PRML MINOR PROJECT REPORT

Problem Statement:

Given dataset contains videos from 4 different YouTubers and all the comments made on those videos. The 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.

Dataset: Youtube Comment Classification

Problem Approach / Idea of solving problem:

1. Our primary goal is to make cluster of the comments and finding the spam clusters that containing all spam comments.
2. First remove the stopping word (like is ,a the ,…..) using the NLTK Library.
3. The dataset contains null values. So to handle the null values we are filling null values by mode of the data.
4. Droping the unnecessary columns like video id, channel id …. Becaues they are not help us to decide the whether the given comment is spam or not.
5. We have to vectorize the data set. Here we are using the TfidfVectorizer() .Becaues it give count of the words along with the weightage of the words.
6. TfidfVectorizer() given high dimension matrix. So we have to reduce it. Here we can use the PCA. But PCA is not work efficiently because the of TfidfVectorizer gives spare matrix. It difficult to PCA() to handle that.So we have used TruncatedSVD .
7. We classifying the spam comments by comments by three method
   1. By authors counts
   2. Promotional links
   3. By KMean() clustering (we are coning above two method in this also)
8. Finally plotting the word cloud of the spam comment and non-spam comments.

Implementation:

Loading and cleaning the data:

- First mount the Google drive to the colab file and load the dataset using pandas.
- Handle the null values.
- Next, we will drop unrequired columns from the dataset.

```
df=df.drop(["User","Video ID" ,"Comment Time","Comment Author Channel ID"]
,axis='columns')
```

- Convert the data type to string using the ‘astype’ method in pandas. Because it may contains some int data type. to handle that we have to convert them into str.
- By converting the data type of these columns to string, we can ensure that any data in these columns is treated as text data rather than numeric or other data types.

Next goal is to remove all stop words from the comment in order to make it easier to analyze later.

Now we must apply a filter to some text data that we have in a DataFrame. First, we define a set of stop words using the stopwords module in NLTK. These are common words in the English language that don't add much meaning to a sentence, such as "the", "and", "of", etc.

- we define a function called filter that takes a string of text as its input. This function splits the text into a list of words, and then uses a list comprehension to remove any words that appear in our set of stop words.
- We convert all words to lowercase to make sure we catch both capitalized and uncapitalized stop words.
- Finally, we apply this filter function to the dataset.

```
# removing the stopping word
```

```
stop_words = set(stopwords.words('english'))

def filter(text):

    words = text.split()
    words = [word for word in words if word.lower() not in stop_words]
    text = ' '.join(words)
    return text


df['Comment (Displayed)'] = df['Comment (Displayed)'].apply(filter)
df['Comment (Actual)'] = df['Comment (Actual)'].apply(filter)
df['Video Description'] = df['Video Description'].apply(filter)
df['Video Title'] = df['Video Title'].apply(filter)
```

- Now, we have to vectorize the text data in a DataFrame column using the TF-IDF method.

Count Vectorizer give number of frequency with respect to index of vocabulary where as *tf-idf* consider overall documents of weight of words.

Reducing the dimension:

Here we are using the TruncatedSVD because TruncatedSVD accepts sparse matrix input whereas PCA does not. We ahev a spare data . Therefore, TruncatedSVD would be a better choice.

Next code performs dimensionality reduction on a sparse matrix using TruncatedSVD, which is a useful technique for reducing memory usage and computational complexity in machine learning applications.

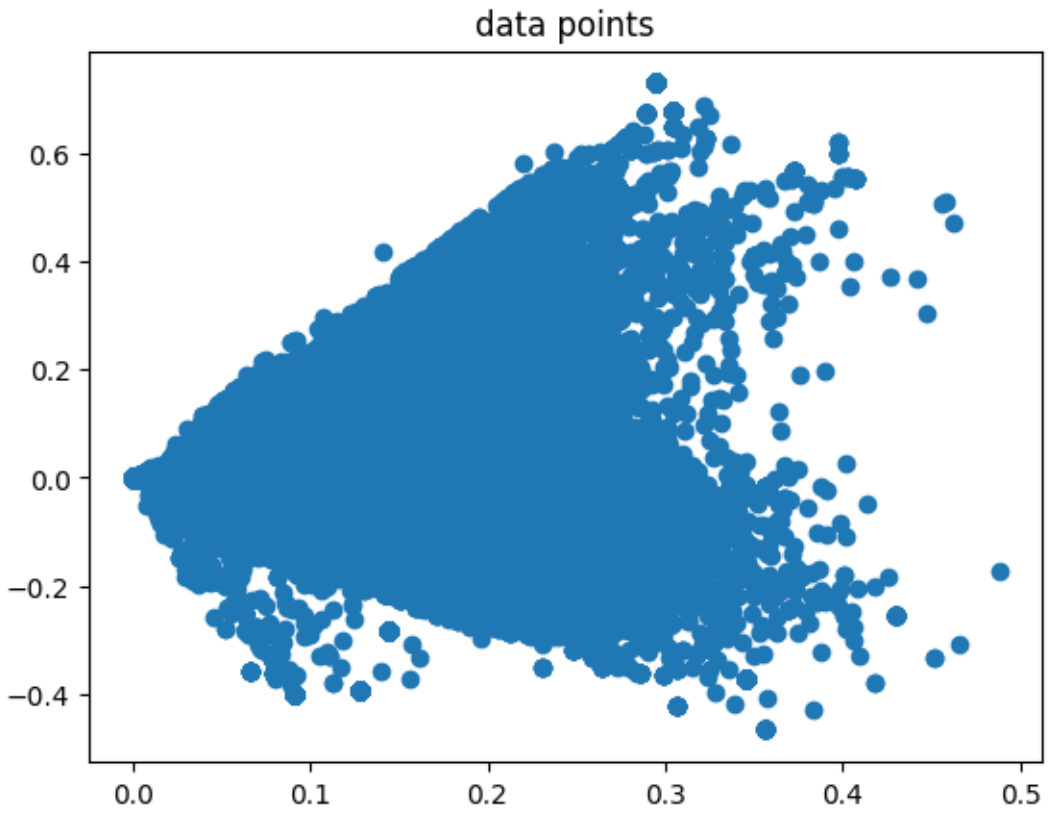
```
#reducing the dimension of the spare matrix

tsvd = TruncatedSVD(n_components=2)
X_sparse_tsvd = tsvd.fit(tfidf_matrix).transform(tfidf_matrix)
```

Plotting the datapoins:

Below is the scatter plot of a lower-dimensional representation of our data.

We are using the truncated singular value decomposition (TSVD) algorithm to reduce the dimensionality of our data and then plotting the first two principal components.

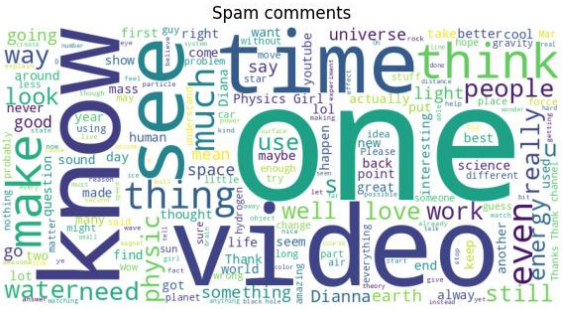


Clustering the comments based on Authors:

- We count the number of comments made by each author and store it in the author counts variable.
- Next, we create a list of spam authors by selecting those whose comment count is greater than 15. To identify which comments are spam, we add a new column to the Data Frame called "by Author".

- After separating the comments into spam and non-spam categories, we create two word clouds using the WordCloud module. These word clouds are a visual representation of the most frequently used words in each category of comments. The generate method is used to create the word clouds by joining all the comments together in each category.

Finally, we plot the word clouds side-by-side using the subplots and imshow methods from the matplotlib module.

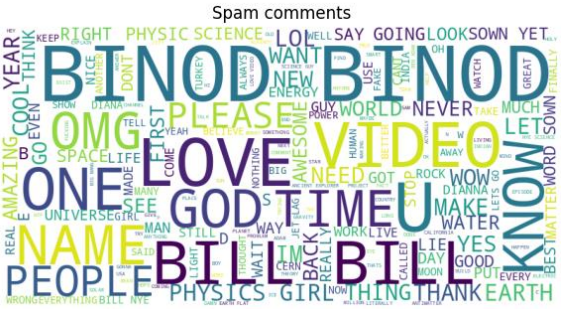


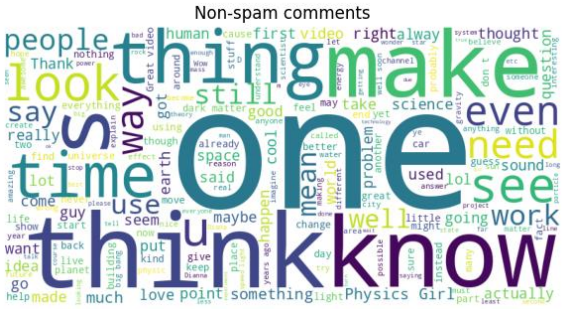
Clustering the comments on multiple links and promotional ones:

- We did this to analyze and filter out spam comments from user-generated content, such as comments on a blog post or social media platform.
- We defined a function named is_spam is defined to check if a comment is a spam or not based on certain conditions. These conditions include checking for promotional content, excessive use of capital letters, and the presence of common spam words or phrases. If a comment meets any of these conditions, the function returns 1 indicating that the comment is spam. Otherwise, it returns 0 indicating that the comment is not spam.
- Next, the function is applied to each comment in the dataset using the apply method and a new column named conditions is added to the dataframe containing the results of the spam check.

- After that, created two dataframes - one containing spam comments and another containing non-spam comments - by filtering the original dataset based on the values in the condition's column.

Finally, we generated word clouds for both the spam and non-spam comments using the WordCloud library and visualizes them using the matplotlib library. The resulting word clouds provide a quick and intuitive way to visually identify the most common words and phrases in the spam and non-spam comments, respectively.



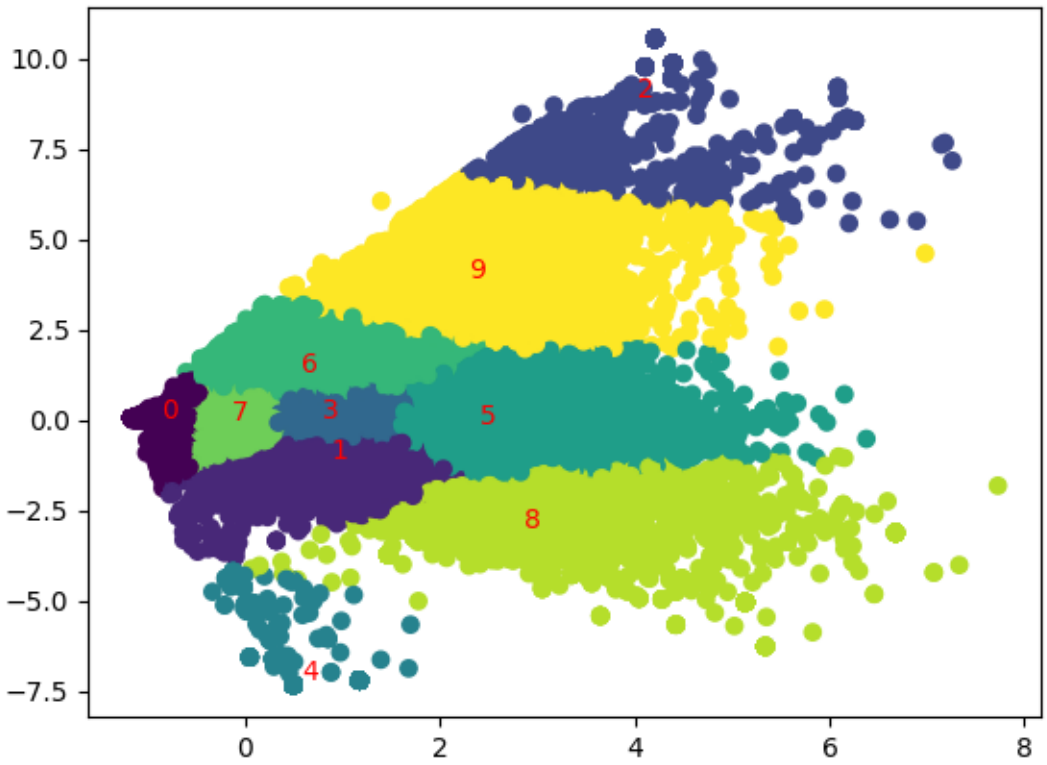


Performing KMeans:

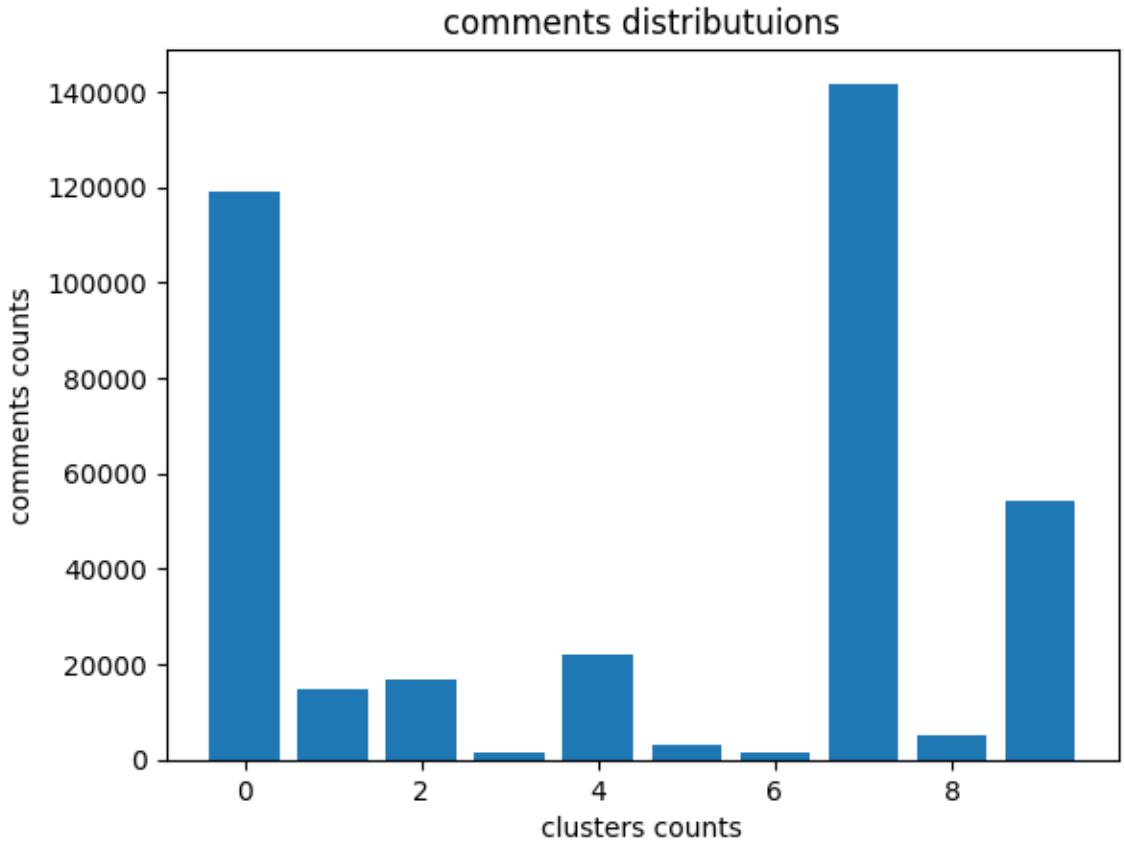
- First, we're normalizing some data using a StandardScaler, which is a common preprocessing step for clustering algorithms.
- We're then using the KMeans algorithm with 10 clusters and a random state of 42 to perform the clustering.
- We're also creating a scatter plot of the data with different colors representing the different clusters, and labeling the centroids of each cluster with their corresponding cluster number.

Spam comments

Non-spam comments



- Next, we're creating a bar chart to show the distribution of comments across each of the 10 clusters



Declaration of spam clusters:

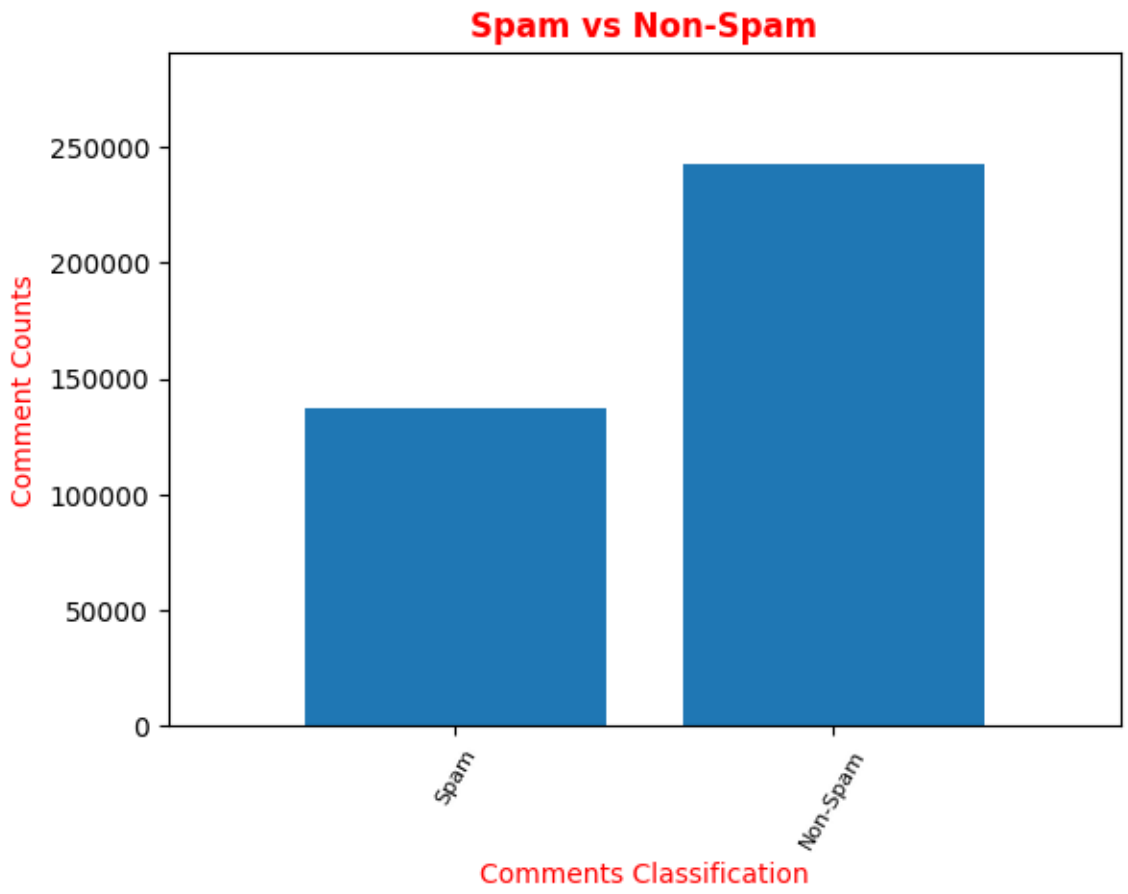
- By observing comments(manually and applying the above two conditions and merging the all the result according to # spam and non-spam).
- After that we can see the comments in cluster 7 and 6 contains so much spam comments compare to others
- Separate that spam and non-spam comments .
- We then use these cluster assignments to create two word clouds, one for the spam comments and one for the non-spam comments.
- We visualize these word clouds side-by-side using a subplot.

- Finally, we evaluate the model using the silhouette score, which is a metric for evaluating the quality of a clustering algorithm's output.

```
The silhouette score is: 0.4739029583073459
```

Conclusions:

1. Data contains almost around 137367 spam comments.
2. Data contains almost around 242161 non- spam comments.
3. The silhouette score of a clustering algorithm is 0.474
4. The percentage of the spam comments in the given data set is around 35.3%
5. The percentage of the non-spam comments in the given data set is around 64.7%



Team Members:

1. Bikas Singh
2. Naman Gurawa
3. Mukesh