their intuition to rely on when making a final decision on which natural language metadata to associate with their video. Our project aims to provide a way for content creators to rank their candidates in order of predicted view counts. Specifically, given a title, description, list of tags, and the identifier of the content creator's channel, we seek to predict how many views a video with the given metadata would get if were posted to the given channel. We wish to specifically restrict our predictor to predicting the view counts for videos on gaming channels, as this category is one of the largest and most popular categories on YouTube. The expected behavior of our system when given the following query:

- Channel Id: UCwRGQapSgTW9xba17OCjgXA

- Title: Minecraft - 1.9 Pre-Release - Villagers, Nether Dungeons, and More!
- Description: 1.9 pre-release is crazy! Download it here: http://t.co/ByDKpDUg
- Tags:
  - Minecraft
  - 1.9
  - pre
  - release

would be to output a number close to 3.273, since the true view count of that currently existing video is $e^{3.273} = 1877$.

1.1 Related Works

Previous researchers in this area have attempted to predict view counts based on time series data [1], or done regressions based purely on the statistics of the uploader's most recently posted video [2], links to videos from social media [3], and the sensitivity of view count to metadata-level information [1]. Many of these techniques relied on datasets that are no longer publicly available due to changes in the YouTube API, such as time series data. The dataset available to us was the YouTube-8M dataset, a set of randomly sampled videos that YouTube created for machine learning researchers, in addition to the videos available via the YouTube API.

The work that we wished to extend was the previous work in [2] that had been done in trying to use regression on video statistics to predict view counts in addition to looking at the content of the title and thumbnail. However, that group found that content-based features didn't provide significant predictability over the statistics of the most recently uploaded video, and so we decided not to pursue features based directly on the content of titles, descriptions, tags, or thumbnails. But we did hypothesize that a predictor that takes into account the semantic meanings of the metadata would outperform one that didn't. So instead of using feature extractors directly on the metadata to be used as input in a regression, we hypothesized that the metadata could more effectively be used to find other similar videos in the dataset with similar statistics. Our intuition that knowing how the query metadata related to other videos in the dataset would be more useful than trying to extract view count predictions solely using the metadata itself.

1.2 Approach

The task of predicting view counts is fundamentally a regression problem, however the form of the inputs is textual rather than numerical, and as such they will need to be transformed to a set of numerical features before being fed into a regression algorithm. While we can't effectively do this directly using only the inputs we're given, we do have a dataset of statistics and natural language metadata of 70,000 YouTube videos in the Video Game category that was scraped from the YouTube API. Given that we have some metadata that we would like to predict statistics for, and a dataset of videos with both metadata and statistics, our hypothesis is that videos with "similar" metadata have similar statistics. As part of this project, we've implemented a NLP-based search algorithm to find a list of relevant videos given textual metadata, and use the statistics of those videos as features in a regression.

While our hypothesis that semantically similar videos will have numerically similar statistics seems plausible, it must also be tested against a reasonable null hypothesis. It could be that the identity of a video uploader more or less determines the view counts of the videos they upload, and the titles and descriptions they provide have little effect on video popularity. A baseline predictor that corresponds to this hypothesis is to randomly sample some number of videos from the uploader's channel, and feed those statistics into the regressor. If textual metadata does indeed affect view counts, then a predictor that takes it into account should outperform a baseline predictor that takes only the identity of the uploader into account. For an oracle, we will perform our regression on the statistics of currently existing videos (besides view count). These statistics won't be available when a content creator wants to use this tool, so a regression on these variables is a natural choice of oracle.

2 Model

Our model has two main subcomponents:

1. A search algorithm that uses Natural Language Processing and heuristics to assign scores to videos in our dataset based on their similarity to a given title, description, tags, and information about the uploader.
2. A regressor that takes the statistics of a list of videos ranked by similarity to a given one and predicts the log of the view counts of the given video.

2.1 Search Algorithm

The purpose of the search algorithm is to use video and channel information to find similar videos from similar channels, with the hope that the statistics of the resulting videos will correspond well with the statistics of the query. The dataset has the form of a two-tiered structure, consisting of a set of channels that each are associated with a set of videos. As a result, we will use two similarity metrics in determining similarity of two particular videos, namely a channel-channel similarity and a video-video similarity.

2.1.1 Word Embeddings

The core of our search algorithm works by considering the semantic similarity of two pieces of English text, and we do this by using word embeddings. More specifically, we measure the semantic similarity of two pieces of English text by tokenizing the texts, generating word vectors for each token using the word2vec algorithm, averaging the word vectors in each text, normalizing each document vector to unit Euclidean norm, and computing the dot product of the two resulting vectors. The resulting score is a real number between -1 and 1, with 1 being semantically similar and -1 being semantically opposite. By averaging word vectors in a document, we are effectively assuming a bag-of-words model. We used a large model that was pre-trained on the entire corpus of Wikipedia text using the Spacy Python library.

2.1.2 Channel-Channel Similarity

The channel-channel similarity metric that we chose to implement is hand-engineered and based on plausible heuristics for determining how similar the statistics of videos in those channels would be. Specifically, we consider two channels to be similar if they are of similar size and deal with similar subject matter. While a more sophisticated approach might learn a channel-channel similarity metric optimized for prediction accuracy of the overall view count prediction system, we opted for this simpler approach and found it to give good results.

To find similar channels to a given one, our search algorithm considers all other channels with the same order of magnitude of subscribers, computes word embeddings of the channel descriptions using word2vec, and finds the channels that have the highest cosine similarity with the channel in question.

2.1.3 Video-Video Similarity

After choosing a list of the three most similar channels to the given one, we wish to find the videos in those channels with the textual metadata that is most similar to the query. To do this, we first form word embeddings of the title, description, and tags, and form the sum of the respective cosine similarities of those text fields.

3 Algorithms Used

Our problem is fundamentally a regression problem, and the variables of our regression are video statistics of similar videos, which are available through the YouTube API. In particular, these statistics are:

1. `ViewCount`
2. `CommentCount`
3. `LikeCount`
4. `DislikeCount`

Since we don't want to limit ourselves to linear models, we decided to use gradient boosted regression. We'll define a couple of terms first.

1. Regression: Regression consists of predicting a target value (in our case, the view count of a candidate video) based on several numerical or categorical predictors. In our case, the predictors are video statistics for the 10 videos in the channel that are most similar to the candidiate video according to some similarity metric.
2. Decision Tree: A decision tree is a method of regression that is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values. At each interior node of the tree, we apply a test to one of the predictor variables, and depending on the value of the outcome

of the test we either go to the left or right subtree. Predictions of the target variable correspond to leaves in the tree. A decision tree regression algorithm will construct a decision tree based on training data that minimizes the least square error of the predictions and training target values, subject to a regularization term.

With these two terms defined, We can go into the 6 different algorithmic approaches we used to solve this problem. Drawing mainly from the `sklearn` library, we belived these methods would each provide a novel approach to solving this problem.

1. Ridge Regression: This is essentially an ordinary least squares regression with an added penalty for larger coefficients. Regular least square regressions assume that features are independent of each other and this can cause inaccurate predictions. This method, helps reduce the effects of collinearity in the feature space.
2. Lasso Regression: This is another least squares regression with a penalty factor for unimportant features. This attempts to compress the feature space and reduce the number of parameters that the out put should depend on
3. Support Vector Machines: Uses subset of training points in decision tree and fits it to a specified Kernel function. We chose to use to the default kernel function, the rbf function. This
4. Random Forest: This creates an ensemble of decision trees that are each trained on random sub-samples of the training set. This means a diverse set of regressions is created by introducing randomness in the regression construction. The prediction of the ensemble is given as the averaged prediction of the individual regressions.
5. Adaptive Boosting: This is also a tree model, which uses an ensemble decision tree method that combines weak learners into a single strong one. It is trained on a modified version of the training set where the most complicated examples gain ever-increasing influence.
6. XGBoost: XGBoost is a gradient tree boosting based on the idea of creating a potentially weak regression tree on the data, creating a regression tree for the residuals of the first model, and then sequentially applying this idea to create a strong regressor out of weak regressors.

4 Error Analysis

. The best regressors were found to be the tree-based models, meaning that Random Forest, Adaptive Boosting, and XGBoost all provided similar results, with the best mean squared error being 2.231 by XGBoost and the others being within .2 of that. Note that these mean squared errors are in terms of log-transformed views, so these numbers correspond to an absolute error of a factor of approximately $e^{\sqrt{2.231}} = 4.45$. This means that our estimates are either to great by a factor of 4.45 or too low by a factor of 4.45. The potential reasons for errors can be found by considering our assumptions and hypotheses, and thinking of aspects of our model or dataset that might break them.

1. We assume that videos from similar channels about similar things will have similar statistics. We found that this assumption is broken by outliers like viral videos that show view patterns significantly different than other ones in the channel. The reason that these outliers are predicted poorly by our model is likely that our search algorithm restricts its search to other similarly sized channels, and videos that have an outsized view count relative to the rest of the videos in the uploaders channel won't give fruitful results when compared with other videos in similarily sized channels that are likely to have far fewer views than the viral video in question. This would explain why our model tends to underpredict the true view counts.
2. When finding channels that are similar to a given channel, we restrict our search to channels that have within the same order of magnitude of subscribers. This might not be the best filter to use, and it could be that two channels could be very similar and yet have wildly different subscriber counts, say if their view counts are similar but one's audience is more consistent.
3. The most important statistic for each video was by far the view count, so this gives us reason to think that our model might benefit from a dimensionality reduction step.

The table has the following performance metrics for the different methods used. R^2 and Mean squared error values are provided for the test set across the different predictors, which were tested with 3-fold cross validation. The R^2 values are the coefficients of determination for the prediction, where a score of 1 represents perfect prediction and a score of zero represents predicting the expected value of Y every time. R^2 values can be negative if the predictions are worse.

Algorithm Used	R^2	Mean Squared Error
XGBoost(Oracle)	.731	1.56
XGBoost (baseline)	.531	3.255
XGBoost (with NLP)	.653	2.231
Random Forest	.621	2.440
Adaptive Boosting	.593	2.618
Ridge Regression	-0.052	6.781
Lasso Regression	-0.042	6.719
Support Vector Regression	0.011	6.368

We see that the linear regressions models are performing extremely poorly, often doing worse than just predicting the expected value of Y. The tree models on the other hand were more far better suited for this task, provided relatively useful results. Since Gradient Boosting was the most successful predictor, we will focus on the results of this predictor for the remainder of the section. Gradient tree boosting allows for nice visualization of features weights, which are depicted below for the model and baseline.

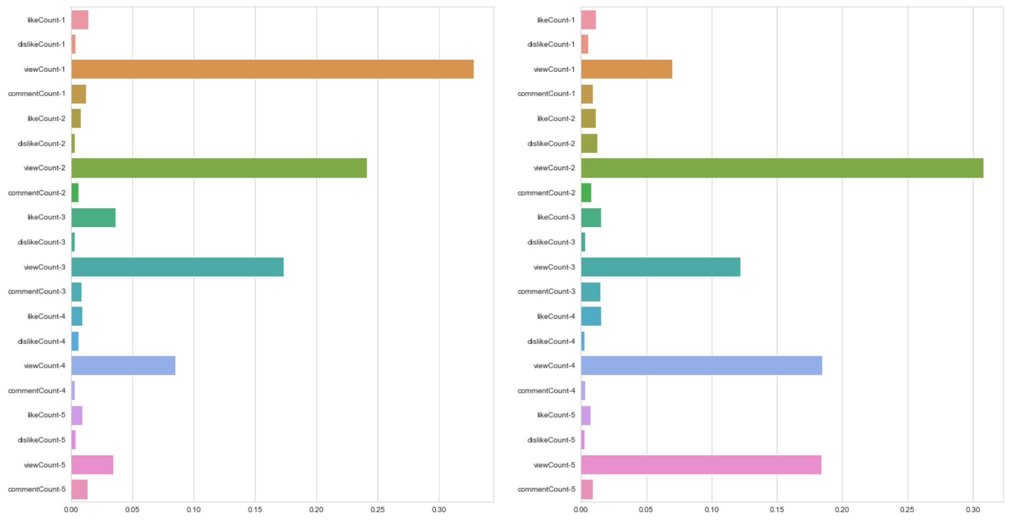

Figure 1: "The feature importances of our regressor based on natural language processing on the left shows structure and predictability, with importance decreasing as videos lower in the relevance ranking are used, in contrast to the baseline on the right."

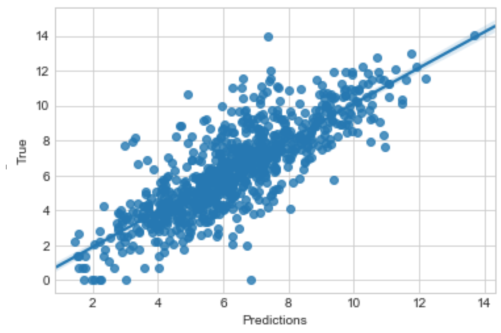

Figure 2: "The accuracy of our most performant model was shown to be a mean squared error of 2.231."

To quantify the extent to which natural language metadata affects a video's popularity, we used the most performant regressor, namely XGBoost, with our baseline search algorithm of choosing statistics from 5 random videos from the same channel. We found that taking metadata into account resulted in a mean squared error that was 31.4% lower than the baseline, indicating that a content creator's choice of title, description, and tags affects a video's popularity to a significant degree. Our oracle maintained a mean-squared error of 1.56, showing that there is still much room for improvement.

We can also take a look at the correlation matrix to see how the statistics of the videos are related:

	viewCount	likeCount	dislikeCount	commentCount
viewCount	1.000	.563	.655	.280
likeCount	.562	1.000	.461	.383
dislikeCount	.655	.461	1.000	.247
commentCount	.280	.383	.247	1.000

The eigenvalues of this correlation matrix are, in descending order, 2.329, .838, .507, and .323. The largest eigenvalue is far greater than the others, suggesting that most of the variance is explained by a single dimension. Instead of using 4 statistics of 5 videos for the regression for a total of 20, we could cut our model down to 5 variables by regressing on the dot product of the list of statistics with the first principal component of the correlation matrix.

References

[1] Hoiles, W. and Krishnamurthy, V., *Engagement and Popularity Dynamics of YouTube Videos and Sensitivity to Meta-Data*,

https://cpb-us-w2.wpmucdn.com/sites.coecis.cornell.edu/dist/f/83/files/2016/08/HAK17-1a53ni4.pdf

[2] Aravind Srinivasan, *YouTube Views Predictor*,

https://towardsdatascience.com/youtube-views-predictor-9ec573090acb

[3] Kong, K. et al., *Will This Video Go Viral? Explaining and Predicting the Popularity of Youtube Videos*,

https://arxiv.org/pdf/1801.04117.pdf