SAVE THE DISLIKES

Analysis & Prediction Of Dislikes On YouTube Data

By, James Mete, Jenna Mekled, & Sashaank Sekar

Team Save the Dislikes GitHub

Introduction:

Dislike counts on YouTube videos are a valuable signal for separating high-quality videos from low-quality videos or even potential scams. YouTube has recently removed the ability to see dislike count data publicly and has disabled dislike count data in their API. Our project seeks to understand the trends that influence dislike activity through analytical research and use those insights to generate a machine learning model to predict the quality of a YouTube video and explore ratios to help alert users to potentially problematic videos based on the available data and features.

Data Acquisition & Processing:

Since YouTube has removed the ability to scrape or query dislike counts actively, we must rely on historical data to conduct our research. Luckily, extensive historical records have been kept by Archive.org. For this project, we will be leveraging "YouTube Metadata Collection (2019-02)," which consists of around 1.46 Billion JSON records related to YouTube metadata such as title, upload date, category, likes, dislikes, language, recommended videos, and other features, a complete list is available on the archive site linked. This archive is composed of around 5000 .tar files, each with about 146 .json.gz files inside them. We first ran our downloadtars.py file to download the archive tar files from Archive.org, and then ran combinejson.py to process the JSON files into 301 parquet files for easier IO processing. Finally, using our dataentry_from_parquet.py program, which will create .pickle files used to store paths that have already been processed, we were able to loop through the stored parquet files, perform some initial data cleaning and processing, arrange and rename columns in the same order/name as the PostgreSQL table schema expects, and then load the data into the database table. After downloading the first 25 files and every fifth file afterward, our data consisted of 80,910,144 videos.

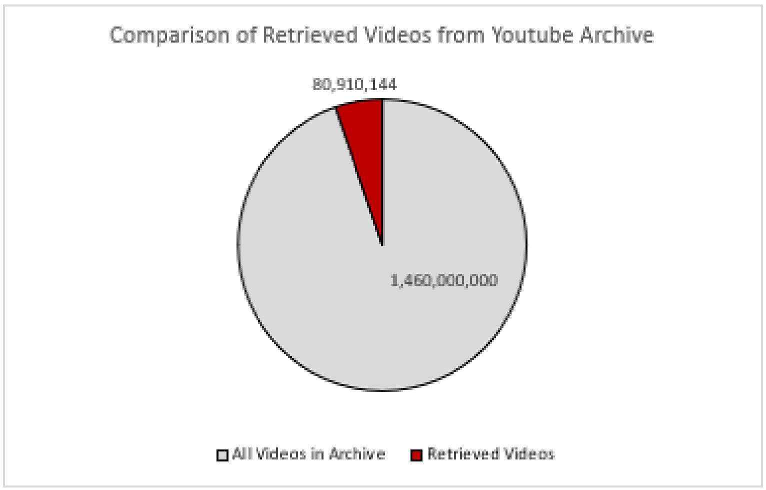
Data processing steps performed at this step include:

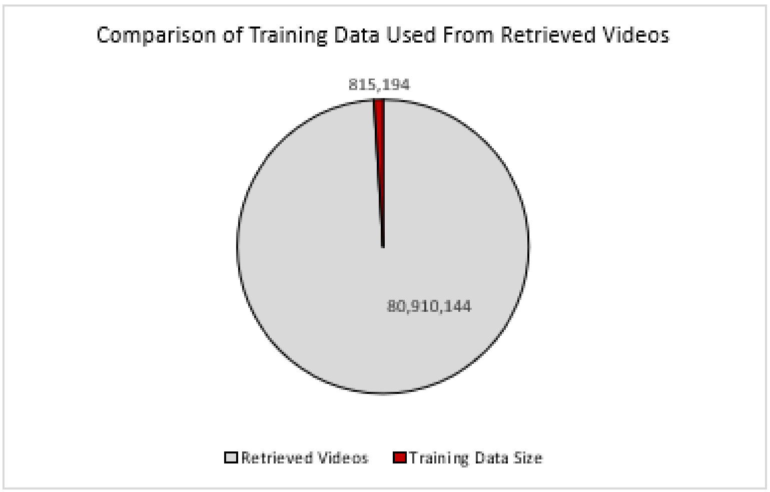
1. Converting date columns to datetime format.
2. Converting dicts to json string objects that are suitable to be imported into PostgreSQL for a JSONB formatted column.
3. Calculates initial ratios including view_like_ratio, view_dislike_ratio, and like_dislike_ratio.
4. Drops INF and NaN rows for the like_dislike_ratio to reduce dataset size and potential errors later on. We mainly only care about videos that actually have dislikes.
5. Rename and Reorder columns to match database table schema.
6. Loads data into PostgreSQL using batch importing utilizing the pd.to_sql command with multi mode enabled and a chunksize of 10,000 as default.

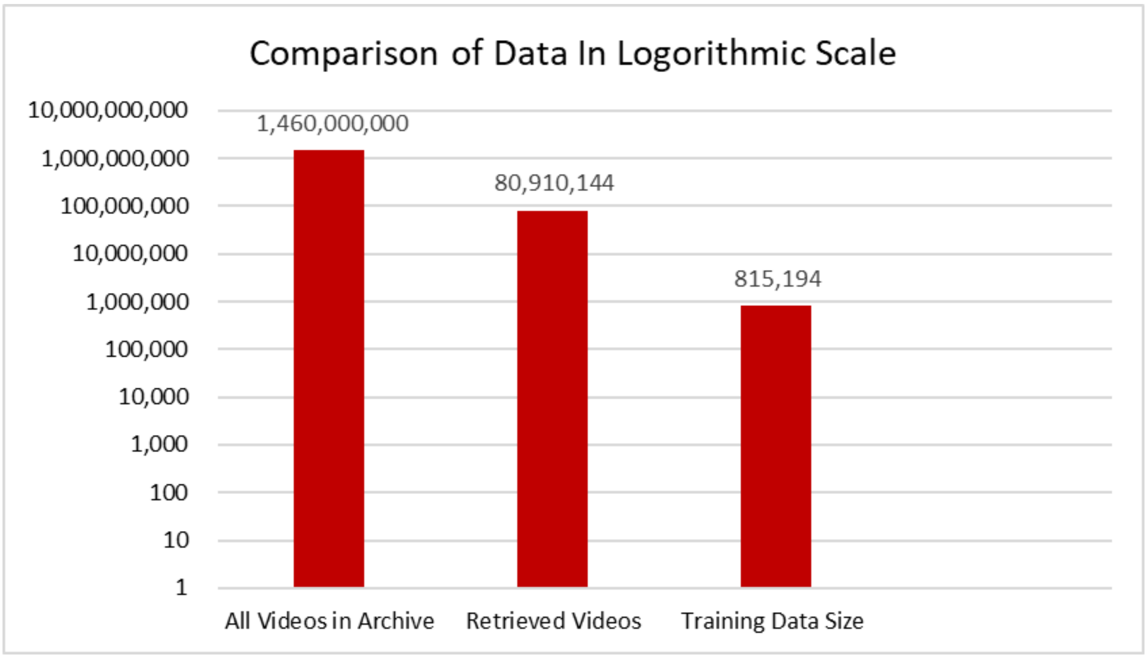
Within our database, leveraging the “ANALYZE” command and creating an index for important columns, we optimized the database, shortening processing times from over 8 hours to a few minutes or even seconds, depending on the complexity of the SQL command.

For this project's scope, our processing techniques involved taking a sample of our full dataset and complementing it with web scraped comments of the top liked and disliked videos as well as a random 1% sample. Once our data was inside PostgreSQL, we used SQL commands to run operations to extract a smaller sample size better suited for statistical analysis. The CSV files we extracted were the following:

- Exporting the top 500,000 most disliked rows based on dislike_like_ratio.
- Exporting the top 500,000 most liked rows based on dislike_like ratio.
- Exporting a 1% random sample resulting in over 800,000 rows.
- Exporting a 10% random sample. This will be used for testing if a lot more data improves predictive performance.
- Exporting a 0.2% random sample used for final testing.







Methodology:

Downloading Comment Data

For this project, we will be constructing a Supervised Machine Learning Model to predict the quality of a YouTube Video in the absence of a dislike button. Before building our model, we needed to clean up our dataset, add our comment data, engineer additional components and select our final features. To gain further insight and acquire more data for our model to use, we decided to download the top 10 comments of each video in our exported dataset, which included the entire comment, number of votes, and hearts (can only be given by the video owner).

After modifying a package on GitHub (cdownload_noargs.py) to aid in our web scraping of comments, we then made a python program to integrate with the modified function to loop through the videos automatically, download the comments and related data, and store them in both individual JSON files as well as overall CSV files. This provided us with a dataset including the above values by video_id, which we were able to join with our original dataset.

Feature Engineering

Once the original and comment datasets were joined together, any remaining JSON columns were removed, and categorical (text) variables were converted to discrete (numeric) values to be included in our feature selection process. Using NLTK VADER (Valence Aware Dictionary for Sentiment Reasoning), a model for text sentiment analysis, we assessed the sentiment scores of the comments and video description on each video, returning scores in four categories: Negative, Neutral, Positive, Compound (calculated by normalizing all scores). This allowed us to put our comment and description data in a format for our model.

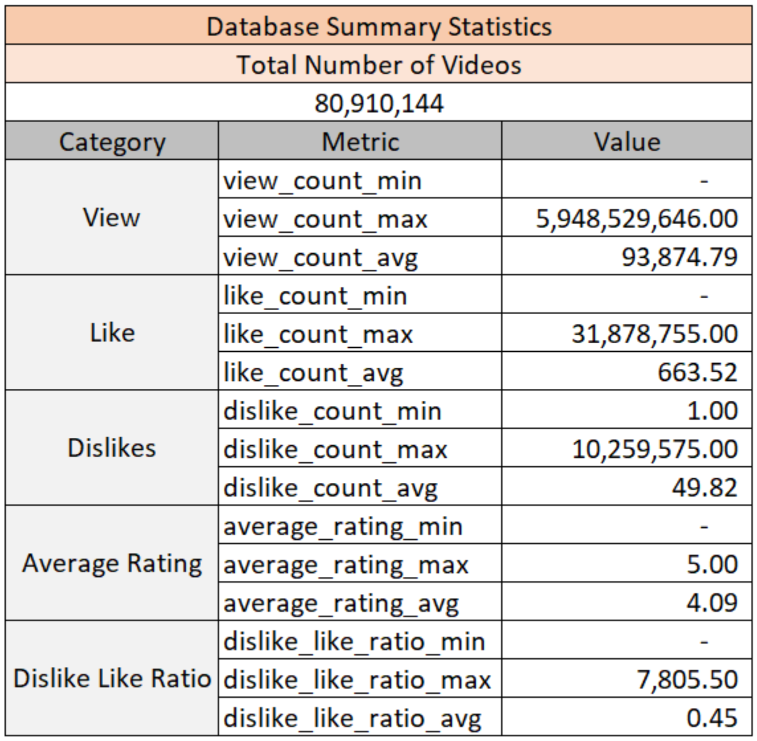
Our initial analysis led us to ideas for feature engineering that we could calculate based on the available data. Features we created to assist our analysis and model building include:

- LD Score: Likes / (Likes + Dislikes)
- LD Score OHE: Converting decimal LD Score to categorical -1 (negative), 0 (neutral), and 1 (positive).
- View_Like Ratio: view_count / like_count
- View_Like Ratio Smoothed: If like_count is 0, we add 1 to view_count and like_count to avoid division by 0.
- View_Dislike Ratio: view_count / dislike_count
- Dislike-Like Ratio: dislike_count / like_count (smoothed by 1 to avoid division by 0)
- NoCommentsBinary: 0 if the video had comments, and 1 if the video did not have comments when we attempted to pull the data.
- VADER Sentiment Scores: (neg, neu, pos, compound)

Although we can not use any of the features involving dislike count at inference time, we used them in our analysis to better understand the data and filter our primary dataset to extract applicable rows such as the most disliked or liked video rows based on dislike_like_ratio. We believed these extremes, combined with the random 1%, would allow for a diverse set of video information for our machine learning models to train on.

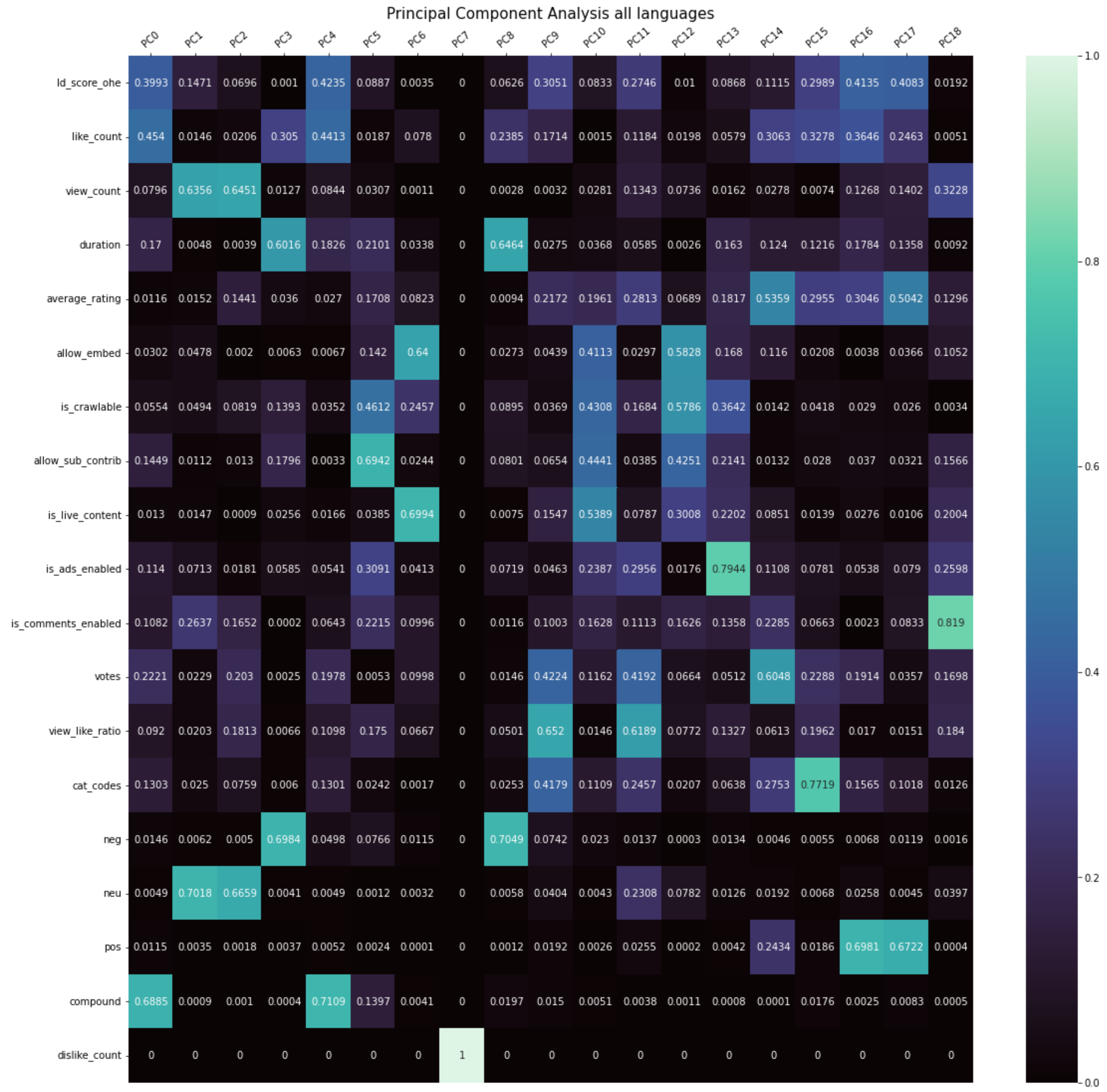
Feature Selection

We performed multiple initial analysis steps to help uncover any insights regarding the data. One of our main goals was to discover if any features were correlated or valuable for predicting our video ranking category. We ultimately decided to create a one-hot encoded Like Dislike Score (ld_score_ohe) category for our ranking system. We took the percentage of total likes over the sum of likes and dislikes, which we used to create our like dislike score (ld_score), then encoded the resulting percentages. Videos scoring 0.75 or over were rated as 1 (Good), anything equal to or lower than 0.50 was rated as -1 (Bad), and anything with scores between 0.51 and 0.74 was rated as 0 (Neutral). This set of criteria in our research was loosely developed based on what YouTubers consider a "Good" or "Bad" video to be concerning their like to dislike ratio. The analysis we performed included a statistical analysis of the complete database, our subsets that have been exported into CSVs, and comparisons of the extremes of most liked and disliked videos across multiple features. In addition, we also performed both a Principal Component Analysis and a Feature Correlation Analysis.



Database Summary Statistics		
Total Number of Videos		
80,910,144		
Category	Metric	Value
View	view_count_min	-
	view_count_max	5,948,529,646.00
	view_count_avg	93,874.79
Like	like_count_min	-
	like_count_max	31,878,755.00
	like_count_avg	663.52
Dislikes	dislike_count_min	1.00
	dislike_count_max	10,259,575.00
	dislike_count_avg	49.82
Average Rating	average_rating_min	-
	average_rating_max	5.00
	average_rating_avg	4.09
Dislike Like Ratio	dislike_like_ratio_min	-
	dislike_like_ratio_max	7,805.50
	dislike_like_ratio_avg	0.45

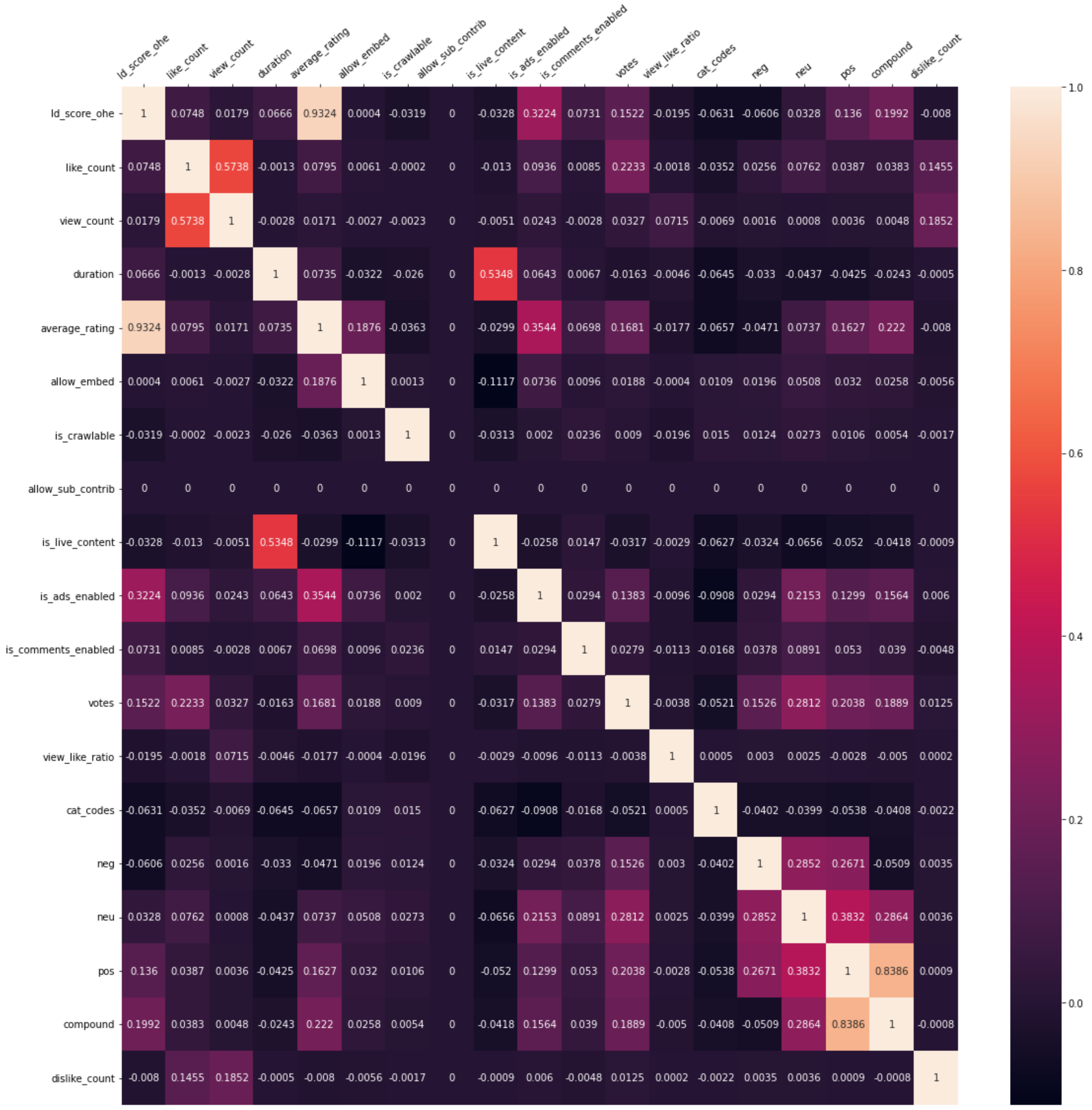
The dimensionality reduction technique, Principal Component Analysis (PCA), aided in our feature selection process for our model. After cleaning our comment text data, we extracted the following table below using the PCA Decomposition feature from scikit-learn. According to our results, *like-count, average rating, is_ads_enabled, votes, neg, neu, pos,* and *compound* had the most significant associations with *ld_score_ohe* making these features top contenders for our model.



	PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16	PC17	PC18
ld_score_ohe	0.3993	0.1471	0.0696	0.001	0.4235	0.0887	0.0035	0	0.0626	0.3051	0.0833	0.2746	0.01	0.0868	0.1115	0.2989	0.4135	0.4083	0.0192
like_count	0.454	0.0146	0.0206	0.305	0.4413	0.0187	0.078	0	0.2385	0.1714	0.0015	0.1184	0.0198	0.0579	0.3063	0.3278	0.3646	0.2463	0.0051
view_count	0.0796	0.6356	0.6451	0.0127	0.0844	0.0307	0.0011	0	0.0028	0.0032	0.0281	0.1343	0.0736	0.0162	0.0278	0.0074	0.1268	0.1402	0.3228
duration	0.17	0.0048	0.0039	0.6016	0.1826	0.2101	0.0338	0	0.6464	0.0275	0.0368	0.0585	0.0026	0.163	0.124	0.1216	0.1784	0.1358	0.0092
average_rating	0.0116	0.0152	0.1441	0.036	0.027	0.1708	0.0823	0	0.0094	0.2172	0.1961	0.2813	0.0689	0.1817	0.5359	0.2955	0.3046	0.5042	0.1296
allow_embed	0.0302	0.0478	0.002	0.0063	0.0067	0.142	0.64	0	0.0273	0.0439	0.4113	0.0297	0.5828	0.168	0.116	0.0208	0.0038	0.0366	0.1052
is_crawlable	0.0554	0.0494	0.0819	0.1393	0.0352	0.4612	0.2457	0	0.0895	0.0369	0.4308	0.1684	0.5786	0.3642	0.0142	0.0418	0.029	0.026	0.0034
allow_sub_contrib	0.1449	0.0112	0.013	0.1796	0.0033	0.6942	0.0244	0	0.0801	0.0654	0.4441	0.0385	0.4251	0.2141	0.0132	0.028	0.037	0.0321	0.1566
is_live_content	0.013	0.0147	0.0009	0.0256	0.0166	0.0385	0.6994	0	0.0075	0.1547	0.5389	0.0787	0.3008	0.2202	0.0851	0.0139	0.0276	0.0106	0.2004
is_ads_enabled	0.114	0.0713	0.0181	0.0585	0.0541	0.3091	0.0413	0	0.0719	0.0463	0.2387	0.2956	0.0176	0.7944	0.1108	0.0781	0.0538	0.079	0.2598
is_comments_enabled	0.1082	0.2637	0.1652	0.0002	0.0643	0.2215	0.0996	0	0.0116	0.1003	0.1628	0.1113	0.1626	0.1358	0.2285	0.0663	0.0023	0.0833	0.819
votes	0.2221	0.0229	0.203	0.0025	0.1978	0.0053	0.0998	0	0.0146	0.4224	0.1162	0.4192	0.0664	0.0512	0.6048	0.2288	0.1914	0.0357	0.1698
view_like_ratio	0.092	0.0203	0.1813	0.0066	0.1098	0.175	0.0667	0	0.0501	0.652	0.0146	0.6189	0.0772	0.1327	0.0613	0.1962	0.017	0.0151	0.184
cat_codes	0.1303	0.025	0.0759	0.006	0.1301	0.0242	0.0017	0	0.0253	0.4179	0.1109	0.2457	0.0207	0.0638	0.2753	0.7719	0.1565	0.1018	0.0126
neg	0.0146	0.0062	0.005	0.6984	0.0498	0.0766	0.0115	0	0.7049	0.0742	0.023	0.0137	0.0003	0.0134	0.0046	0.0055	0.0068	0.0119	0.0016
neu	0.0049	0.7018	0.6659	0.0041	0.0049	0.0012	0.0032	0	0.0058	0.0404	0.0043	0.2308	0.0782	0.0126	0.0192	0.0068	0.0258	0.0045	0.0397
pos	0.0115	0.0035	0.0018	0.0037	0.0052	0.0024	0.0001	0	0.0012	0.0192	0.0026	0.0255	0.0002	0.0042	0.2434	0.0186	0.6981	0.6722	0.0004
compound	0.6885	0.0009	0.001	0.0004	0.7109	0.1397	0.0041	0	0.0197	0.015	0.0051	0.0038	0.0011	0.0008	0.0001	0.0176	0.0025	0.0083	0.0005
dislike_count	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

Performing a person's correlation on the data frame allowed us to see that average_rating produced the highest correlation with ld_score_ohe. This metric is a YouTube Algorithm generated metric. Additional significant features included is_live_content, is_ads_enabled, is_comments enabled, neutral, positive and compound.

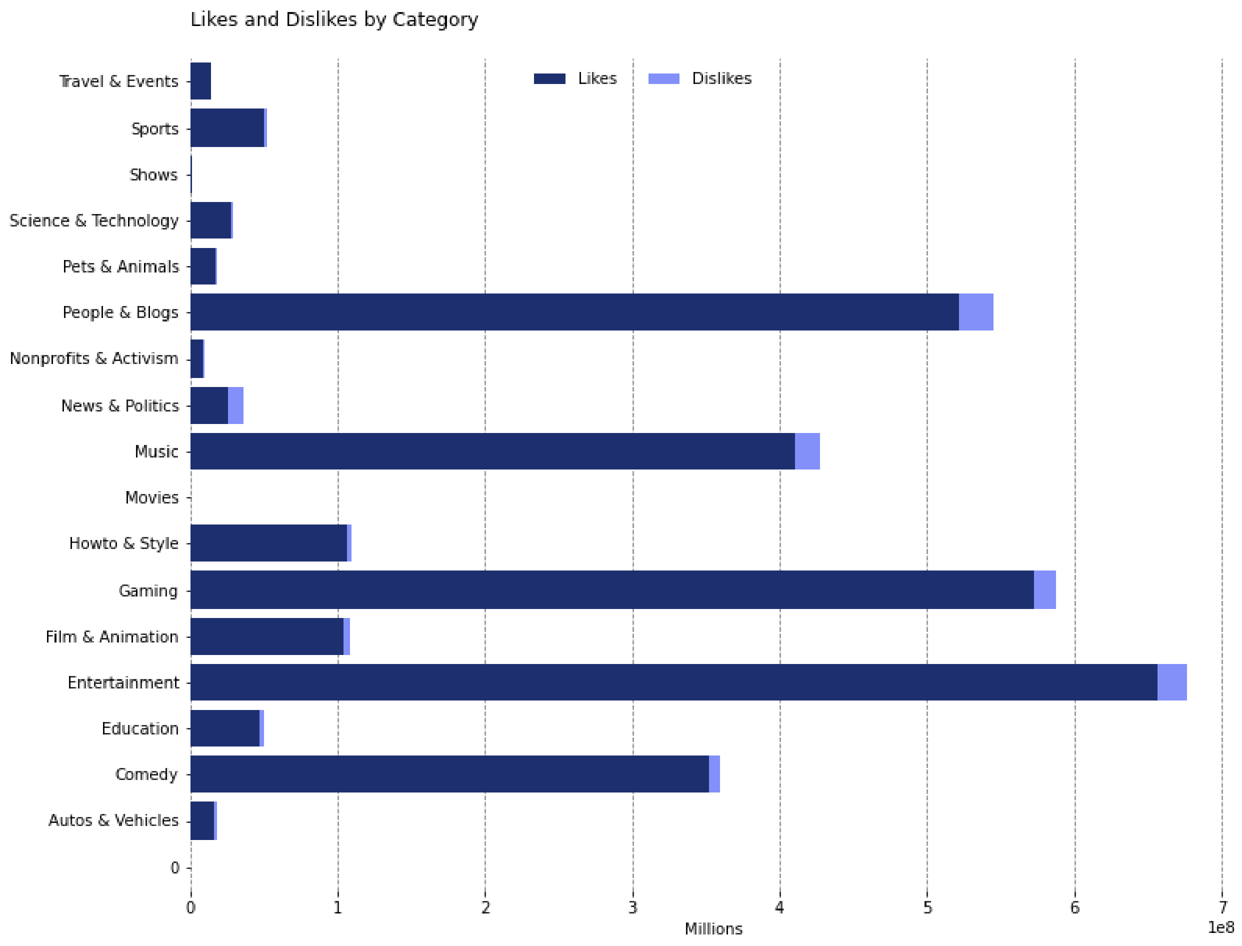
Correlation Matrix



	ld_score_ohe	like_count	view_count	duration	average_rating	allow_embed	is_crawlable	allow_sub_contrib	is_live_content	is_ads_enabled	is_comments_enabled	votes	view_like_ratio	cat_codes	neg	neu	pos	compound	dislike_count
ld_score_ohe	1	0.0748	0.0179	0.0666	0.9324	0.0004	-0.0319	0	-0.0328	0.3224	0.0731	0.1522	-0.0195	-0.0631	-0.0606	0.0328	0.136	0.1992	-0.008
like_count	0.0748	1	0.5738	-0.0013	0.0795	0.0061	-0.0002	0	-0.013	0.0936	0.0085	0.2233	-0.0018	-0.0352	0.0256	0.0762	0.0387	0.0383	0.1455
view_count	0.0179	0.5738	1	-0.0028	0.0171	-0.0027	-0.0023	0	-0.0051	0.0243	-0.0028	0.0327	0.0715	-0.0069	0.0016	0.0008	0.0036	0.0048	0.1852
duration	0.0666	-0.0013	-0.0028	1	0.0735	-0.0322	-0.026	0	0.5348	0.0643	0.0067	-0.0163	-0.0046	-0.0645	-0.033	-0.0437	-0.0425	-0.0243	-0.0005
average_rating	0.9324	0.0795	0.0171	0.0735	1	0.1876	-0.0363	0	-0.0299	0.3544	0.0698	0.1681	-0.0177	-0.0657	-0.0471	0.0737	0.1627	0.222	-0.008
allow_embed	0.0004	0.0061	-0.0027	-0.0322	0.1876	1	0.0013	0	-0.1117	0.0736	0.0096	0.0188	-0.0004	0.0109	0.0196	0.0508	0.032	0.0258	-0.0056
is_crawlable	-0.0319	-0.0002	-0.0023	-0.026	-0.0363	0.0013	1	0	-0.0313	0.002	0.0236	0.009	-0.0196	0.015	0.0124	0.0273	0.0106	0.0054	-0.0017
allow_sub_contrib	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
is_live_content	-0.0328	-0.013	-0.0051	0.5348	-0.0299	-0.1117	-0.0313	0	1	-0.0258	0.0147	-0.0317	-0.0029	-0.0627	-0.0324	-0.0656	-0.052	-0.0418	-0.0009
is_ads_enabled	0.3224	0.0936	0.0243	0.0643	0.3544	0.0736	0.002	0	-0.0258	1	0.0294	0.1383	-0.0096	-0.0908	0.0294	0.2153	0.1299	0.1564	0.006
is_comments_enabled	0.0731	0.0085	-0.0028	0.0067	0.0698	0.0096	0.0236	0	0.0147	0.0294	1	0.0279	-0.0113	-0.0168	0.0378	0.0891	0.053	0.039	-0.0048
votes	0.1522	0.2233	0.0327	-0.0163	0.1681	0.0188	0.009	0	-0.0317	0.1383	0.0279	1	-0.0038	-0.0521	0.1526	0.2812	0.2038	0.1889	0.0125
view_like_ratio	-0.0195	-0.0018	0.0715	-0.0046	-0.0177	-0.0004	-0.0196	0	-0.0029	-0.0096	-0.0113	-0.0038	1	0.0005	0.003	0.0025	-0.0028	-0.005	0.0002
cat_codes	-0.0631	-0.0352	-0.0069	-0.0645	-0.0657	0.0109	0.015	0	-0.0627	-0.0908	-0.0168	-0.0521	0.0005	1	-0.0402	-0.0399	-0.0538	-0.0408	-0.0022
neg	-0.0606	0.0256	0.0016	-0.033	-0.0471	0.0196	0.0124	0	-0.0324	0.0294	0.0378	0.1526	0.003	-0.0402	1	0.2852	0.2671	-0.0509	0.0035
neu	0.0328	0.0762	0.0008	-0.0437	0.0737	0.0508	0.0273	0	-0.0656	0.2153	0.0891	0.2812	0.0025	-0.0399	0.2852	1	0.3832	0.2864	0.0036
pos	0.136	0.0387	0.0036	-0.0425	0.1627	0.032	0.0106	0	-0.052	0.1299	0.053	0.2038	-0.0028	-0.0538	0.2671	0.3832	1	0.8386	0.0009
compound	0.1992	0.0383	0.0048	-0.0243	0.222	0.0258	0.0054	0	-0.0418	0.1564	0.039	0.1889	-0.005	-0.0408	-0.0509	0.2864	0.8386	1	-0.0008
dislike_count	-0.008	0.1455	0.1852	-0.0005	-0.008	-0.0056	-0.0017	0	-0.0009	0.006	-0.0048	0.0125	0.0002	-0.0022	0.0035	0.0036	0.0009	-0.0008	1

Preparing Data for Machine Learning

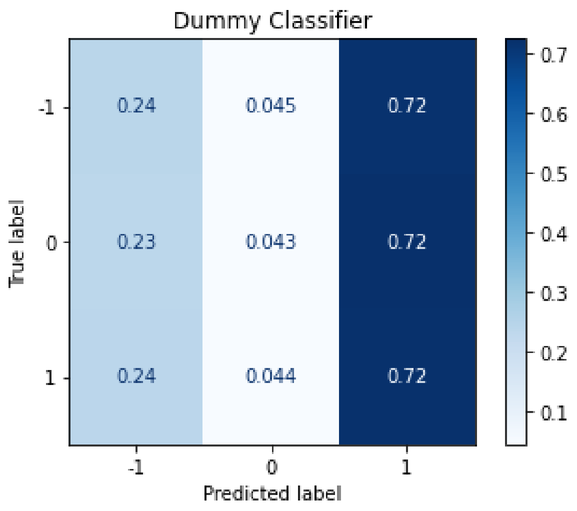
After selecting the features for our model we made the decision that we would train on Model on English Videos only as these videos contained comments in a language that could be fed to our sentiment analysis. As there was no way for us to determine which videos were in English we found that by running the cleaned description text field of each video through our sentiment analysis, any video in a foreign language returned a value of 1 in the neutral column. Filtering out the videos that met that criteria allowed us to retain a dataset of english videos only. We developed a pipeline to properly format our data as suitable data frames for our machine learning models. In developing our model, we explored numerous types of models to determine which model had the best performance for our task. Our objective, in this case, is not actually to get the overall best performance on the entire dataset but rather a specific focus on identifying “Bad” videos. Thus, we relied on numerous metrics to help give a holistic view of the model's performance. Due to our imbalanced dataset (“Good” videos are much more present than “Bad”or “Neutral” videos), we focused on various F1 scores(percentage), Accuracy (percentage), and MCC (Matthews correlation coefficient, ranges between -1 to 1 for positive and negative correlation) to track model performance. Furthermore, we checked each model's confusion matrix to better understand which classes/labels our models were performing well on and which they were performing poorly on. Below is a summary of the average likes and dislikes per category in our final dataset for the model.



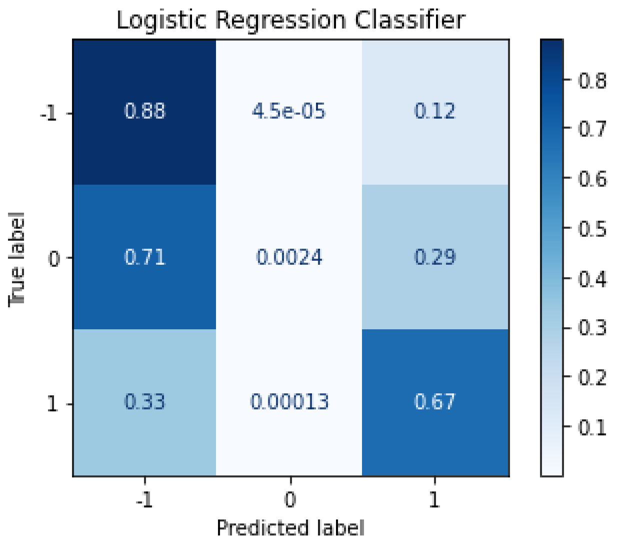
Model Testing

After selecting our features and preparing our data we ran it through various models to ensure that we produced the best results possible. In total we attempted six different models and the results can be found below.

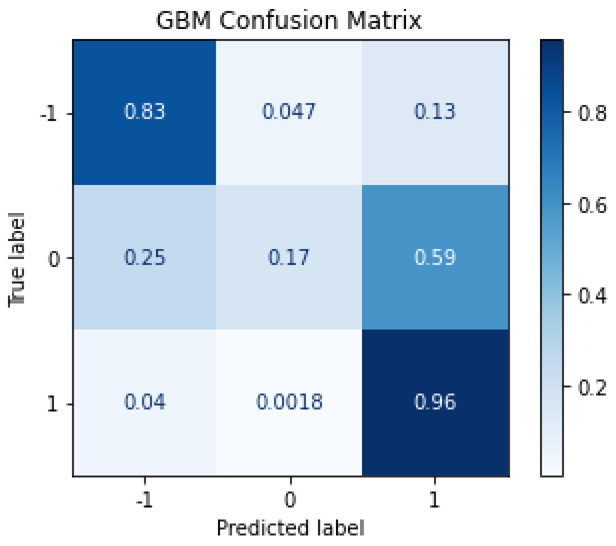
1. **Dummy Classifier**: To get a baseline of performance. This model returned an accuracy score of 57%, a weighted F1 score of 58% and an MCC score of 0.0008 indicating no correlation.



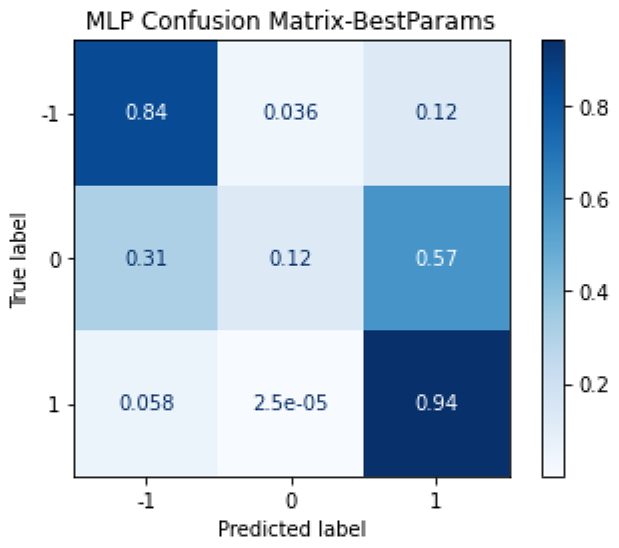
2. **Logistic Regression:** It performed better than the Dummy but wasn't ideal. However, we did not tune it much. The accuracy score was 62%, an F1 Score 63% and an MCC score of 0.3311 indicating weak positive correlation.

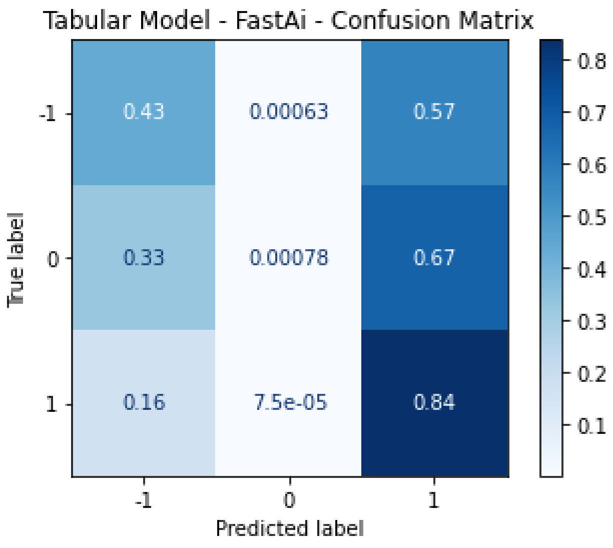


3. **Gradient Boosting Machines:** Performed quite well on the overall testing dataset but sacrificed “Bad” video accuracy in favor of “Good” video accuracy which is not ideal for our specific task. This model had an accuracy score of 85%, an F1 Score of 82% and an MCC of 0.6093 indicating a strong positive correlation.

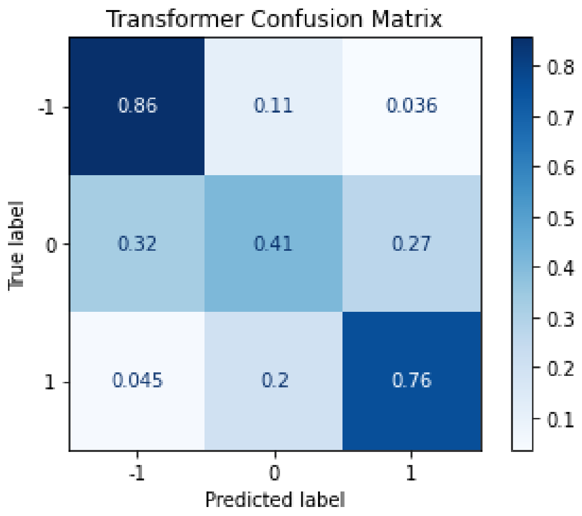


4. **Multi-Layer Perceptron:** Decent performance but took a long time to train using sklearn. We also tried a GPU accelerated tabular learner through fast.ai but both didn't have the best performance. Each of them seemed to completely ignore the neutral videos. The MLP model had an accuracy score of 83%, an F1 Score of 80%, and an MCC of 0.5741 indicating moderately strong positive correlation. The FastAi MLP model had an accuracy score of 68%, an F1 Score of 65%, and an MCC of 0.1890 indicating weak positive correlation.

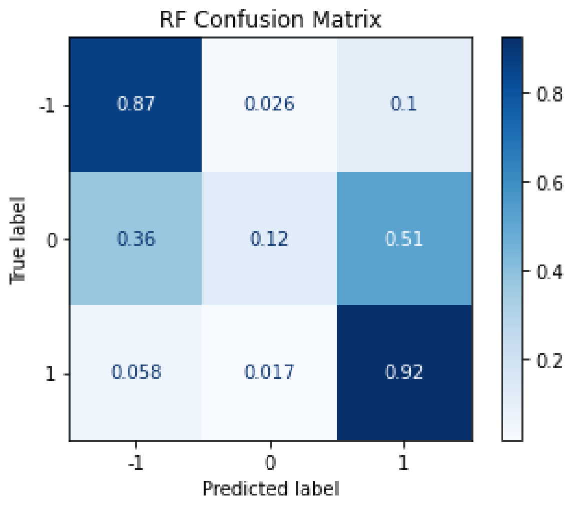




5. **Transformer Network:** Deep learning has become a pillar of the modern world, especially with the rise of language models. We were curious if processing our features as one long string (separated by special tokens) would work. While it took nearly an hour to train on a GPU, after some tweaking the performance of the transformer network performed quite well. The pre-trained base model we used was microsoft/deberta-v3-small via the HuggingFace library. This model had an accuracy of 77%, an F1 score of this model was 80% and an MCC of 0.4999 indicating a positive correlation.

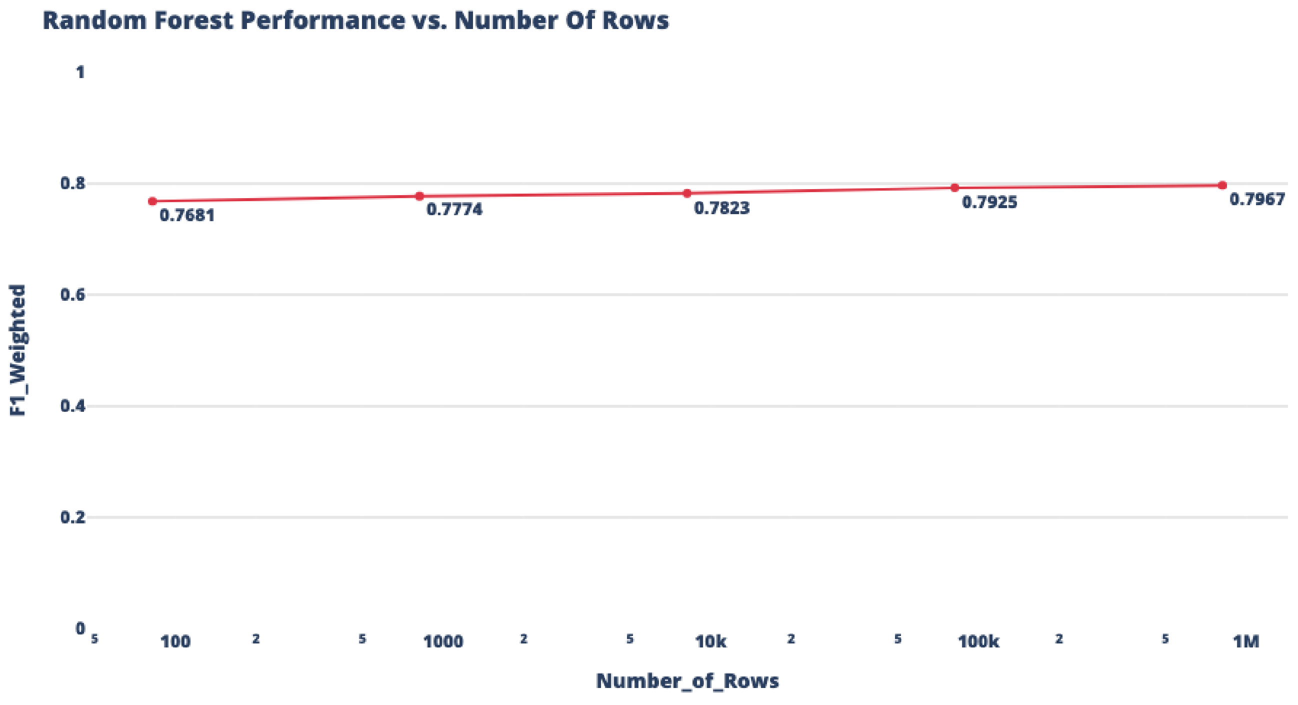


6. **Random Forest:** Performed the best overall with minimal tuning, and was not impacted by standardization. The Random Forest was quick to train, performed well even with default settings, and was very interpretable due to the easy visualization capabilities of feature importance. The results were an accuracy score of 82%, an F1 Score of 80% and an MCC of 0.5641 indicating a moderately strong positive correlation.

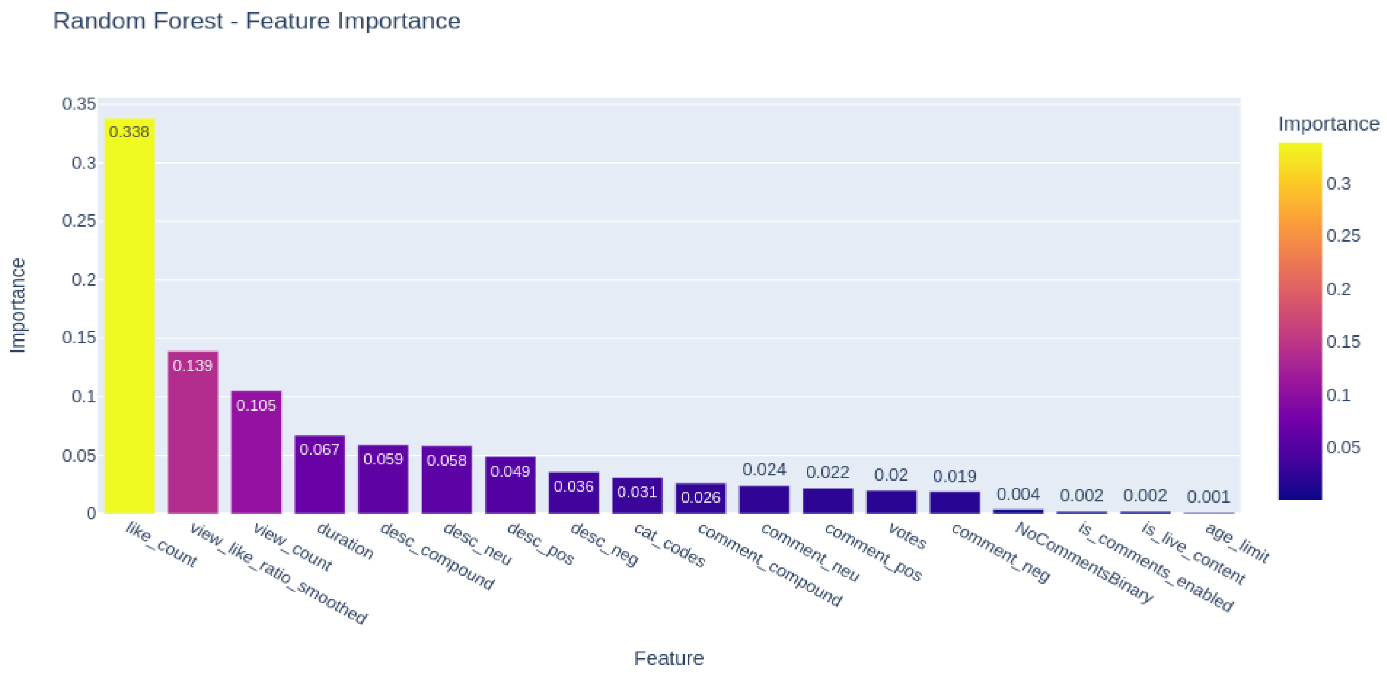


Results & Discussion

Ultimately, we decided to go with the Random Forest. It was quick to train, provided good predictive performance, had short inference times, and had easily interpretable performance by examining the feature importance graphs. Our comprehensive testing across many different models, parameters, and even sample size variations put the Random Forest with custom class weights in the top spot with 82% overall accuracy and a 0.80 weighted F1 score on our test set. Our model results have approximately an 87% chance of correctly classifying "Bad" videos and a 92% chance of correctly identifying "Good" videos. The model can only predict neutral videos with an accuracy of 12%. While this is lower than we desired, This is ideal for our task where we care more about the minority class (the "Bad" videos) but don't want to completely disregard the overall dataset, which is dominated by the "Good" videos. Therefore, we selected the model with the highest accuracy level for predicting "Bad" videos. Interestingly, the Random Forest was able to get sufficiently high performance even with minimal number of training data, which shows the importance of well-chosen features and model architecture.



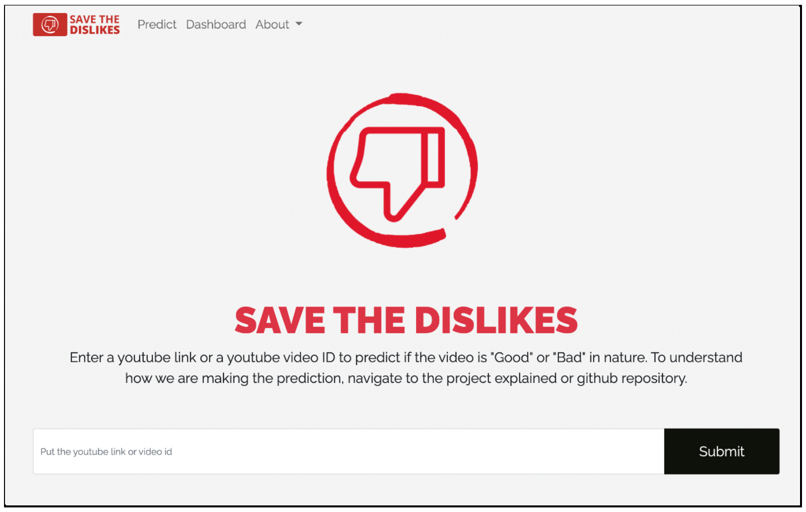
Our final model uses all the features to maximize predictive performance over runtime or storage costs, but eliminating minimal features can be helpful if the model is intended to be used in a scaled-up version in the future. Our feature importance analysis revealed that like_count was the most important feature, followed by the view_like_ratio_smoothed and the view_count. Other features such as duration and our sentiment columns also helped a decent amount. It also helped showcase that features such as our boolean columns of age_limit, is_live_enabled, and is_comments_enabled are less valuable. We tested removing multiple features based on our results and faced limited degradation in terms of predictive performance. However, the model did start to drift towards heavily predicting the "Good" class (1) as more features were removed.



Web App

We have developed a web app that easily allows a user to submit a video ID or url and receive a prediction based on our model of whether that video is considered negative, neutral, or positive. The process is as follows:

- The server takes in the video, and runs our YouTube API and web scraper for relevant data to fit the columns we have trained our model on.
- We run the retrieved data through a similar processing pipeline that was used to train the model in order to generate a data frame suitable for model inference.
- We load our trained Random Forest model (exported via our overall pipeline as a .joblib.pkl file) on the server.
- Our model generates the prediction based on the data provided which we then showcase to the user along with other relevant video information.



Enter a youtube link or a youtube video ID to predict if the video is "Good" or "Bad" in nature. To understand how we are making the prediction, navigate to the project explained or github repository.

Future Direction

In the process of training our model, it was interesting to observe that the transformer on numeric data input as text performed as well as it did. Given this information, we believe an area for future research is utilizing the full text of comments/descriptions and more tuning of the deep learning model as well as using other larger pre-trained models to examine its performance on similar data. Our model was trained on English videos. While we do believe our model will still perform reasonably well due to the impact of other features, we can't guarantee any accuracy metrics on non-English videos, primarily due to current limitations in the sentiment analysis library used. We recognize this may introduce bias into the model and hopefully can improve it in the future. Other options for improving our model include exploring new features such as subscriber count and other related video metrics as well as actual video content using CNNs (Convolutional Neural Network) or ViTs (Visual Transformers) to better understand the video. Transcript information from videos could also prove to be a valuable source for our model using transformers. Lastly, we would consider allowing users to confirm or deny if the model's prediction is correct in our future research. Even though this may introduce some user bias, ultimately, it would extend our data and train the model again for better possible results. If you are interested in viewing more details regarding our project, please refer to our GitHub repository!

Conclusion

One of the most significant benefits of the dislike button on YouTube was the ability for an individual to assess the potential quality of a video prior to watching it. Removing the dislike button was YouTube's attempt to promote respectful interactions between viewers and creators, but viewers now lose out on a viable metric for ascertaining video quality. After retrieving and processing archived data from a repository of YouTube videos and supplementing it with web scraped comment data for each video, we successfully trained a Random Forest Model. The performance of our model, in terms of predicting whether a video can be categorized as "Good", "Neutral", or "Bad", produced results including an accuracy score of 82% and an F1 score of 80%.

As our final deliverable for this project, we have developed a web application that embodies the spirit of our project, saving the dislikes. While dislike counts may never make their return, our app attempts to satisfy a viewer's desire to gauge whether or not a video is worth their time to view. Leveraging the YouTube API to download the necessary data to use our trained Random Forest model, the web app takes in either a YouTube Video ID or link, retrieves the desired features, downloads top comment data, and then makes a prediction based on all the data that has been gathered on whether the video is Good, Neutral or Bad. Try out our web application for yourself at SaveTheDislikes.com!

Statement of Work

James Mete focused on the Data Acquisition and pipeline creation, Feature Selection and Processing, as well as Machine Learning Model Training & Testing

Jenna Mekled focused on Feature Engineering & Selection and Preparing Data for Machine Learning.

Sashaank Sekar focused on Downloading Comment Data and the development of the Web Application

All members contributed visuals in their respective sections.

Special thanks to Michelle LeBlanc for contributing so much of her time to us providing feedback and guidance throughout this project.

We would also like to thank the other instructors and teaching team members at the University of Michigan for their guidance along the way. Go Blue!