Detection of YouTube Spam Comments

Problem Statement

This project addresses the problem of detecting spam comments on YouTube, one of the world's most popular video-sharing platforms. YouTube enables dynamic interaction through its comment section, allowing viewers to share opinions and connect with content. However, this creates an opportunity for spam comments—unsolicited messages designed to gather personal information, promote fraudulent activities, or mislead viewers away from the platform (Google, n.d.). The presence of spam comments diminishes the user experience and can create a potentially hostile environment. Therefore, accurately detecting and preventing such comments is essential to not only enhancing user satisfaction, but protecting users from malicious content.

To combat this issue, the project aims to apply machine learning classification techniques to automate the detection of spam comments. The dataset consists of 1369 labeled YouTube comments which provides the foundation for training our model. Using a Random Forest model and feature engineering, we created an algorithm to classify comments as spam or not spam.

Statistical Analyses

Data Processing

The Youtube spam comment data set consists of 1369 rows and 6 columns. The variables include COMMENT_ID, a unique identifier for each comment; AUTHOR, the username of the individual who posted the comment; and DATE, the time stamp said comment was posted, which is precise up to the second. Additionally, the CONTENT column contains the actual comments posted, whereas VIDEO_NAME, includes the title of the video the "Author" commented on. The last Variable included is CLASS, a binary classifier which classifies a comment as spam (1) or not (0); this variable was not used in training but rather in testing. Prior to processing the data, these attributes provide essential descriptive information for the analysis.

To clean the data, we checked for duplicate values and missing data. No duplicate values were found, and the only missing data was in the DATE variable. Upon further review, The DATE and COMMENT_ID variables were deemed logically irrelevant in spam classification and thus were excluded from feature selection to preserve computational efficiency. As a result, the missing values in this variable were disregarded and no data cleaning was deemed necessary.

Additionally, to transform the CONTENT variable into a readable form, the primary text data was vectorized using the TF-IDF (Term Frequency-Inverse Document Frequency) approach to transform the comment content into numerical features suitable for model training.

Feature Engineering and Selection

To enhance the predictive power of our model, feature engineering was employed. Exploratory analysis was conducted to justify the inclusion of any additional predictors; this exploration was guided by our intuition and prior knowledge of spam comments. Intuitively, spam comments can have some commonly used words and punctuations. Spam comments may also disproportionately contain URLs and emojis compared with the non-spam comments. Ultimately the following variables were created: Contains_url, Contains_emo ,Contains_punc, Contains_words.

The Contains_url variable is a binary variable denoting the presence of a URL. This was prompted by the overwhelming presence of URLs in spam comments. Our exploration found that approximately 95% of spam comments contained URLs, highlighting the relevance of this variable.

The Contains_emo variable similarly classifies a comment, flagging comments that contain emojis. Our analysis found that only 36 comments contained emojis of which, 30.6% were classified as spam. This variable seemed less compelling than the previous in its predictive power however, it was included for variable selection to determine the significance.

The Contains_punc variable was implemented to identify specific punctuation patterns frequently associated with spam. The top 5 punctuations occurring in spam comments were determined and considered:

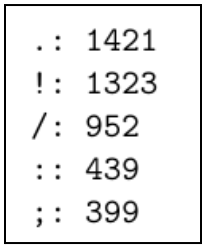

Figure 1. Top 5 punctuation occurring in spam comments.

Analysis revealed that the two most common punctuations found were the period "." and the explanation mark "!". The other three were also common and can likely be attributed to their prevalence in URLs.

Lastly, the Contains_words variable was created to detect common spam related words. We obtained a list of the five most common words. Lemmatization was used to return the words into their root form, for example "checking" and "checked" would be reduced to "check" and thus would be counted as the same word. The five most common words were found to be:

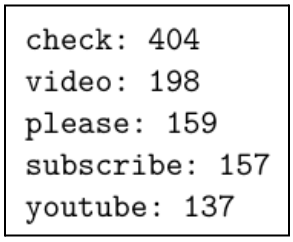

Figure 2. Top 5 words occurring in spam comments.

The original variables were also subject to exploratory analysis. The VIDEO_NAME was analyzed to see if the distribution of spam comments changed by video. As seen in *Figure 3*, the distribution remains relatively constant between videos, thus this variable will not be considered in subset selection.

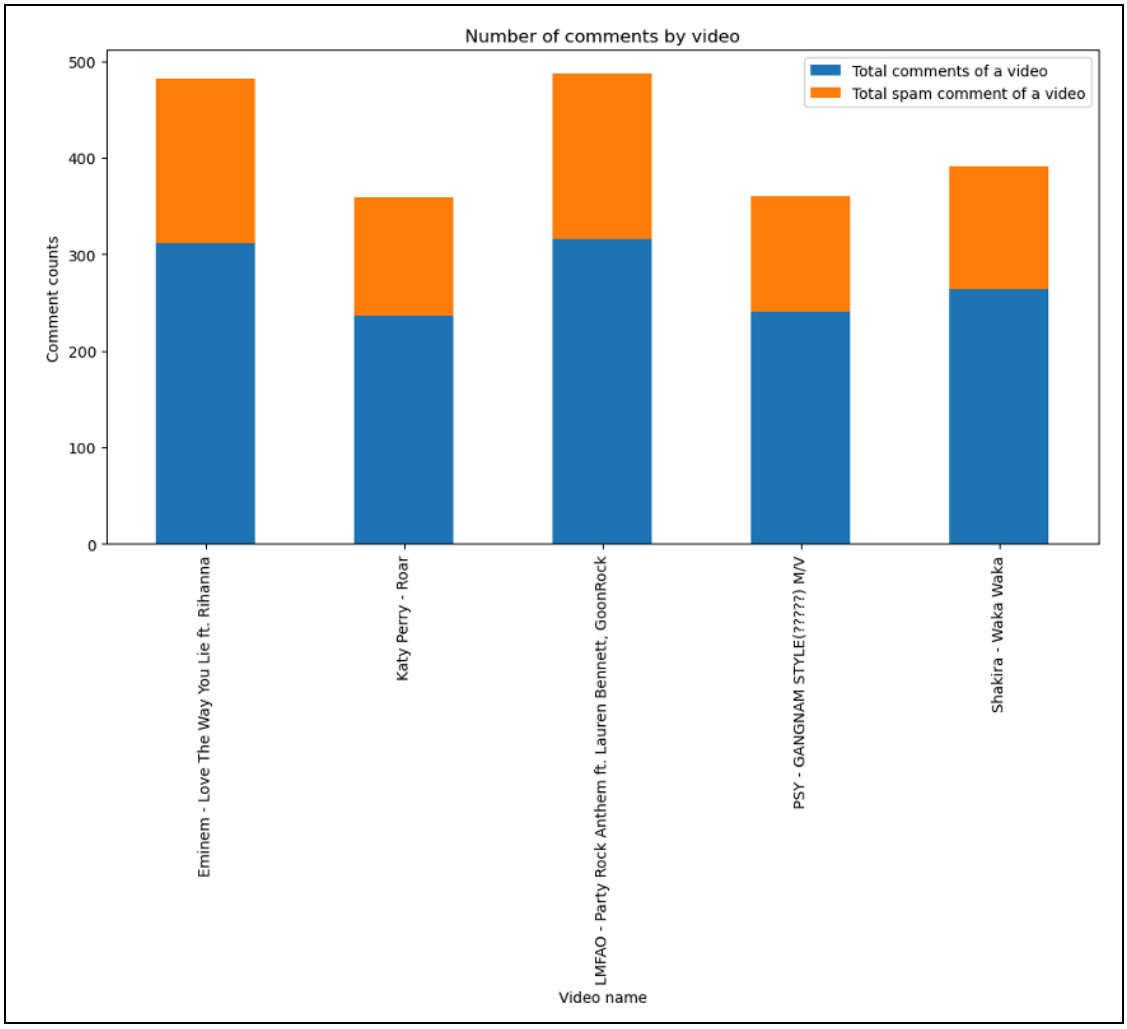

Figure 3. Distribution of spam comments between videos.

The AUTHOR variable was also analyzed; although there was a correlation between the five most avid commenters and spam comments, there was not enough data to suggest statistical significance; adding this variable consistently decreased performance thus it was not included in subset selection.

With CONTENT as the only original variables and the additional four engineered features, five features were considered for feature selection. The forward stepwise selection algorithm was used to perform feature selection. This is an iterative, greedy approach to identify the best performing predictors for a given model. We started with a simple model, using only one predictor, and iterated over all possible features, adding each remaining feature one at a time, each time choosing the feature that resulted in the highest validation approach testing classification accuracy. These features were then added to the model and removed from the unused features. This process was repeated until a full model was created (with all five features). Then, the five models were compared using their respected testing classification accuracy; the model that produced the highest value was chosen.

Our algorithm selected CONTENT to be the predictor in the model. The validation set approach was used for explicit feature selection where the dataset was split into training (70%) and testing (30%) subsets.

The model was trained on the training data and evaluated on the testing data. It achieved an accuracy of 96%, demonstrating its effectiveness in identifying spam comments

As stated before, this is a greedy approach, meaning it does not account for all possible combinations of variables and may not choose the best subset. However, using this method means sacrificing a degree of accuracy for computational efficiency. It should be noted that our algorithm only considers:

$$\frac{p\times(p+1)}{2} = \frac{5\times(5+1)}{2} = 15\ models$$

As opposed to best subset selection which would consider:

$$2^p = 2^5 = 32\ models$$

Best subset selection would more than double the amount of models to consider; this disparity shows the computational benefit from performing forward selection.

Model Selection

We opted to use Random Forest classification as our final model. We justified using a decision tree based approach so that it would be easily interpretable, and to minimize data preprocessing. Due to the high variance and lower accuracy of a simple regression tree, we opted for the more robust random forest model, suitable for high dimensional datasets.

Random Forest is an ensemble learning method, it uses multiple decision trees (called a forest) as a means to mitigate overfitting and improve predictive accuracy; our forests comprised of 100 decision trees each. Each tree in the forest is trained by bootstrapping the data; this is done to reduce the risk of overfitting by incorporating randomness into training. To decorrelate the trees, only a subset of predictors is considered at each node. After each tree in the forest classifies a comment as spam or not spam, the comment is classified by the majority outcome of the forest. Because bootstrapping typically only requires approximately $\frac{2}{3}$ of the data, the data not used in training is called the “out-of-bag sample” and can be used to calculate the out of bag error, providing an internal cross-validation mechanism.

The model was trained with 5 predictors, CONTENT, Contains_url, Contains_emo, Contains_punc and Contains_words. Random Forest provides an estimate of feature importance which decreases the need for supplementary feature selection methods, however, we found that using subset selection improved interoperability, training speed, and model performance. Feature selection using the forward stepwise algorithm identified that CONTENT was the most impactful predictor resulting in a simple model with high accuracy.

Alternative models such as Logistic Regression and Support Vector Machines were considered, however, after implementation it was determined that Random Forest outperformed them in terms of Accuracy and interpretability. We also considered “overloading” the training by creating a voting system between different models, however this created issues with processing speed, model complexity, and interpretability issues. Ultimately we opted for the simpler Random Forest model.

Model Performance Validation

Several key metrics were used to evaluate model performance. First and foremost, model accuracy was considered as the primary metric. As mentioned earlier, the model achieved a validation set accuracy of

96%, reflecting the proportion of correctly classified comments and demonstrating its effectiveness in identifying spam comments.

Precision was also a crucial metric for evaluating the model's performance. Precision measures how many of the predicted positive results are true positives, focusing on the accuracy of the model's positive predictions. The precision scores were 98% for spam (class 1) and 94% for non-spam (class 0), indicating a high level of correctness in the model's positive predictions.

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP)+False\ Positives\ (FP)}$$

Similarly, recall measures how many of the actual positive results were correctly predicted by the model, emphasizing the completeness of positive predictions. Recall scores were 94% for spam and 98% for non-spam, highlighting the model's ability to identify relevant instances effectively.

$$Recal = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP)+False\ Negatives\ (FN)}$$

These results imply that the false positive and false negative rates were low. A confusion matrix analysis further supported this, revealing minimal false positives and false negatives, which emphasized the model's reliability (*Figure 4.)*.

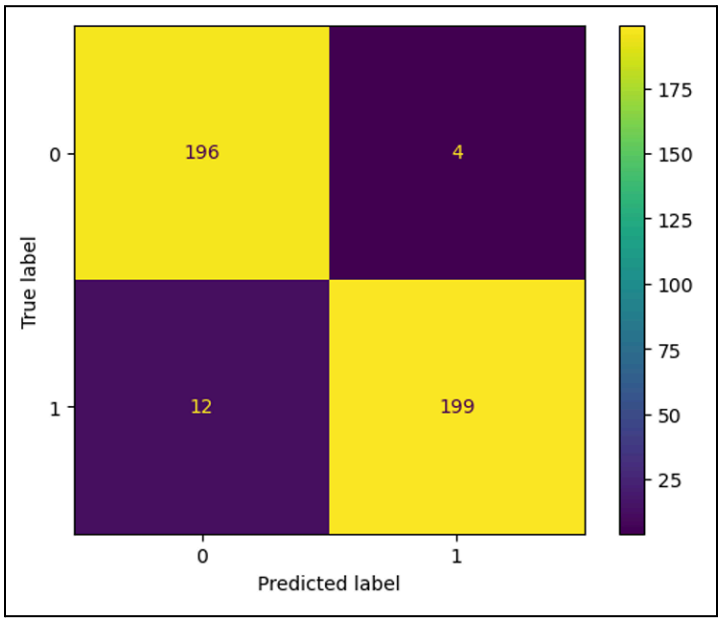

Figure 4. Confusion matrix

We plotted the Sensitivity against the false positive rate to create the AUC-ROC curve. This also illustrated strong performance, with the classifier achieving an AUC of 0.988, a near-perfect separation of spam and non-spam comments (*Figure 5.)*.

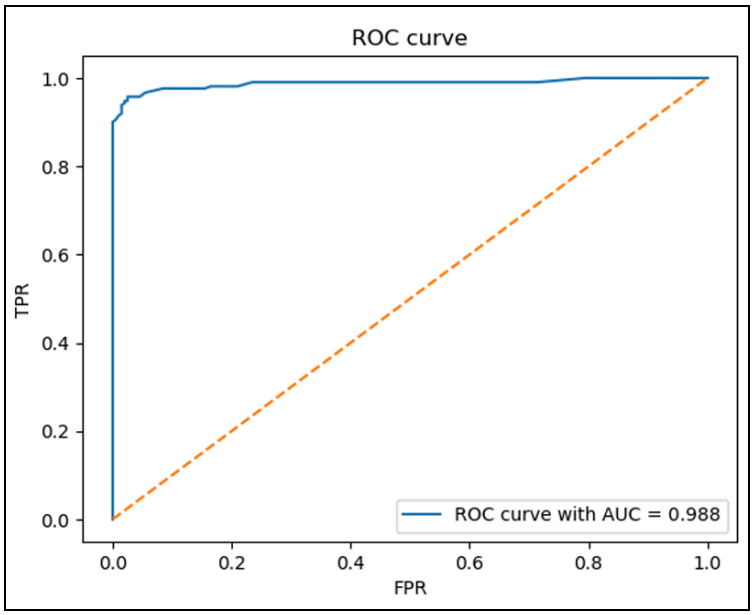

Figure 5. ROC curve.

The last metric used to evaluate model performance is the F1-score. It is a common metric used in binary classification which balances precision and recall. This value was balanced at 96% for both classes. Again, this value is high indicating good model performance.

$$F1 - Score = \frac{2 \times Precision \times Recal}{Precision + Recal}$$

Results and Conclusion

Using Random Forest Machine learning, our model was trained on a dataset containing 1369 labelled YouTube comments. We employed extensive data processing and feature engineering to enhance the predictive power of the model as much as possible. Exploratory analysis yielded the necessary inclusion of key variables such as “Contains_url”, “Contains_emo”, “Contains_punc”, “Contains_words”. These features captured patterns that are more associated with spam comments than with normal ones and complemented the CONTENT variable. Feature selection done via forward selection identified CONTENT as the most impactful predictor.

Our random forest model demonstrated exceptional performance, achieving a 96% accuracy on the testing subset of our training data. Precision and recall scores further highlighted the model’s reliability, with the precision scores being 98% for spam and 94% for non-spam comments, and recall scores being 94% and 98% respectively. These metrics indicate low false positive and false negative rates, which were validated through confusion matrix analysis. The model’s AUC-ROC score of 0.988 reflected a near-perfect separation between the spam and non-spam classes, and the F1 score of 96% for both classes emphasized the balance between precision and recall. The performance metrics are summarized in *Figure 6.*

	precision	recall	f1-score	support
0	0.94	0.98	0.96	200
1	0.98	0.94	0.96	211
accuracy			0.96	411
macro avg	0.96	0.96	0.96	411
weighted avg	0.96	0.96	0.96	411

Figure 6. Summary of performance metrics.

After submitting to Kaggle, our model received a public score of 94.059% on the 52% of testing data that was used to compute an unofficial ranking. We received a 94.718% accuracy on the remaining 48%, putting us in 15th place and giving an overall accuracy of 94.38% on the testing data. This underscores the effectiveness of our model, but also shows that there is evidently room for improvement. Further methods for how we can achieve a higher accuracy are discussed in the following discussion section.

For this project, we successfully developed a robust and reliable solution for spam comment detection on YouTube. Our results validate the efficacy of our random forest model, thus we conclude our model can be used for its intended purpose of spam detection to enhance user satisfaction and protect users from malicious content. Furthermore, it can serve as a foundational framework for future enhancement that can be integrated to enhance the detection strength even more.

Discussion

One major limitation of the project is the limited amount of available data. The data consists of 1,369 labeled comments, a pale comparison to the billions of comments found on YouTube; in the second quarter of 2024, YouTube removed around 1.37 billion comments due to violations of the platform's community guidelines (Statista, n.d.). A larger dataset with more diverse comments would yield more holistic and accurate results and would be more representative of the platform.

Another limitation is the evolving nature of spam comments; spammers frequently adapt techniques to avoid detection. As such, our model may currently suffice however, it will need to be continuously trained to maintain model effectiveness over time.

Because we prioritized time sensitivity, we used forward subset selection as opposed to best subset selection. This selection is a greedy approach meaning there is no guarantee that the best subset of features was selected, thus limiting potential accuracy.

A limitation of the model we chose, Random forest, is that it does not reduce bias. It also runs into issues with complexity and scalability. Lastly, Random Forest typically does not perform well in high dimensions meaning in cases of very high-dimensional data such as text data, irrelevant or sparse features can overwhelm the model, making feature reduction beneficial. It should also be noted that method wise, we were limited by the scope of the course. In the future it would be interesting to try more complex decision tree methods that reduce bias and variance such as implementing a Bayesian Additive Regression Tree (BART) or applying neural networks.

We must also note there are ethical concerns relating to automated comment moderation. It is important that our model does not discriminate against diverse linguistic styles and non-standard language. For future exploration it would be imperative to investigate this effect further.

Finally, future exploration would aim to use more complex models and data to improve accuracy. Beyond technical aspects, integrating a real time spam detection system would be the most practical next step for YouTube. This would require speed optimization for implementation.

Bibliography

Aiyar, S., & Shetty, N. P. (2018). N-gram assisted YouTube spam comment detection. *Procedia Computer Science*, *132*, 174–182. https://doi.org/10.1016/j.procs.2018.05.181

Aggarwal, U. (2020, April 29). *Text Preprocessing | tokenization | cleaning | stemming | stopwords | lemmatization*. YouTube. https://www.youtube.com/watch?v=hhjn4HVEdy0

Aman. (2020, April 21). *Text data cleaning in Python | How to clean text data in python*. YouTube. https://www.youtube.com/watch?v=KhXU7KOxQcg

Corey Schafer. (2014, August 24*). Python Tutorial: Comprehensions - How they work and why you should be using them*. YouTube. https://www.youtube.com/watch?v=3dt4OGnU5sM

Google. *Spam, deceptive practices, & scams policies*. YouTube Help. Retrieved November 20, 2024, from https://support.google.com/youtube/answer/2801973?hl=en

Kaggle. *Detect spam in YouTube comments*. Retrieved November 20, 2024, from https://www.kaggle.com/competitions/detect-spam-youtube-comment

Lazy Programmer Inc. (n.d.). *Python for data science and machine learning bootcamp*. Udemy. https://www.udemy.com/share/105D2K3/

Nisha Arya Ahmed. (2024, November 10). *What is A Confusion Matrix in Machine Learning? The Model Evaluation Tool Explained.* DataCamp. https://www.datacamp.com/tutorial/what-is-a-confusion-matrix-in-machine-learning

Statista. *Number of removed YouTube video comments worldwide as of Q1 2023*. Retrieved November 20, 2024, from https://www.statista.com/statistics/1132989/number-removed-youtube-video-comments-worldwide/

Samsudin, N. M., Mohd Foozy, C. F., Alias, N., Shamala, P., Othman, N. F., & Wan Din, W. I. (2019). YouTube spam detection framework using naïve Bayes and logistic regression. *Indonesian Journal of Electrical Engineering and Computer Science*, *14*(3), 1508. https://doi.org/10.11591/ijeecs.v14.i3.pp1508-1517

Stats Wire. (2022, May 2). *Python Feature Selection: Forward Feature Selection | Feature Selection | Python*. YouTube. https://www.youtube.com/watch?v=POCvLGRLDzM

Stats Wire. (2021, August 4). *Random forest classification | Machine learning | Python*. YouTube. https://www.youtube.com/watch?v=3NdH3egUjpM

Telusko. (2018, July 14). *#22 Python tutorial for beginners | Break continue pass in Python*. YouTube. https://www.youtube.com/watch?v=yCZBnjF4_tU

Vidhi Chugh. (2024, September 10). *AUC and the ROC curve in machine learning*. DataCamp. https://www.datacamp.com/tutorial/auc

Xiao, A. S., & Liang, Q. (2024). Spam detection for YouTube video comments using Machine Learning Approaches. *Machine Learning with Applications*, *16*, 100550. https://doi.org/10.1016/j.mlwa.2024.100550

Exploratory data analysis

Understanding the training data set

```
# Loading the necessary module
import pandas as pd

# Reading the Youtube Spam data
pre_data = pd.read_csv('train.csv')

# Display first 5 rows of the data
print(pre_data.head())
```

```
# Show the names of the columns
colnam = pre_data.columns

for x in colnam:
    print(f'Colume name: {x}')
```

```
# Show the dimensions of the data
pre_data.shape
```

Checking for NAs and duplicates in the data

```
# Count the number of NAs in each columns
print('The number of NAs in the data set categorized by columns are: \n', pre_data.isnull().sum())
```

```
# Count the number of duplicated rows
print('The total number of duplicated rows in the data set is:', pre_data.duplicated().sum())
```

```
# Count the number of unique contributors in the data set
pre_data['AUTHOR'].nunique()
```

Explore AUTHOR colume

```
# Count the number of unique contributors in the data set
pre_data['AUTHOR'].nunique()
```

```
# Loading the necessary module
from collections import Counter

# Count the top 5 most dedicated contributors
top_5_author = Counter(pre_data['AUTHOR']).most_common(5)

for x in top_5_author:
    print(f'{x[0]}: {x[1]}')
```

```
# Loading the necessary module
import matplotlib.pyplot as plt

# Visualize top 5 most common contributors in the data set
name = [x[0] for x in top_5_author]
count = [x[1] for x in top_5_author]

plt.bar(name, count, width = 0.4)

plt.xlabel('Authors')
plt.ylabel('Number of contributions')
plt.title('Top 5 most common contributors')

plt.show()
```

```
# See who are some top spammers in the data set
filtered_data = pre_data[pre_data['CLASS'] == 1]

top_5_spammer = Counter(filtered_data['AUTHOR']).most_common(5)
```

```
for x in top_5_spammer:
    print(f'{x[0]}: {x[1]}')
```

```
# Visualize top 5 spammers in the data set
name = [x[0] for x in top_5_spammer]
count = [x[1] for x in top_5_spammer]

plt.bar(name, count, width = 0.4)

plt.xlabel('Authors')
plt.ylabel('Number of contributions')
plt.title('Top 5 spammer')

plt.show()
```

Explore CONTENT colume with respect to the spam data

```
# Load nessary module
import re

# A function to see if a comment contains URL
def contains_url(comment):
    pattern = r'http[s]?://\S+|www\.\S+'
    return bool(re.search(pattern, comment))

pre_data['Contains_url'] = pre_data['CONTENT'].apply(contains_url)

num_urls = pre_data['Contains_url'].sum()

print(f'The number of comments containing URLs is: {num_urls}')
```

```
# Understand the existence of URLs in comments
url_spam = pre_data[(pre_data['CLASS'] == 1) & (pre_data['Contains_url'] == True)]

prop_url_spam = len(url_spam) / num_urls * 100

print(f'The proposition of spam comments containing a URL is: {prop_url_spam}%')
```

```
# Load necessary module
import emoji

# Function to check if a comment contains emojis
def contains_emoji(comment):
    return bool(emoji.emoji_count(comment))

pre_data['Contains_emo'] = pre_data['CONTENT'].apply(contains_emoji)

num_emo = pre_data['Contains_emo'].sum()

print(f'The number of comments containing emoji is: {num_emo}')
```

```
# Understand the existence of emojis in comments
emo_spam = pre_data[(pre_data['CLASS'] == 1) & (pre_data['Contains_emo'] == True)]

prop_emo_spam = len(emo_spam) / num_emo * 100

print(f'The proposition of spam comments containing emoji is: {prop_emo_spam}%')
```

```
# Extracting the spam comment and non-spam comment
spam_comment = pre_data[pre_data['CLASS'] == 1]['CONTENT']
non_spam_comment = pre_data[pre_data['CLASS'] == 0]['CONTENT']

punctuation_pattern = r'[^\w\s]'

punc_list = []
```

```
for comment in spam_comment:
    comment = comment.replace('\ufeff', '') # Removing the Byte Order Mark
    punctuations = re.findall(punctuation_pattern, comment)
    punc_list.extend(punctuations)

punctuation_counts = Counter(punc_list)

top_5_punc = punctuation_counts.most_common(5)

for x in top_5_punc:
    print(f'{x[0]}: {x[1]}')
```

```
# Define whether the comment contains top spam punctuations
def contains_punc(comment):
    tokens = comment.lower().split(' ')
    return any(word in top_5_punc for word in tokens)

pre_data['Contains_punc'] = pre_data['CONTENT'].apply(contains_punc)
```

```
# loading the necessary modules
import nltk # Natural language toolkit
from nltk.corpus import stopwords # Importing common stopwords
from nltk.tokenize import word_tokenize # Import tokenizer
from nltk.stem import WordNetLemmatizer # Import lemmatizer

# Download necessary module resources
nltk.download('stopwords') # Download stopwords database
nltk.download('punkt') # Download Punkt tokenizer
nltk.download('wordnet') # Enable the WordNetLemmatizer by downloading WordNet database
nltk.download('omw-1.4') # A additional database
```

```
# We are using WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

# Building the text cleaning function for the comments
def clean_text(text):
    # Remove URLs
    text = re.sub(r'http[s]?://\S+|www\.\S+', '', text)

    # Remove emojis
    text = emoji.replace_emoji(text, replace="")

    # Convert to lowercase
    text = text.lower()

    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)

    # Tokenization
    token = word_tokenize(text)

    # Remove stopwords
    token = [word for word in token if word not in stopwords.words('english')]

    # Lemmatization
    token = [lemmatizer.lemmatize(word) for word in token]

    return token
```

```
# Count top 5 most commonly used words in spam comment in their root forms
all_tokens = []
for comment in spam_comment:
    token = clean_text(comment)
    for x in token:
        all_tokens.append(x)
```

```
top_5_words = Counter(all_tokens).most_common(5)

for x in top_5_words:
    print(f'{x[0]}: {x[1]}')
```

```
# Function to check if a comment contain those top used spam words
def contains_words(comment):
    tokens = clean_text(comment)
    return any(word in tokens for word in top_5_words)

pre_data['Contains_words'] = pre_data['CONTENT'].apply(contains_words)
```

Explore VIDEO_NAME colume

```
# Count the number of unique videos in the data set
pre_data['VIDEO_NAME'].nunique()
```

```
# Visualize the composition of comments of the videos
vid_comment = pre_data['VIDEO_NAME'].value_counts()
spam_vid_comment = filtered_data['VIDEO_NAME'].value_counts()

total_vid_comment = pd.DataFrame({
    'Total comments of a video': vid_comment,
    'Total spam comment of a video': spam_vid_comment
})

total_vid_comment.plot(kind='bar', figsize=(12, 6), stacked=True)

plt.title("Number of comments by video")
plt.xlabel("Video name")
plt.ylabel("Comment counts")
plt.show()
```

Explore CLASS colume

```
# Understanding the compositon of this colume
num_comment = pre_data.shape[0]

num_spam = len(spam_comment)

prop_spam = len(spam_comment) / pre_data.shape[0] * 100

print(f'The proposition of spam comment is: {prop_spam}%')
```

Feature selection

```
# Showing all columes we have in the dataset so far.

colnam = pre_data.columns

for x in colnam:
    print(f'Colume name: {x}')
```

```
print(pre_data.head())
```

Forward stepwise selection

```
# Load the necessary modules
import spacy
from sklearn.feature_extraction.text import TfidfVectorizer

# Vectorizing the text data in the data set
```

```
pre_data['text_data'] = pre_data['CONTENT']

vectorizer = TfidfVectorizer()

tfidf_matrix = vectorizer.fit_transform(pre_data['text_data'])
tfidf_features = pd.DataFrame(tfidf_matrix.toarray(),
                  columns=vectorizer.get_feature_names_out())
```

```
# Load the necessary modules
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Define all possible predictors, we use dictionary becuase it is easier to present the final feature
all_predictors = {
    'Content colume': tfidf_features,
    'Contains_url colume': pre_data[['Contains_url']],
    'Contains_emo colume': pre_data[['Contains_emo']],
    'Contains_punc colume': pre_data[['Contains_punc']],
    'Contains_words colume': pre_data[['Contains_words']]
}

# Define the response variable
response = pre_data['CLASS']

# Define unused predictors
unused_predictors = list(all_predictors.keys())

# Initialization
selected_predictors = []
best_accuracy = 0

# The adjusted forward stepwise selection
while unused_predictors:
    temp_best_accuracy = 0
    temp_best_predictor = None

    for index in unused_predictors:
        model = RandomForestClassifier(bootstrap = True, random_state = 123)

        temp_predictors = pd.concat([all_predictors[x] for x in selected_predictors + [index]], axis = 1)

        x_train, x_test, y_train, y_test = train_test_split(temp_predictors, response, test_size = 0.3, random_state = 123)

        model.fit(x_train, y_train)

        y_prediction = model.predict(x_test)

        accuracy = accuracy_score(y_test, y_prediction)

        if temp_best_accuracy < accuracy:
            temp_best_accuracy = accuracy
            temp_best_predictor = index

    if temp_best_predictor != None and best_accuracy < temp_best_accuracy:
        selected_predictors.append(temp_best_predictor)
        unused_predictors.remove(temp_best_predictor)
        best_accuracy = temp_best_accuracy
    else:
        break
```

```
selected_predictors
```

```
best_accuracy
```

Investigating model statistics

```
# Investigate the preliminary model
final_predictors = pd.concat([all_predictors[x] for x in selected_predictors], axis=1)

x_train, x_test, y_train, y_test = train_test_split(final_predictors, response, test_size=0.3, random_state=123)

pre_model = RandomForestClassifier(bootstrap=True, random_state=123)

pre_model.fit(x_train, y_train)

y_prediction = pre_model.predict(x_test)
```

```
# Investigate the accuracy score
accuracy_score(y_test, y_prediction)
```

```
# Load the necessary package
from sklearn.metrics import classification_report

# Investigate F1 score
print(classification_report(y_test,y_prediction))
```

```
# Load the necessary module
from sklearn import metrics
from sklearn.metrics import confusion_matrix

# Investigate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_prediction)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = conf_matrix)
cm_display.plot()
plt.show()
```

```
# Load the necessary package
from sklearn.metrics import roc_curve, auc

# ROC and AUC
y_pred_prob = pre_model.predict_proba(x_test)[:, 1]

fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, label = f'ROC curve with AUC = {roc_auc:.3f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.legend(loc = 'lower right')
plt.show()
```

Final model construction

```
# Constructing final model using the variable selected above
final_predictors = pd.concat([all_predictors[x] for x in selected_predictors], axis=1)

final_model = RandomForestClassifier(bootstrap=True, random_state=123)

final_model.fit(final_predictors, response)
```

```
# Output the file for kaggle submission with the final model
test_data = pd.read_csv('test.csv')

test_data['Contains_url'] = test_data['CONTENT'].apply(contains_url)

test_data['Contains_punc'] = test_data['CONTENT'].apply(contains_punc)
```

```
test_data['Contains_emo'] = test_data['CONTENT'].apply(contains_emoji)

test_data['Contains_words'] = test_data['CONTENT'].apply(contains_words)

test_tfidf_matrix = vectorizer.transform(test_data['CONTENT'])
test_tfidf_features = pd.DataFrame(test_tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())

all_test_predictors = {
    'Content colume': test_tfidf_features,
    'Contains_url colume': test_data[['Contains_url']],
    'Contains_emo colume': test_data[['Contains_emo']],
    'Contains_punc colume': test_data[['Contains_punc']],
    'Contains_words colume': test_data[['Contains_words']]
}

final_test_predictors = pd.concat([all_test_predictors[x] for x in selected_predictors], axis = 1)

new_predictions = final_model.predict(final_test_predictors)

output = pd.DataFrame({
    'COMMENT_ID': test_data['COMMENT_ID'],
    'CLASS':new_predictions
})

output.to_csv('predictions_test_final.csv', index = False)
```