YouTube Comments Classification

Anshul Tomar, Govind Mali , Hardik Jatolia

Abstract : *This paper reports our experience with building a YouTube Spam comment classifier. We have a dataset provided on Kaggle. This dataset contains videos from 4 different YouTubers and all the comments made on those videos.YouTubers in this dataset include -Cleo Abram,Physics Girl,Jet Lag: The Game,Neo. Primary objective of this dataset is to cluster the comments to identify a cluster that contains all the spam comments and fix the issue once and for all.*

Introduction

The rise of social media platforms such as YouTube has made it easier for people to connect and share information. However, as these platforms have grown in popularity, so has the prevalence of spam and fake content. Spam on YouTube can take many forms, such as misleading video titles, irrelevant comments, or links to phishing sites. This not only detracts from the user experience, but can also be harmful by spreading misinformation, scams, and malware.

Our Project Aim to combat this issue and develop machine learning models to classify and filter out spam on YouTube. In this report, we will be discussing our own project on YouTube spam classification, which involved building a machine learning model to identify and filter out spam comments on YouTube videos.

Dataset:

The Dataset is from “YT_Videos_Comments.csv” file.

The Dataset Contains 861962 Rows and 9 Columns. The Dataset has user,video title, video description, video id, comment(displayed), comment(actual) , comment (Author) , channel id and etc.

To achieve the aforementioned, the following workflow is performed:

1. **Data Import**: Data is the lifeblood of predictive analytics. We have to know which data to use, where to gather them, and how to make them useful to solve our problem.
2. **Data Cleaning**: Raw data are generally incomplete, inconsistent, and contain many errors. Thus, we need to prepare the data for further processing. We structure and enrich data into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes, such as analytics.
3. **Feature Extraction**: We perform feature extraction to transform the original data to a data set with a reduced number of variables, which contains the most discriminatory information.
4. **Data Modelling**: There is no machine learning algorithm that works best in all situations. So, the best approach is to generate a high performance model is by trying multiple algorithms, optimizing them, and comparing them for your specific scenario.

Data modelling involves selecting and optiming the machine learning models that generate the best predictive performance based on the data we have.

5. **Prediction**: Once we have developed the best predictive model, we can deploy it to make predictions.

Methodology

Pre-Processing:

- The first step in preprocessing the dataset was to expand any contractions present in the text. Contractions are shortened versions of words or phrases where one or more letters have been removed and replaced with an apostrophe, such as "can't" for "cannot" or "you're" for "you are".
- We then cleaned the text of each comment by removing any URLs, punctuation marks, special characters and emojis. We also converted all text to lowercase to ensure consistency. This step reduced the number of unique words in the dataset.
- After this we did feature extraction , We did tokenization,it is the process where we convert sentences into list of words here words are features and after that we removed stop words like : “The”,”And” , “Have” with this we left with a list of important words which may be more important for our clustering purpose.
- After this we performed lemmatization, it helped us to reduce the dimensionality of text data and improve the accuracy of downstream text analysis tasks.

Visualization:

In 2nd Step we analysed the text using various functions. Text analysis is a crucial part of natural language processing and can provide valuable insights into the structure and content of textual data.

One common technique for text analysis is to extract n-grams, which are contiguous sequences of n words. In this report, we present a Python code that extracts the top-n bigrams from a given corpus of text data.

We used a Python function called "get_top_n_bigram" that takes two input parameters: a corpus of text data and the value of n, which specifies the number of top bigrams to extract. The function uses the CountVectorizer class from the scikit-learn library to transform the text data into a bag-of-words representation and extract bigrams with a given value of k. The stop_words parameter is set to 'english' to remove common English stop words from the bag-of-words representation. The function then calculates the frequency of each bigram in the corpus and returns a list of the top-n bigrams in descending order of frequency.

Extracting the Top 20 Words

The first part of the code calls a function called get_top_n_bigram, which extracts the top 20 most commonly used words from the 'Processed Comment' column of a pandas dataframe called df1. The function has three parameters:

1. text_list: The column of the data frame that contains the text data.
2. n: The number of most common words to extract.
3. ngram_range: The ngram range to be considered for the analysis. In this case, it is set to 1, which means that individual words are considered.

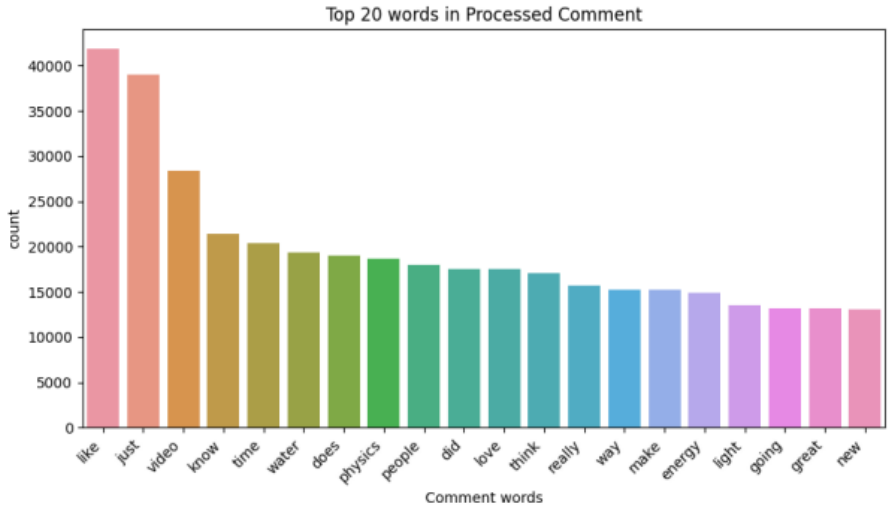
The function returns a list of tuples containing the top 20 words and their frequency of occurrence.

Converting to Pandas Data frame

the list of tuples returned by get_top_n_bigram is converted to a Pandas dataframe called df2. The dataframe has two columns, 'Comment words' and 'count'. The 'Comment words' column contains the individual words, and the 'count' column contains the number of occurrences of each word.

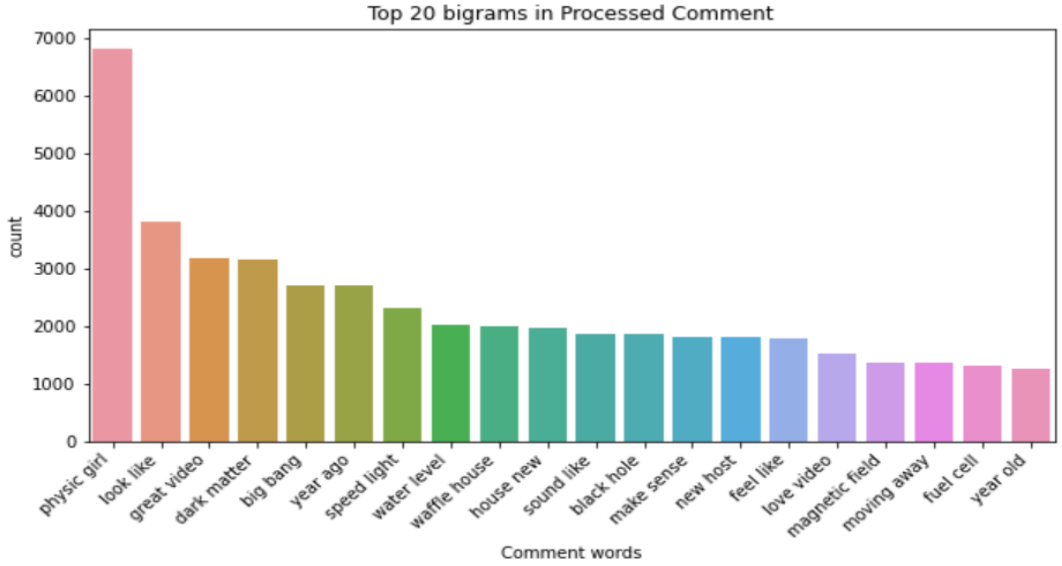
Creating a Bar Plot

A bar plot was created using the Seaborn library. The sns.barplot function is used to create the plot, with the 'Comment words' column on the x-axis and the 'count' column on the y-axis.



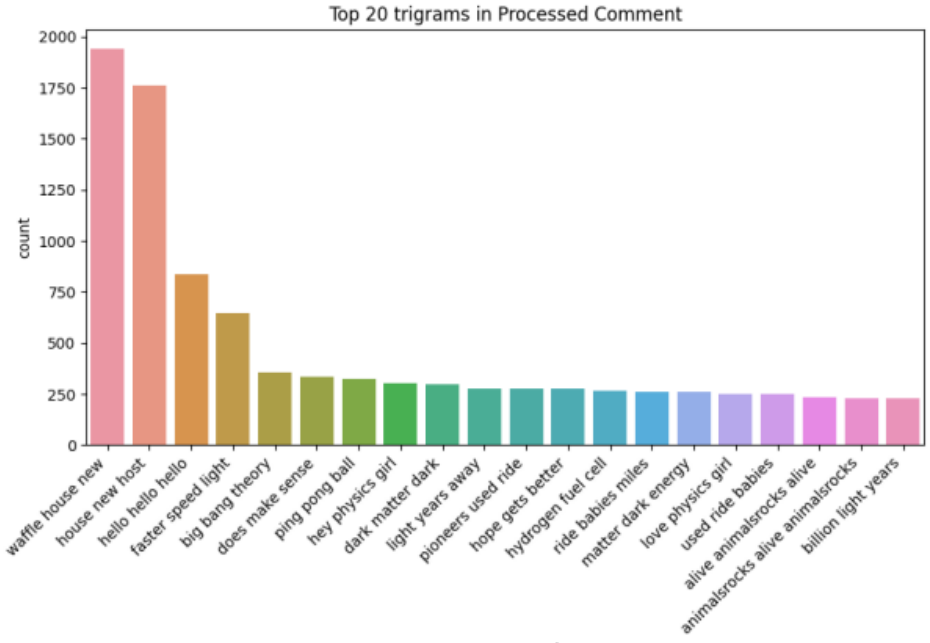
Extracting the top 20 bigrams from the `Processed Comment`

We extracted the top 20 bigrams from the Processed Comment column of a Pandas data frame df1, created a Pandas data frame of these bigrams with their frequency counts, and then visualized the top 20 bigrams using a bar plot with Seaborn.



Extracting the top 20 Trigrams from the `Processed Comment`

We extracted the top 20 Trigrams from the Processed Comment column of a Pandas data frame df1, created a Pandas data frame of these Trigrams with their frequency counts, and then visualize the top 20 bigrams using a bar plot with Seaborn.



Implementation:

Word Cloud generation

First of all we loaded an image file for the word cloud mask. we used the image file named youtube.jpg. The image is opened using the PIL (Python Imaging Library) and converted into a numpy array using the np.array() function.

Then we created a set of stop words using the STOPWORDS variable from the wordcloud module. These are words that are common in the English language but are usually not useful for analysis, such as "the", "and", "a", etc.

After that the processed comments from the data frame are joined together using the join() function and assigned to the comment_text variable.

After that, a WordCloud object is created with the following parameters:

- height and width: the height and width of the word cloud image in pixels.
- background_color: the background color of the word cloud image.
- mode: the color mode of the word cloud image, which is set to "RGBA" for transparency.
- stopwords: the set of stop words created earlier.
- mask: the numpy array of the image file used as the word cloud mask.

We called generate() on the WordCloud object with the comment_text variable as the argument. This generates the word cloud image.

To add colors to the word cloud, the ImageColorGenerator() function from the wordcloud module is used to generate a coloring from the image mask. This coloring is then applied to the word cloud image using the recolor() method.

Finally, the word cloud image is displayed using plt.imshow(), and the image is saved as a file using plt.savefig().

Vectorizing The Dataset

TF-IDF VECTORIZATION (Term Frequency –Inverse Document Frequency)

We are Going to vectorize the lemmatized comments from the dataset.

The TF-IDF value increases proportionally to the number of times a word appears in a document but is offset by the frequency of the word in the corpus.

We create an empty pandas dataframe called "Random_df" and then fill it with a random sample of 5000 rows from another pandas dataframe called "df". The purpose of this code was to create a smaller subset of the original data for quicker analysis. The "sample" function from pandas is used to select a random subset of rows from "df".

After that we created an instance of the TfidfVectorizer class from scikit-Learn's feature extraction module. This class is used to convert text documents into a matrix of TF-IDF

features. The "max_features" parameter is set to 10000, which means that the vectorizer will only consider the top 10000 most frequent words in the dataset.

After this we applied the vectorizer to the "Lemmatized Comment" column of "Random_df" using the "transform" method. Because of This the text in each comment was converted into a sparse matrix representation of its TF-IDF features.

we displayed the first few comments from the "Lemmatized Comment" column of "Random_df". This was done for verifying that the text data was preprocessed correctly before applying the vectorizer.

We calculated and printed the sparsity of the TF-IDF matrix. Sparsity refers to the percentage of zero elements in a matrix.

Reducing the dimensionality of a sparse matrix of TF-IDF features.

The TruncatedSVD class from scikit-learn's decomposition module was used to perform a linear dimensionality reduction on a matrix using truncated SVD. The "n_components" parameter was set to 1000, which means that the dimensionality of the matrix will be reduced to 1000 dimensions.

Then we applied the SVD to the TF-IDF matrix using the "fit_transform" method. This applies the SVD algorithm to the matrix and returns a new matrix that has been transformed into the reduced dimensional space.

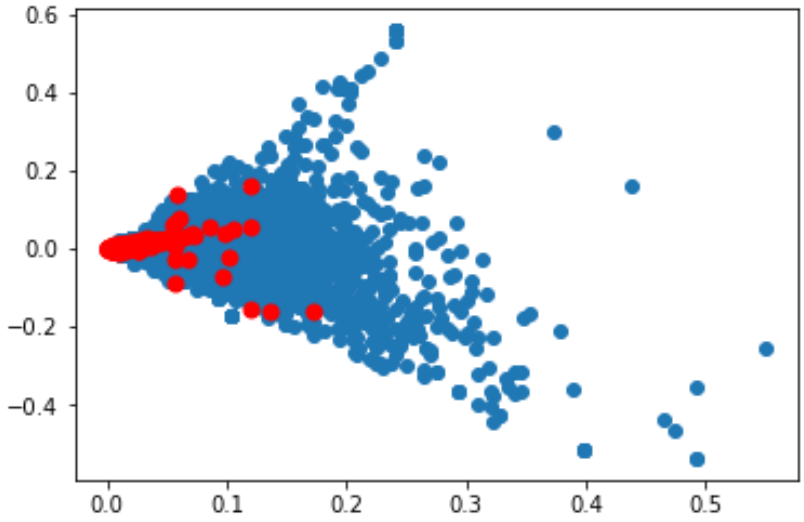
The shape of the reduced data was determined . We did this for verifying that the dimensionality reduction was performed correctly and that the resulting matrix has the expected shape.

computing the inverse covariance matrix of the TF-IDF matrix after applying SVD. The inverse covariance matrix is a measure of how the variables in the data are related to each other.

After we calculated the Mahalanobis distances for each data point. The Mahalanobis distance is a useful way to identify outliers in a dataset because it takes into account the correlation between variables, which is important for text data that has a large number of dimensions.

Then applied Truncated SVD to the TF-IDF matrix to reduce its dimensionality to two dimensions. This is done so that the data points can be plotted in a two-dimensional space.

creating a scatter plot of the data points in two dimensions and highlight the outliers in red.



We displayed the Outliers data from the original dataframe

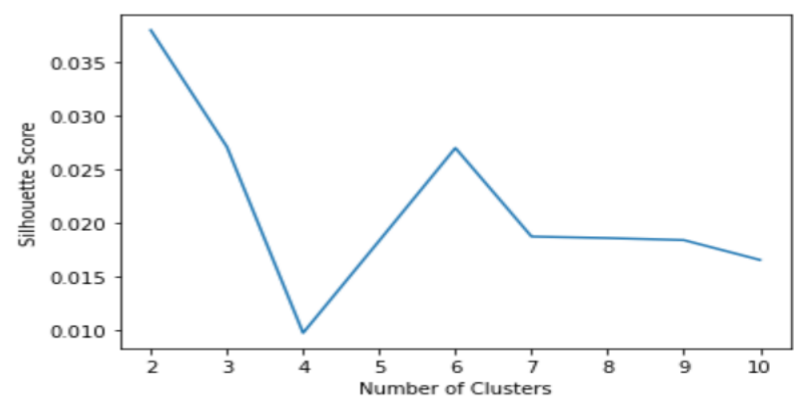
	Comment (Actual)	Lemmatized Comment
159988	Wheres the scientists third cousin Chuck?	scientist third cousin chuck
736496	Vortex	vortex
460217	Do you think there is a connection to the end of the mayan calendar and the discovery of the Hig...	think connection end mayan calendar discovery higgs boson
854055	you're the smartest blonde I know!\n(except for Isaac newton my English teacher and other blond ...	smartest blonde know except isaac newton english teacher blond physicist
789032	8:20 This is the secret he told Antwuan Dixon. We've all waited to hear this for years.	secret told antwuan dixon waited hear year
...	...	...
814776	Is that william osman	william osman
462067	I never heard blah blah blah in the quadratic formula song, you must be singing a remix	never heard blah blah blah quadratic formula song must singing remix
822805	It's high noon	high noon
113701	Do it out of Spite, Deanna	spite deanna
860251	And thanks to Dirk Maller of Versassium for filming that for me. ;)	thanks dirk maller versassium filming

250 rows × 2 columns

Finding the No of Optimal clusters

defined a range of values for the number of clusters to be used in the K-means algorithm.

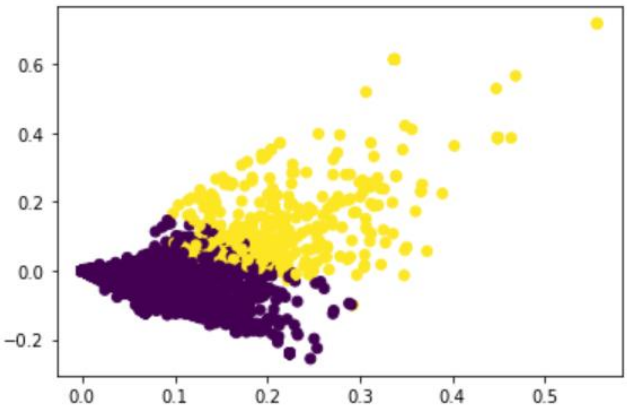
We provided a range of 2 –10 clusters . We calculated the silhouette score for finding the optimal clusters. After this created a line plot of the silhouette scores as a function of the number of clusters, using Matplotlib.




```
The optimal number of clusters is: 2
```

K-Means

We applied Kmeans with no of clusters =2 on tvidf_svd whose dimenison was reduced to 1000. And we stored the clustering labels in clusters assignment.



After analysing manually by seeing the comments, we concluded that yellow(label = 1) ones are spam and other are not spam

Majority Voting

We trained the models KMeans, AgglomerativeClustering, SpectralClustering on tfidf_svd and stored the corresponding cluster labels in a list, then we performed majority voting to find the new final labels named y-pred for the clusters.

Evaluation Metrics

We used Silhouette score, Calinski-Harabasz , Davies-Bouldin to evaluate our clustering models and got the following results :

```
                    Model 1: KMeans
-------------------------------------------------
              Silhouette score: 0.038
          Calinski-Harabasz score: 36.106
            Davies-Bouldin score: 6.050


          Model 2: AgglomerativeClustering
-------------------------------------------------
              Silhouette score: 0.106
          Calinski-Harabasz score: 37.046
            Davies-Bouldin score: 0.991


            Model 3: SpectralClustering
-------------------------------------------------
              Silhouette score: 0.106
          Calinski-Harabasz score: 37.046
            Davies-Bouldin score: 0.991
```

We found that Silhouette score for AgglomerativeClustering is maximum.

Training Decision Tree Model:

Using tfidf as training data and y_pred from majority voting as y_train we trained the decision tree classifier model to predict the labels for the whole dataset.

Then we predicted final labels for our remaining data and then printed them and analysed them manually to which we got spam comments

	Comment (Actual)	Lemmatized Comment	predicted labels
4835	Something tells me this host also loves microc...	something tell host also love microchip cashle...	1
5180	I love this channel. it's simple, non imposing...	love channel simple non imposing host sound li...	1
5287	The Waffle House has found it's new host	waffle house found new host	1
5441	The waffle house has found its new host	waffle house found new host	1
5484	The waffle House has found its new host	waffle house found new host	1
...	...	...	...
839303	Amazing channel, impressive content and a real...	amazing channel impressive content really love...	1
840153	Ashley could host her own channel, Physics Gir...	ashley could host channel physic girl	1
846978	It's to bad that a potentially interesting vid...	bad potentially interesting video ruined extre...	1
850610	Love Physics Girl but with my strong dislike f...	love physic girl strong dislike bill nye tv sh...	1
853344	I wish people would stop scripting narration l...	wish people would stop scripting narration lik...	1

The comment highlighted has been repeated many times in data, we checked it manually so it shows that cluster 1 is the spam comments

The final number of clusters we got are :

```
0    377328
1      1704
Name: predicted labels, dtype: int64
```

According to our model predictions we can say that around 0.5 % of the comments are spam.