Binary Classification of text/comments Toxic and non-Toxic

Group Name: Group Bimal

Group Members:

First name	Last Name	Student number
Aanal Kamleshkumar	Patel	C0910376
Bimal Kumar	Shrestha	C0919385
Danilo	Diaz	C0889539
Ernie	Sumoso	C0881591
Jayachandhran	Saravanan	C0910392

Submission date: ***25-02-2024***

Template Prepared by: William Pourmajidi

Updated by Vahid Hadavi Nov 2023

Contents

List of Figures

Abstract

This project aims to develop a machine learning based model to classify the text-based comments/inputs into toxic or non-toxic. With the rapid growth of digital media platforms like YouTube, Quora, Reddit, and X (formerly known as Twitter), Instagram and Facebook have created a significant impact both in terms of insightful knowledge and hateful comments. Special attention is required to address the challenge of monitoring online communities and detecting toxic-related threats or comments.

In this activity, we try to leverage various Machine Learning techniques to perform a text classification. We explore the implementation of Natural Language Processing techniques like word vectorization, Term Frequency-Inverse Document Frequency(TF-IFD) and Bag-of-words (BoW). Further, the performance of the model is evaluated, and the impact of the above techniques is investigated. This report further discusses in detail the data procurement, pre-processing, visualization, modelling and fine-tuning procedures. Later, the most efficient model technique is identified, and the results are compared with the other approaches used in this project. The strategies to handle the imbalanced label and the mitigation steps are included in the methodology of extensive experiments. The overall process and methodology are tested, and their performance is disclosed.

Keywords: Digital media platform, toxic, Bag-of-words, Term Frequency-Inverse Document Frequency(TF-IFD), Machine learning, Natural Language Processing(NLP)

Introduction

The developments and improvements of digital technology in the last few years have created more mediums to gather virtually and share their perspectives as they have the freedom of speech on anything found on social platforms and streaming channels. The insights from Statista state that as of June 2022, YouTube uploads videos of 500 hours in length for every minute, and around 100 million social reactions are created in that instant. This includes comments, likes and content-creating transcripts. Most of the social interactions are unmonitored as the amount of data created is very huge, and processing with conventional algorithms will be heavy on computing and capital cost. This is just one of many such platforms on which social conversation is happening, and there are various sectors where the feature of exchanging individual opinions takes place. Most of the time, the comments or text being shared contain in-depth knowledge and are also used for good causes. Still, there are instances where it turns out to be cyberbullying, like threatening comments, harassment and abuse. These texts are termed “toxic content” or “hatred text” and create potential menaces among the public that need to be addressed.

In this assignment, we tried to implement a machine learning-based approach to tackle the mentioned problem and create a classifier model to detect and classify any given statement or review broadly into two categories, namely TOXIC and NON-TOXIC. Advancements in the field of computer science allow us to work on any complex entity and create an effective solution. One such advancement in AI (Artificial Intelligence) is Natural Language Processing (NLP), leverages text- and speech-based (linguistics) operations, making computation smooth and organized. Through this project, we aim to develop a classification-based model to detect toxicity in the text and enrich the online platform to be safer and healthier.

Our data was labeled manually because we identified discrepancies in which comments were perceived as violent and which were not. Additionally, we could implement some improving methodologies to guide the model towards a specific label when certain English words with the potential for violence needed consideration. This adjustment allows us to handle instances where words may be interpreted as violent, ensuring a more accurate classification. By focusing on the appearance of specific words in the text, the model can learn associations between those words and the target classes. However, this approach may not capture contextual information and word relationships effectively. More advanced techniques like word embeddings (e.g., Word2Vec, GloVe) or pre-trained models like BERT and GPT can handle contextual information and may provide better performance in various NLP tasks.

Literature Review

1. Related work – 1

   There are various studies and research being carried out to classify the text into toxic and non-toxic based on a generic or single type of toxicity. Only distinct numbers of works were done to unify all the categories of toxic content and classify them against non-toxic ones. This paper[5] discusses a detailed approach to the effective use of all toxic content, which is not only limited to hate, abuse, racism, sexual harassment and bullying. It further concludes the performance of three distinct models by comparing their learning curves with different vectorization techniques, like Bag-of-words and TF-IDF. Interestingly, the performance of the Conventional Neural Network outperforms all the other models used in their study, though the data size is not that big.

2. Related work – 2

   This literature[6] review provides us with a detailed overview of how to handle the imbalanced dataset using the “Synthetic Minority Oversampling Technique” SMOTE

method. It further discusses the approaching strategies for text-based datasets and the implementation of pre-trained models. The results of seven different models are compared for better accuracy and precision.

This prior work helps us to get a basic idea of how to approach the text binary classifier and the potential parameters to be taken into consideration while building the machine learning model. The insights and knowledge attained through this literature review is applied on the following steps of this project.

Pipeline

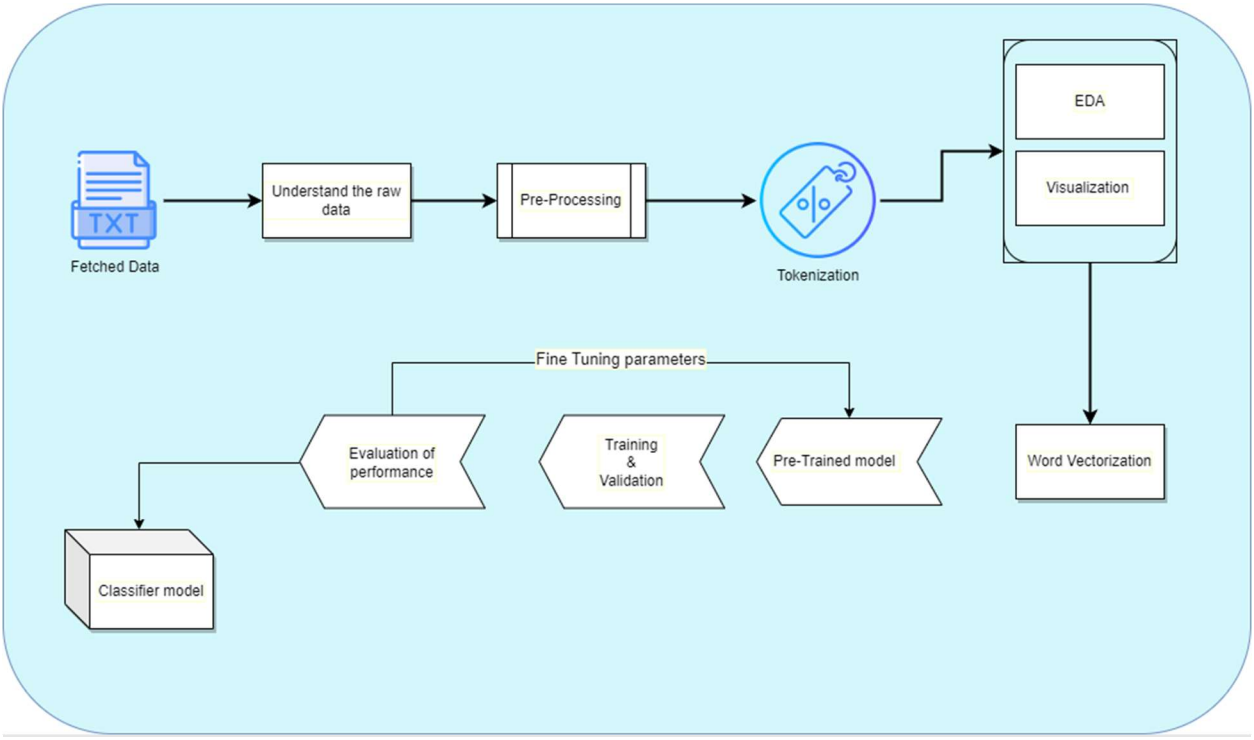

Figure 1: End-End Process Pipeline

Data Source

In this activity, we created a binary classification model developed on top of data scrapped from YouTube, the social media platform. Further, the processing pipeline from analyzing the raw data to creating cleaned useful data is described clearly in the following sections. The dataset is gathered from the renowned journal IEEE (Institute of Electrical and Electronics Engineers), where research work was published under the topic [4] “THREAT: A Large Annotated Corpus for Detection of Violent Threats.”

The dataset consists of around 30,000 unique sentences from a set of 10,000 comments on YouTube posted by various individuals. It is the first-ever dataset to group all the variety of anti-social comments available and prevailing in the on-stream media sites. The importance and relevance of choosing the dataset are considered, along with the significance of annotations applied to it. The characteristics of the data and the challenges to extracting useful information are discussed in detail in the data pre-processing stage of this project. The data is being fetched as a text file encoded in UTF-8 format, containing 2.8 million (2869069) characters from 48K line columns. The genuineness and usage of the dataset are verified before carrying it forward for processing and exploratory data analysis.

Pre-processing

1. Parsing and structuring raw data

   The first step is to check the original (raw) format of the data and underlying patterns to get the required information for processing. The data used in this project is stored in an unstructured text file format. It has noises like escape sequences, unwanted headers, non-alphanumeric letters, and special characters. The figure (2) shows the visual representation of the text data in its original format. The numeric values present in the beginning of the sentence are the annotated labels which denote the presence of toxic words in that sentence. The header

line contains the meta data of the comments like video ID, comment ID, commenter ID and the time of comment posted in YouTube at the time of annotation. The interesting challenge is to convert the raw data into a structured data frame for further processing and training purposes.

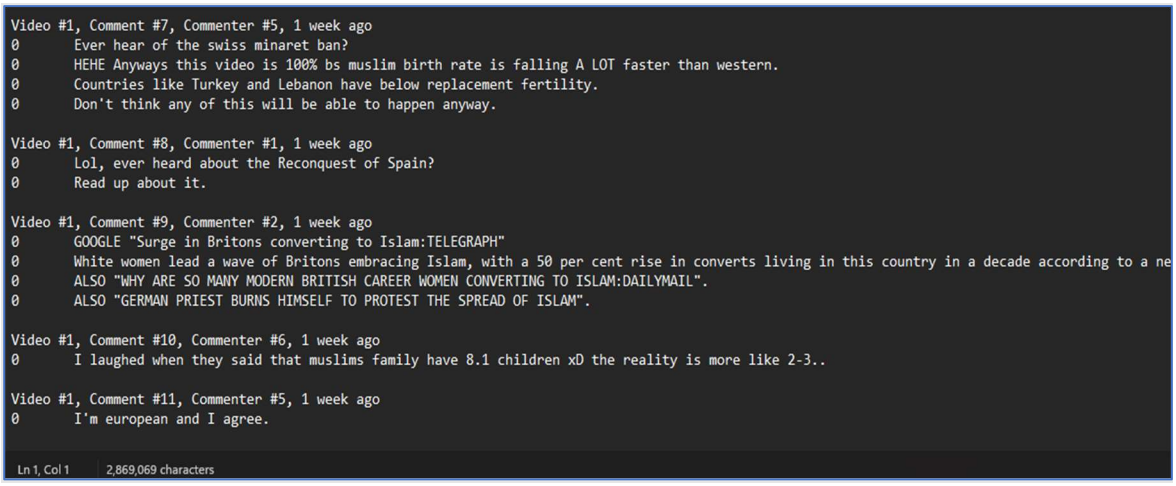

Figure 2: Snippet of Raw Data in Text file (UTF-8)

2. Data cleansing

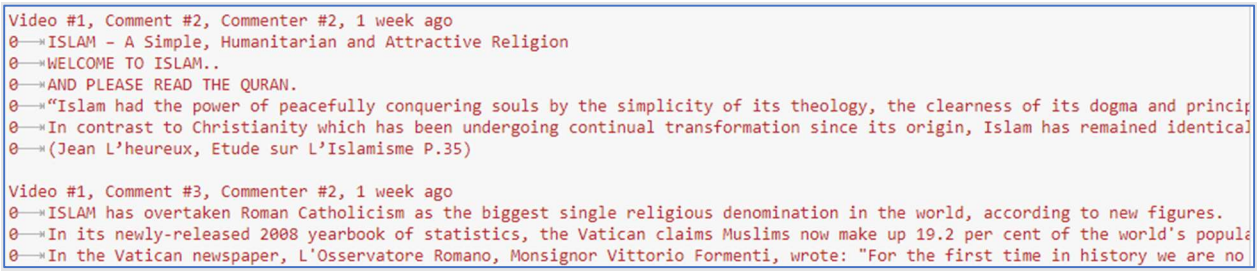

Figure 3: Data with all the noise and unwanted information

The text data is analyzed, and the occurrences are considered against creating a cleaned data frame is discussed below.

a. Special Characters

Special characters are the characters used in the text including punctuation and word limiters. For example, in our data, the occurrence of special characters like “[, . ! &

^ % $ # / \ ; ‘ ~ - _] ” is around 64,170. Though these characters give sentiment or meaning to the human while reading, their significance while converting it into numeric is not valued and thus the removal of these characters is a crucial step in our data cleaning or pre-processing stage. A maximum of 67 special characters appeared in a sentence and if they are not treated, it will create unwanted bias during classification.

b. Non- English words

This data is originated from the YouTube steaming platform comments, where people from all over the world has access and facility to add their thoughts in their own languages other than English. It has created challenge will processing the data as whole which will brings ambiguity in sematic and word-embeddings. This creates biased values during vectorization and the error rate will be unidentifiable. The following figure (4) represents the presence of non-English words in the review comments.

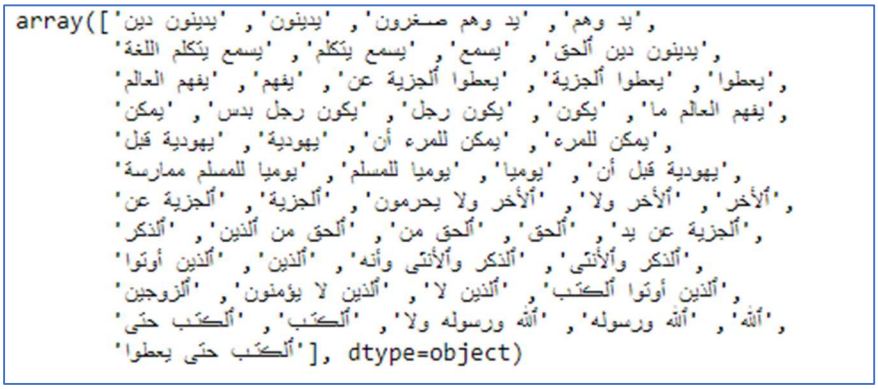

Figure 4: Array representation of non-English comments

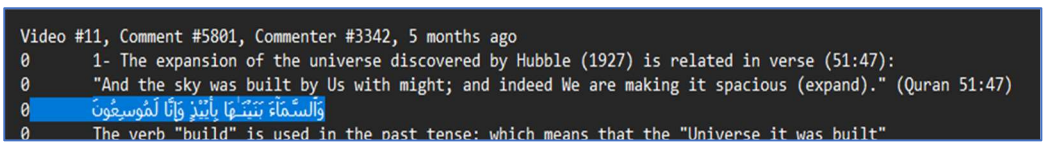

Figure 5: Instance of non-English word in the text file

This work is limited to discussing the effect of word-vectorization methods on different machine learning techniques applied to text data. Further, the language translation and the conversion process will be considered as future scope of topic to be explored and experimented with.

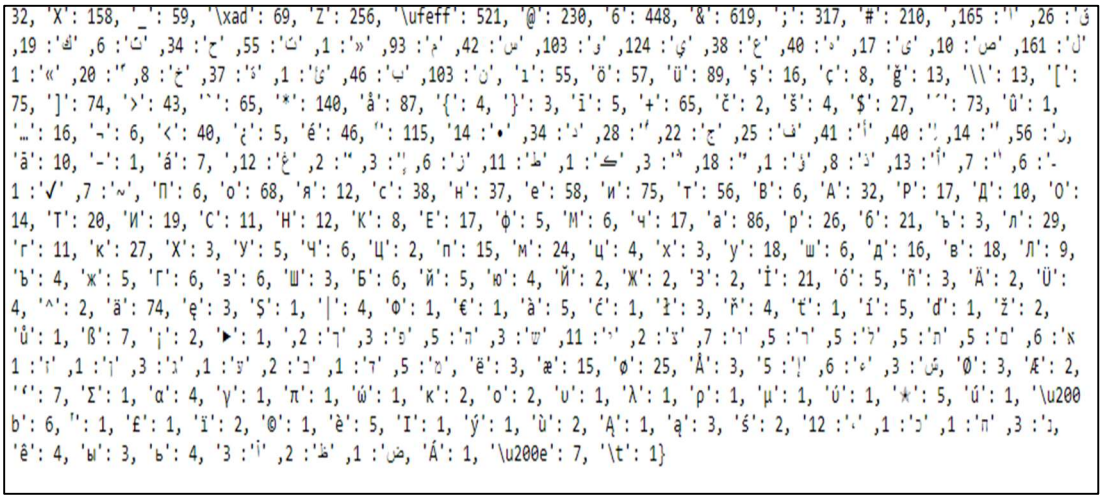

Figure 6: Special character occurrence and its count

By checking the characters present in the text file, character count is presented in dictionary format. It helps us to understand the diversity of the letters present in our data and we addressed this through the effective usage of ASCII (American Standard Code for Information Interchange) value. Total of 256 characters are present in ASCII table, out of which 128 unique ASCII codes represents English alphabets, numbers and some additional special characters like comma, full stop etc., Here we replaced all the characters which are not encoded in the ASCII table, thereby the focus of our processing is closer to one language i.e., English.

```
{'e': 141072,
 'u': 40987,
 'r': 77153,
 'o': 73135,
 'p': 31000,
 'a': 95088,
 'n': 78506,
 's': 89161,
 ' ': 176562,
 'w': 20060,
 't': 79985,
 'c': 40897,
 'h': 34991,
 'g': 32082,
 'y': 21189,
 'l': 72735,
 'i': 98992,
 'f': 15249,
 'm': 43348,
 'k': 18935,
 'b': 17787,
 'd': 43724,
 'v': 12901,
 'q': 1768,
 'j': 3901,
 'x': 2327,
 'z': 2093}
```

Figure 7: English Alphabet character occurrence and its count

c. White spaces

The white spaces in between texts and the header files are a quite common challenge in any text data. The white spaces are used to differentiate one word from the other by inserting a hexadecimal value of “0x20” in UTF-8 format holding size of 1 byte. The white space is important in text processing as it acts as a delimiter between words and allows easy tokenization of sentences or documents. But it also requires computing resources to process the extra white spaces present in the data which are created by human negligence, styling or by mistake. The extreme cases where the text data contains more than 2 white spaces are handled by trimming it with the help of regular expression functions.

d. Escape sequence characters

The escape notations are highly handy while creating a document in different format, it helps user by providing more readability and alignment-based features. As the data source is of UTF-8 based text file, the text contains various escape sequences to address the formatting standards. While processing the text file to convert it to a structured data frame, the handling of escape notations like "[\n \t \xhh \0 \f \b]" is a crucial step. The figure (8) gives a clear picture of how the dataset initially looks like.

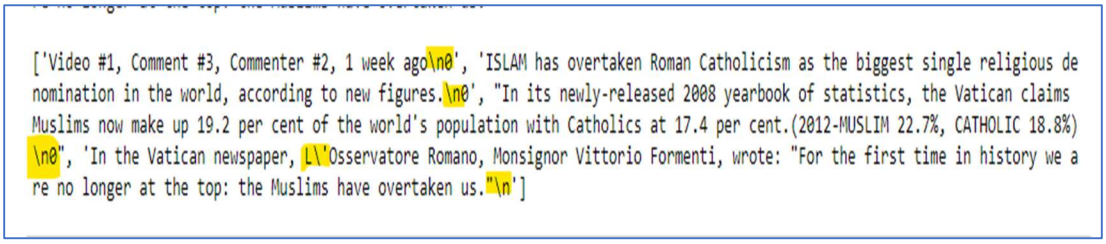

Figure 8: Presence of escape token '\n' in between the text

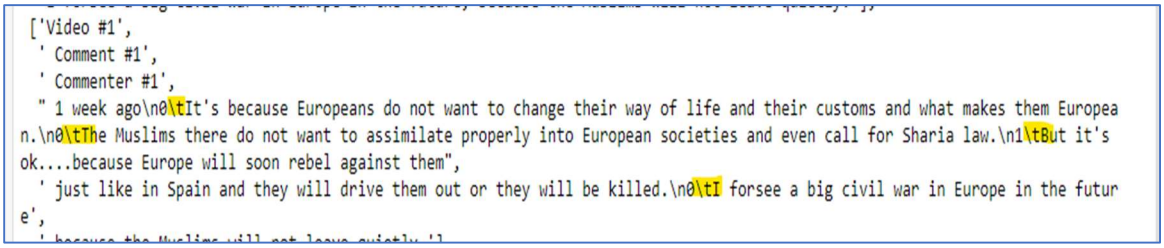

Figure 9: Presence of escape token '\t' in-between text

3. Stop word removal and Word Tokenization

The stop words are considered the most frequently used words in a text but do not have any sematic or significant value in the context of feature set. It does not carry information related to binary classification of this project, so handling them requires care.179 English words are deemed stop words according to NLTK corpus. Removing stop words from the text assists in dimensionality reduction of cleaned text data before

passing to the classification model. It also helps in improving efficiency of the model by neglecting the stop words which count to be the most frequent words in an unprocessed text data. This step helps the model to focus on more essential information from the text by removing the less vital words.

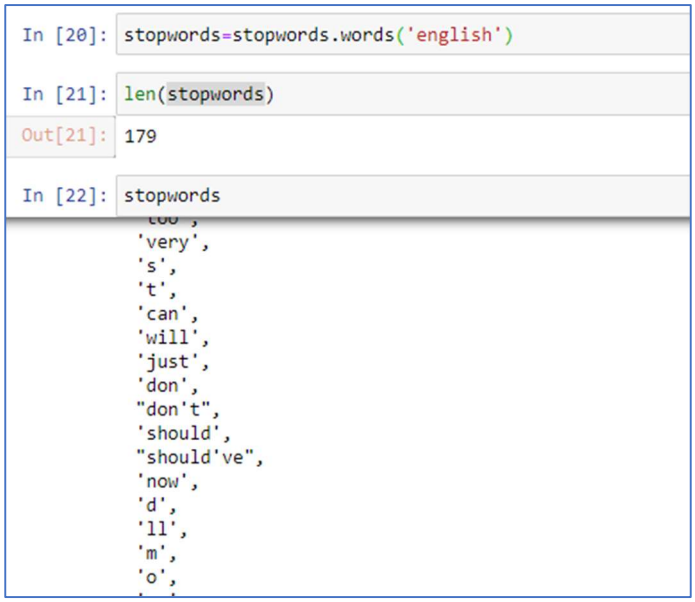

Figure 10: NLTK Stop word count (default)

Negation words:

The negation words present in the text contains huge impact on the sentiment of the particular sentences or comment. In general procedure, the stop words are removed from the text and processed for building model. In this project, we are dealing with sentiment-based analysis which has direct correlation with negation words like “not” , “no”, “doesn’t” . To handle this special case, we have modified the NLTK corpus by removing a few words from it. Further, colloquial words like [“u” for you , “v” for we , “d” for the] frequently used in the comments are added as new stop words to the corpus. This step allows us to retain the context of any comment regardless of negation

used in it. In future, we can try to implement indicator or new feature capturing techniques to these edge cases.

```
"wouldn't","wouldn","won't","won","weren't","weren","wasn't","wasn","shouldn't","shouldn","shan't","shan","needn't","needn","mustn't","mustn","mightn't","mightn","isn't","isn","haven't","haven","hasn't","hasn","hadn't","hadn","doesn't","didn't","didn","couldn't","couldn","aren't","aren","ain","don't","don","not","no","nor"
```

```
In [24]: t="it is a not good movie"
         t=t.split(" ")

In [26]: for words in t:
             if words not in stopwords:
                 print(words)

         good
         movie
```

Figure 11: Negation words and its impact over sentiment based comments

Our modified stop word list has 143 words in it. The negations words are retained without being ignored as stop words. This is a crucial step in pre-processing of our data, as the removal of negation based stop words remove the main context and lexicons of the sentence which contributes to the sentiment.

Tokenization

Tokenization is a mandatory process in NLP and is the process of separating text data into its smaller chunks called tokens. This can be characters for a word, words or group of words for a sentence whereas sentences for a paragraph. Tokenization is essential as it reduces the size of raw text and helps words to be represented numerically for analysis.

For this project, we have utilized sentence tokenization for EDA purposes and word tokenization for processing purposes. Sentence tokenization helped us find out the average number of sentences in the text. Word tokenization, however, helped us handle the stop words and apply lemmatization, thus helping vectorization.

word_english_aplha	cleaned_v_1	word_tokens	word_tokens_v
It's because Europeans do not want to change t...	europeans not want change way life customs mak...	[europeans, not, want, change, way, life, cust...	[europeans, not, want, change, way, life, cust...
The Muslims there do not want to assimilate pr...	muslims not want assimilate properly european ...	[muslims, not, want, assimilate, properly, eur...	[muslims, not, want, assimilate, properly, eur...

Figure 12: Tokenization of cleaned words using NLTK word_tokenization method

4. Word lengthening and typo :

The next step in our pre-processing is checking the typo and spelling errors present in the text. Initially, the word lengthening is applied to all the tokenized words present in the sentence. Repetition of same character is allowed two times in a word and anything exceeding the limit is lengthened using regular expression.

```
: ve.get_feature_names()[:10]

: ['aaaaaaah',
  'aaaaad',
  'aaaand',
  'aah',
  'aallah',
  'aan',
  'aandraag',
  'aanhangertje',
  'aankomt',
  'aanpakken']
```

Figure 13: Typo and character repetition

We tried to implement a spell checker using the penchant module, but the implementation failed, which we will address in the future scope of this project. The multiple occurrence of a character (typo) is addressed and the repetition is removed.

5. Lemmatization:

The final step is to derive the root form of the words present in the sentence, it is a main process as it provides more word normalization with respect to parts-of-speech (POS) tag. The NTLK module uses a dictionary called “WordNet”, it consists of all available and its relevant words to check near similar root form. For this project, we tried to bring the words to its noun based root form as the normalization technique. Further, we wanted to check the same for verb based root form as it is recommended practice for any sentiment based reviews or comments processing. It will be carried out in the future scope of this project, but we checked the variation of words between noun and verb root form.

	word_tokens	word_tokens_noun
0	[europeans, not, want, change, way, life, cust...	[european, not, want, change, way, life, custo...
1	[muslims, not, want, assimilate, properly, eur...	[muslim, not, want, assimilate, properly, euro...
2	[ok, europe, soon, rebel, like, spain, drive, ...	[ok, europe, soon, rebel, like, spain, drive, ...

	word_tokens	word_tokens_verb
0	[europeans, not, want, change, way, life, cust...	[europeans, not, want, change, way, life, cust...
1	[muslims, not, want, assimilate, properly, eur...	[muslims, not, want, assimilate, properly, eur...
2	[ok, europe, soon, rebel, like, spain, drive, ...	[ok, europe, soon, rebel, like, spain, drive, ...

Figure 14: Lemmatization of word tokens (verb, noun)

The upcoming section of this document will discuss in detail about the Data analysis carried out and provide interesting findings.

Exploratory Data Analysis & Visualization

Exploratory data analysis is another crucial step in any NLP or data related project, as it helps us to understand the content of the data and uncover the trends and patterns in the data. The EDA is done twice during this project work, one on the raw data and the later one is on the processed data. It helped us to navigate the preparation steps required for building the word vectorizations and model pipelining.

EDA – 1

This extensive analysis is applied on the raw data fetched from the source. We followed trail and error approach to gather all sort of information from the given data. Later, we constructed a pipeline for preliminary data analysis which includes,

1. Word count
2. Character count
3. Special character count
4. Stop word count
5. Unique and repetition of words
6. Case sensitivity
7. Target variable ratio
8. Duplicated values
9. Missing values
10. Word token and sentence token
11. White spaces
12. Character Dictionary

Apart from this basic EDA like understanding the shape, size, statistical summary and the distribution are observed. This helped us to find the presence of non-English words in the text and the spacing conflicts.

```
------------------------------
SHAPE
# Rows:  28643
# Columns: 2
------------------------------
TARGET VALUE COUNTS
label
0    27256
1     1387
Name: count, dtype: int64
Total: 28643
------------------------------
TARGET VALUE COUNTS PERCENTAGE (%)
label
0    95.15763
1     4.84237
Name: proportion, dtype: float64
Total: 100.00 %
------------------------------
MISSING VALUES
text     0
label    0
dtype: int64
Total: 0
------------------------------
MISSING VALUES PERCENTAGE (%)
text     0.0
label    0.0
dtype: float64
Total: 100.00 %
------------------------------
DUPLICATED ROWS
Total: 352
------------------------------
DUPLICATED ROWS PERCENTAGE (%)
Total: 1.23 %
------------------------------
```

Figure 15: Analysis and stats of raw data

As per the initial analysis the missing value count is null and the duplicate values found in the rows are around 1.23 % which is not impacting our dataset, so it will be removed. The shape of the dataset is 28643,2 which is quite decent size for building a binary classifier. The target variable imbalance is found, which is a most common in any real world dataset. To address this, we used SMOTE method to create synthetic data for the minority class. It will be discussed further in the methodology step of this project.

	index	text	label	word_count	char_count	special_char_count	numerics_count	upper_count	lower_count	stopwords_count	unique_words_c
28286	28638	yeah we are all monsters..I'm gonna kill u rig...	1	10	56	5	0	1	39	3	
28287	28639	stupid brainwashed idiot..\n	0	3	27	2	0	0	22	0	
28288	28640	have you EVER been to Serbia or kosovo...fucki...	0	13	79	8	0	9	48	7	
28289	28641	probably u mean to this monsters, fucker /watc...	0	8	62	3	2	4	43	2	
28290	28642	the fucking funniest thing is that fucking ame...	0	13	87	3	0	0	71	4	

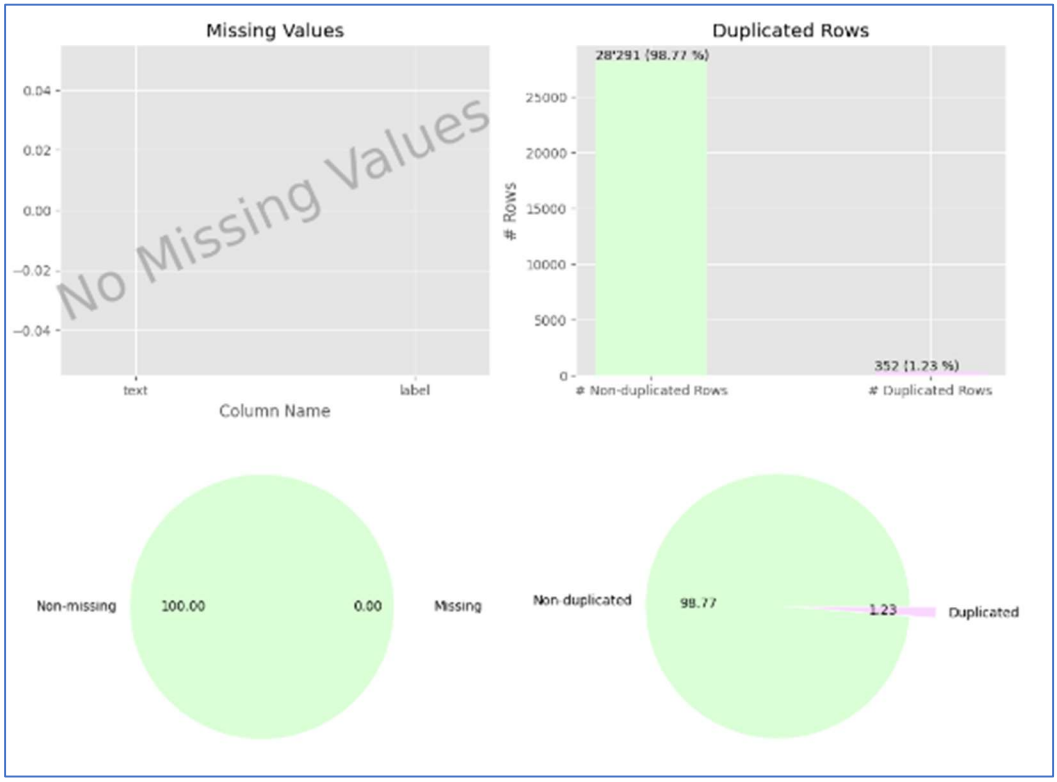

Figure 16: Data frame snippet and plot of missing & Duplicate values

The above figure(16) compromise the elaborated insights on the text data gather from the source, it will be utilized during pre-processing steps. The statistical based summary is created for both the labels and the results are attached below.

	index	label	word_count	char_count	special_char_count	numerics_count	upper_count	lower_count	stopwords_count	unique_words_co
count	26932.000000	26932.0	26932.000000	26932.000000	26932.000000	26932.000000	26932.000000	26932.000000	26932.000000	26932.000
mean	14257.617258	0.0	14.028888	79.051537	2.238638	0.339373	4.614473	57.367889	5.588705	12.953
std	8228.680561	0.0	11.132556	63.259047	2.468285	1.520522	15.381334	49.607612	5.435356	9.156
min	0.000000	0.0	1.000000	6.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000
25%	7089.750000	0.0	7.000000	37.000000	1.000000	0.000000	1.000000	25.000000	2.000000	7.000
50%	14315.500000	0.0	11.000000	62.000000	1.000000	0.000000	1.000000	44.000000	4.000000	11.000
75%	21358.250000	0.0	18.000000	99.000000	3.000000	0.000000	3.000000	74.000000	8.000000	17.000
max	28642.000000	0.0	108.000000	507.000000	67.000000	88.000000	396.000000	412.000000	67.000000	83.000

Figure 17: Non-Toxic comment statistical analysis

	index	label	word_count	char_count	special_char_count	numerics_count	upper_count	lower_count	stopwords_count	unique_words_count
count	1359.000000	1359.0	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000
mean	15547.682855	1.0	15.116262	82.760854	2.364238	0.199411	10.647535	54.115526	5.540839	13.707873
std	9008.263592	0.0	13.073364	71.426901	3.665978	0.848181	30.098001	54.345787	6.077402	10.443305
min	2.000000	1.0	2.000000	11.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.000000
25%	8632.000000	1.0	7.000000	37.000000	1.000000	0.000000	1.000000	19.000000	1.000000	7.000000
50%	14406.000000	1.0	11.000000	62.000000	1.000000	0.000000	1.000000	41.000000	4.000000	11.000000
75%	24832.500000	1.0	19.000000	106.000000	3.000000	0.000000	4.000000	72.000000	8.000000	17.000000
max	28638.000000	1.0	92.000000	500.000000	55.000000	9.000000	405.000000	383.000000	52.000000	75.000000

Figure 18: Toxic comment statistical analysis

The stop words and the word occurrences in each sentences of the document is plotted for through exploitation.

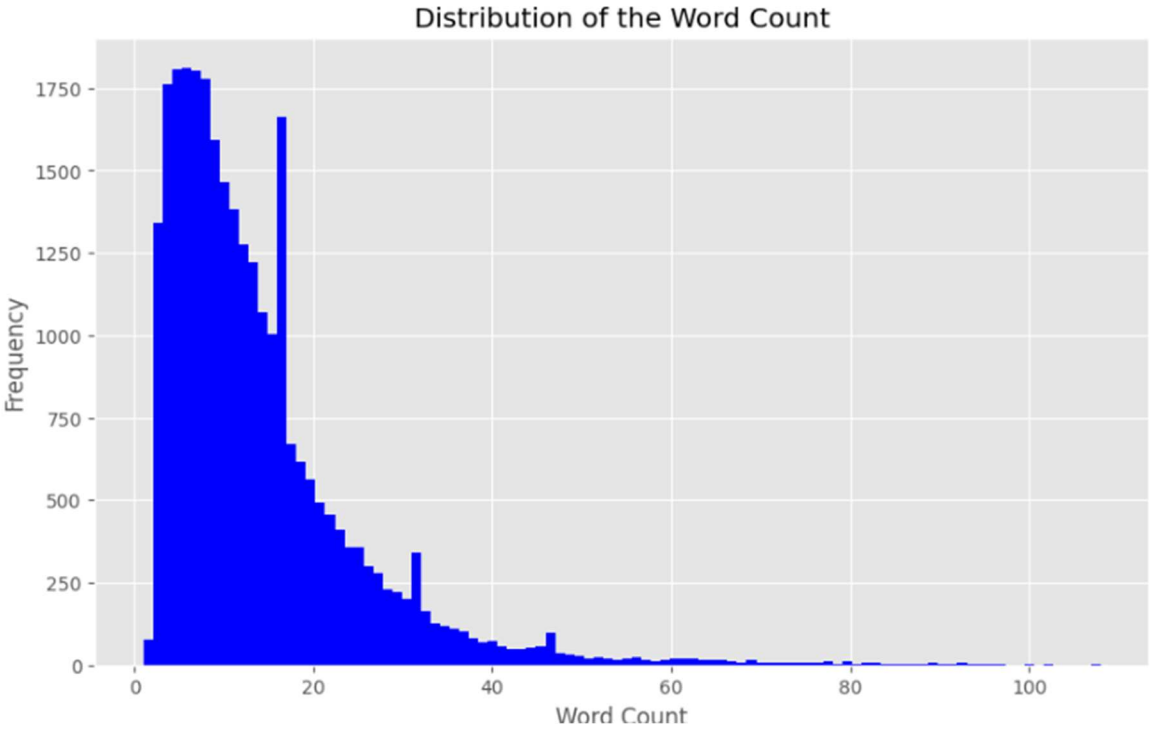

Figure 19: Visualization of word count frequency

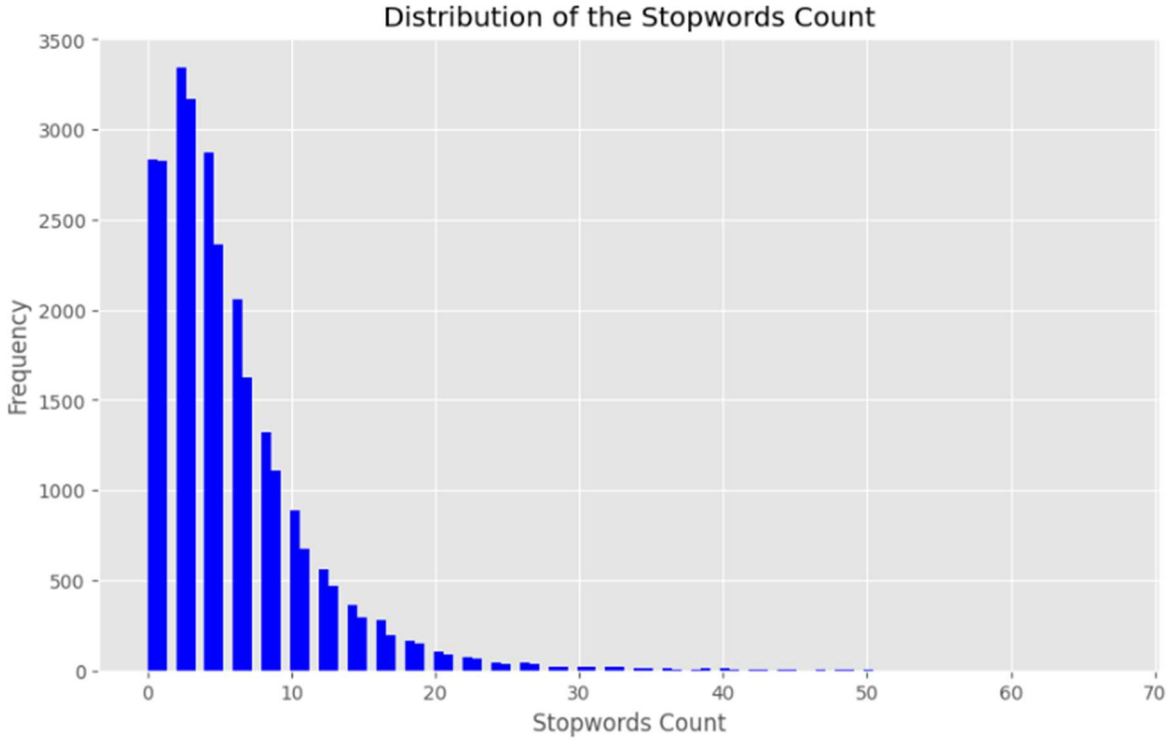

Figure 20: Visualization of stop word count frequency

The skewness and kurtosis is checked understand the spread out and insights like symmetrical based characteristics to conclude the analysis. The positive kurtosis in the both stop words and word count suggests that the extreme values like ‘a’ etc.,. The distribution is skewed towards the right side.

```
Skewness: 2.474932
Kurtosis: 9.424836
```

```
Skewness: 2.324923
Kurtosis: 9.248397
```

Figure 21: Skewness and Kurtosis analysis on stop words & Word count

The whole character set is stored inside the dictionary with character as key and the count is added as values present in it. This provide a over all view of letters present in the text file.

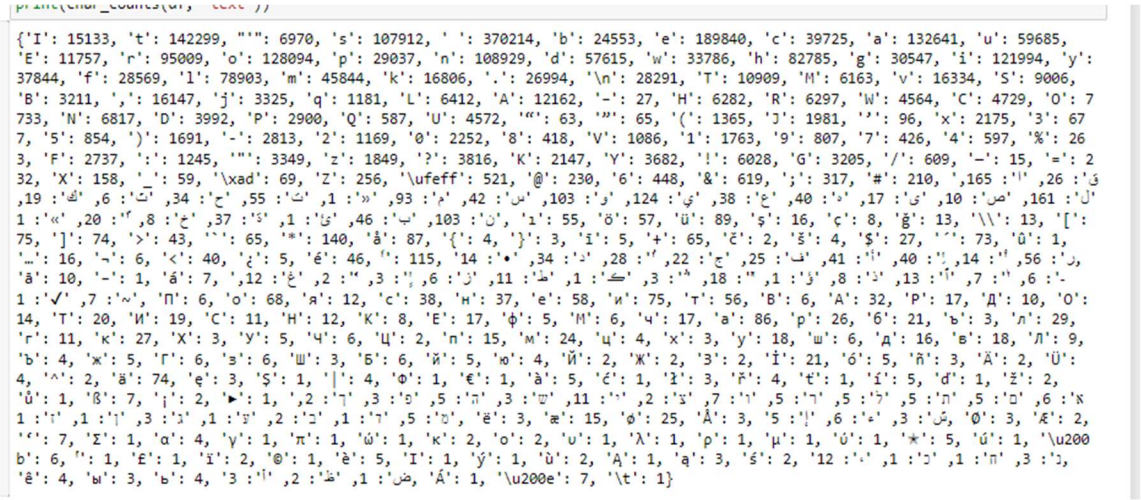

Figure 22: Dictionary containing character and its count

Word cloud is a visualization method, where the words are displayed in an order, the high frequent words are displayed with increased size and low frequent words are displayed in smaller size. The unigram, bigram based word set is being utilized.

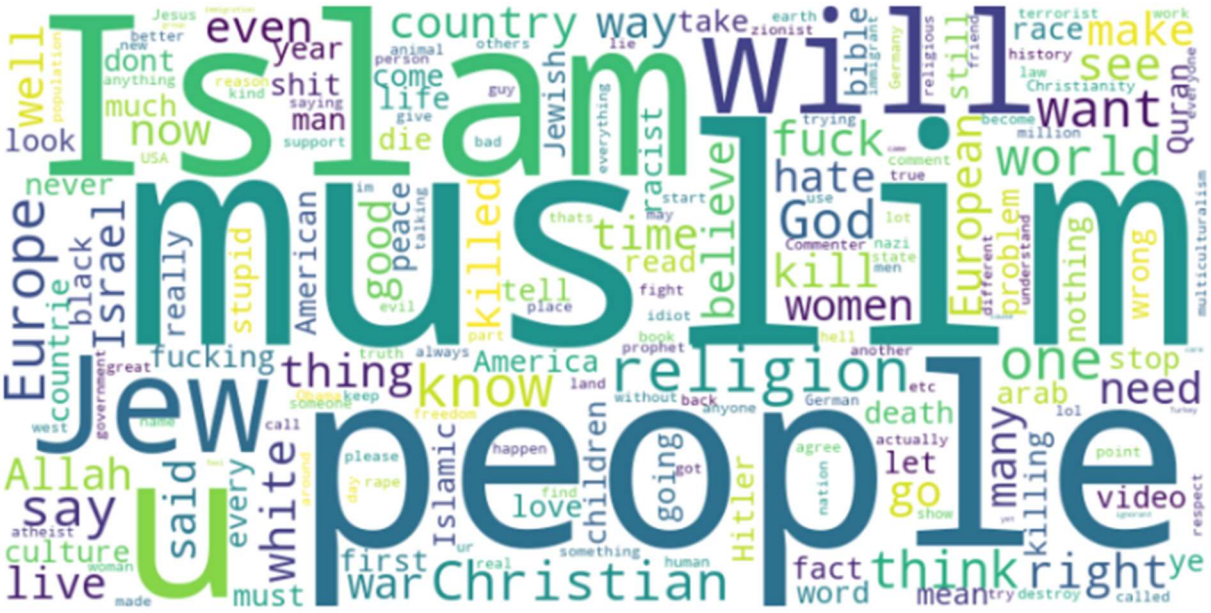

Figure 23: Word cloud of Unigram word set (non-toxic)

Figure 24: word cloud of Unigram word set (toxic)

EDA – 2

Post text pre-processing, the exploratory data analysis is applied on the cleaned data to check the data characteristics and features. It also allowed us to structure the dataset into toxic and non-toxic

word tokens in noun-based lemma format. Further, the dataset is segregated into train and test split with the size of 9:1. The imbalance is addressed by applying SMOTE sampling method on the training data set.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28643 entries, 0 to 28642
Data columns (total 17 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   text                28643 non-null  object
 1   label               28643 non-null  int64
 2   word_count          28643 non-null  int64
 3   char_count          28643 non-null  int64
 4   special_char_count  28643 non-null  int64
 5   numerics_count      28643 non-null  int64
 6   upper_count         28643 non-null  int64
 7   lower_count         28643 non-null  int64
 8   stopwords_count     28643 non-null  int64
 9   unique_words_count  28643 non-null  int64
 10  sent_token          28643 non-null  object
 11  word_english_aplha  28643 non-null  object
 12  cleaned_v_1         28549 non-null  object
 13  word_tokens         28643 non-null  object
 14  word_tokens_a       28643 non-null  object
 15  word_tokens_verb    28643 non-null  object
 16  word_tokens_noun    28643 non-null  object
dtypes: int64(9), object(8)
memory usage: 3.7+ MB
None
 ================================================

shape (28643, 17)
 ================================================

size 486931
 ================================================

columns Index(['text', 'label', 'word_count', 'char_count', 'special_char_count',
       'numerics_count', 'upper_count', 'lower_count', 'stopwords_count',
       'unique_words_count', 'sent_token', 'word_english_aplha', 'cleaned_v_1',
       'word_tokens', 'word_tokens_a', 'word_tokens_verb', 'word_tokens_noun'],
      dtype='object')
 ================================================
```

Figure 25: Analysis of cleaned data after pre-processing

The cleaned data set consists of only English based characters only. The figure(25) represents the count values of each characters.

```
: {'e': 141072,
   'u': 40987,
   'r': 77153,
   'o': 73135,
   'p': 31000,
   'a': 95088,
   'n': 78506,
   's': 89161,
   ' ': 176562,
   'w': 20060,
   't': 79985,
   'c': 40897,
   'h': 34991,
   'g': 32082,
   'y': 21189,
   'l': 72735,
   'i': 98992,
   'f': 15249,
   'm': 43348,
   'k': 18935,
   'b': 17787,
   'd': 43724,
   'v': 12901,
   'q': 1768,
   'j': 3901,
   'x': 2327,
   'z': 2093}
```

Figure 26: Count of English characters present in the cleaned data

After processing of data, the null values are included in place of non-English words and its overall presence is less than a percentage, so it is dropped from the data frame.

```
index                0
text                 0
label                0
word_count           0
char_count           0
special_char_count   0
numerics_count       0
upper_count          0
lower_count          0
stopwords_count      0
unique_words_count   0
cleaned_text       115
dtype: int64
```

```
index                0
text                 0
label                0
word_count           0
char_count           0
special_char_count   0
numerics_count       0
upper_count          0
lower_count          0
stopwords_count      0
unique_words_count   0
cleaned_text         0
dtype: int64
```

Figure 27: Checking and removal of Null values present in the data

The following visualization techniques are applied on the cleaned data set to infer the words present in two categories. Later, the word frequency is also plotted with unigram word set.

Figure 28: Word cloud of bigram word set in non-toxic comments.

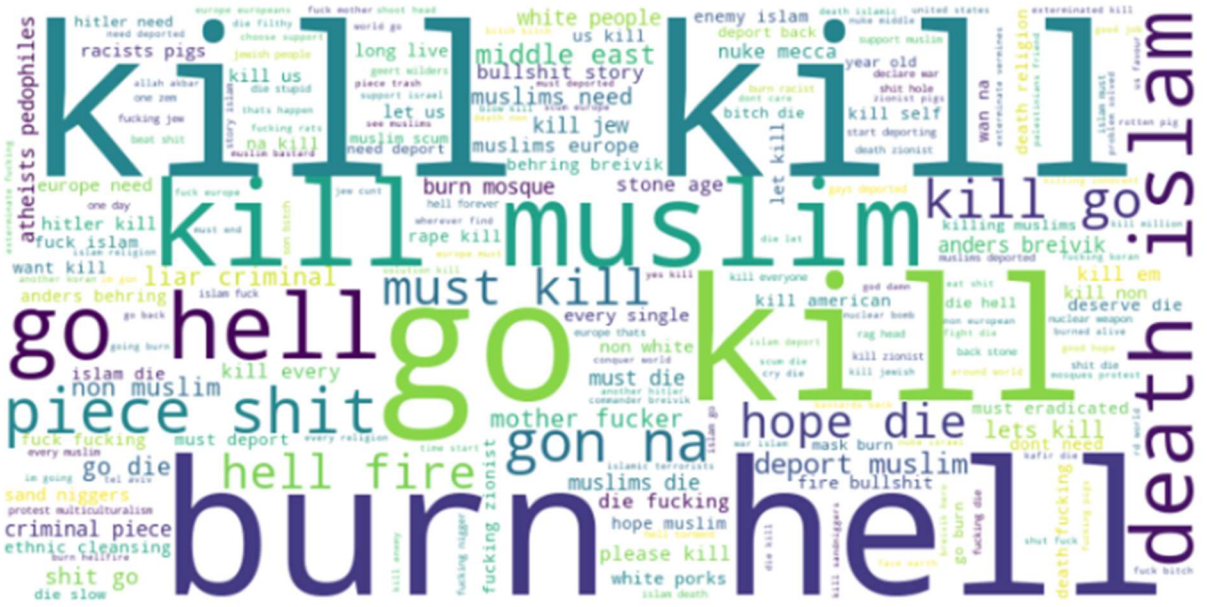

Figure 29: Word cloud of bigram word set in toxic comments.

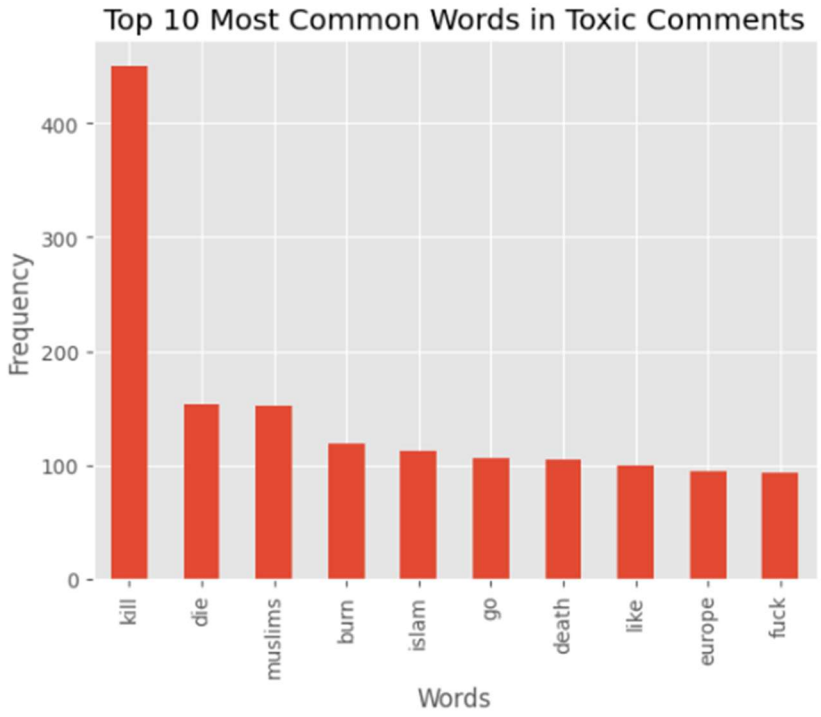

Figure 30: Toxic comment word frequency plot

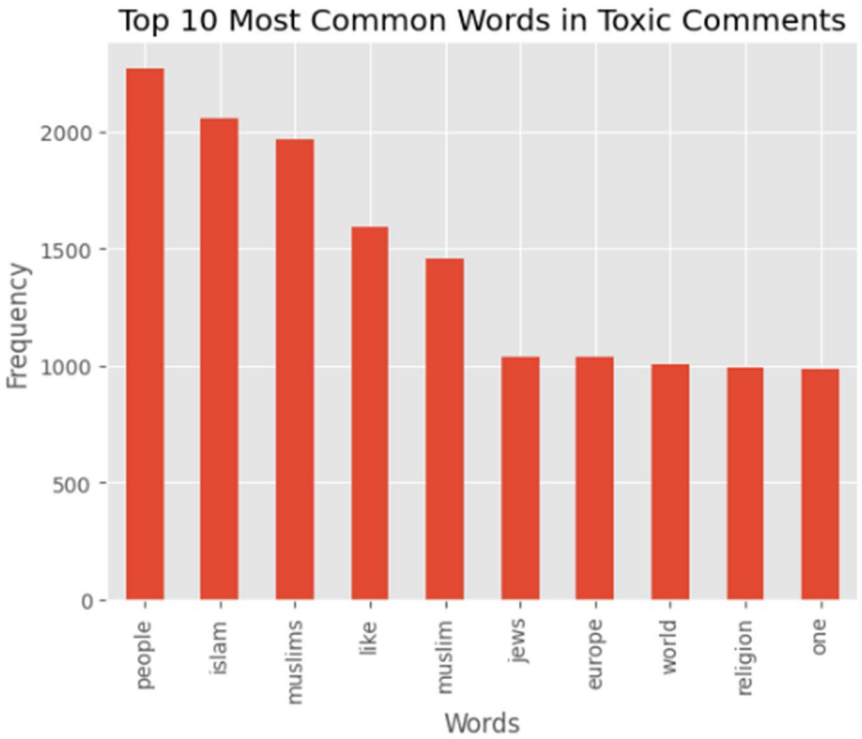

Figure 31: Non-Toxic comment word frequency plot

Methods

1. **Bag Of Words (BOW):** In NLP, Bag-of-words is a text modeling technique. The BOW model is implemented to determine how frequently a word collection or "Bag" has appeared in a document. It assists in the process of extracting features from the given text data. Unorganized and different length data can be converted into fixed length vectors using this method. For instance, for every record in the document, it will produce an array that contains the number of repetitions of each word from entire document.

    a. Our Implementation of Bag of words:

```
: from sklearn.feature_extraction.text import CountVectorizer

  vectorizer = CountVectorizer(
      tokenizer=LemmaTokenizer(),
      strip_accents="unicode", # Remove accents and perform other character normalization
      ngram_range=(1, 2), # Use unigrams and bigrams
      min_df=0.0005, # Ignore terms that have a document frequency strictly lower than the given threshold
      max_df=0.8, # Ignore terms that have a document frequency strictly higher than the given threshold
  )

: vectorizer

: CountVectorizer(max_df=0.8, min_df=0.0005, ngram_range=(1, 2),
                  strip_accents='unicode',
                  tokenizer=<__main__.LemmaTokenizer object at 0x000002C97EB8A470>)
```

Figure 32: Bag of Words – Vectorization

```
a = vectorizer.fit_transform(x_train)
a.toarray()
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
vectorizer.get_feature_names_out()
```

```
array(['ability', 'able', 'abortion', ..., 'zionist', 'zionist jew',
       'zionists'], dtype=object)
```

Figure 33: Converstion of BoW into array

2. **Term Frequency – Inverse Document Frequency (TF-IDF):** The most famous NLP based statistical method for determining word frequency from a document using natural language processing. The relevance of each word in the document will be converted into vectors by the importance of the words. The exact opposite of BOW, which assigns greater importance to less frequence words, is how TF-IDF calculates word significance. There are two components to this method.

    a. **Term Frequency (TF):** This section will count how often each term appears in each document. A function is available to determine the frequency of each word.

$$TF = \frac{\text{number of times the term appears in the document}}{\text{total number of terms in the document}}$$

b. **Inverse Document Frequency (IDF): This** section will determine the importance of words in the documents. Specifically, uncommon or distractive words will be given more weight to aid the prediction. while more common words will be given less importance as they will not contribute to future prediction of the model.

$$IDF = log(\frac{\text{number of the documents in the corpus}}{\text{number of documents in the corpus contain the term}})$$

c. The multiplication of both the terms is considered as **TF-IDF**

$$TF\text{-}IDF = TF * IDF$$

Our Implementation of Vectorization - TF-IDF:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(
    tokenizer=LemmaTokenizer(),
    strip_accents="unicode", # Remove accents and perform other character normalization
    ngram_range=(1, 2), # Use unigrams and bigrams
    max_features = 3000
)
```

```
tfidf_vectorizer
```

```
TfidfVectorizer
TfidfVectorizer(max_features=3000, ngram_range=(1, 2), strip_accents='unicode',
                tokenizer=<__main__.LemmaTokenizer object at 0x000002C943D2DC00>)
```

Figure 34: Term Frequency – Inverse Document Frequency -Vectorization

We created the features using bi-grams and trigrams, here is the output of the features we got from TF-IDF vectorization.

```
b = tfidf_vectorizer.fit_transform(x_train)
b.toarray()
```

```
C:\Users\bimal\AppData\Local\Programs\Python\Python310\lib\site-packages\skle
  warnings.warn(
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
tfidf_vectorizer.get_feature_names_out()
```

```
array(['aan', 'ability', 'able', ..., 'zionist jew', 'zionists', 'zu'],
      dtype=object)
```

```
len(tfidf_vectorizer.get_feature_names_out())
```

```
3000
```

Figure 35: Conversion of TF-IDF into array

Implementing SMOTE:

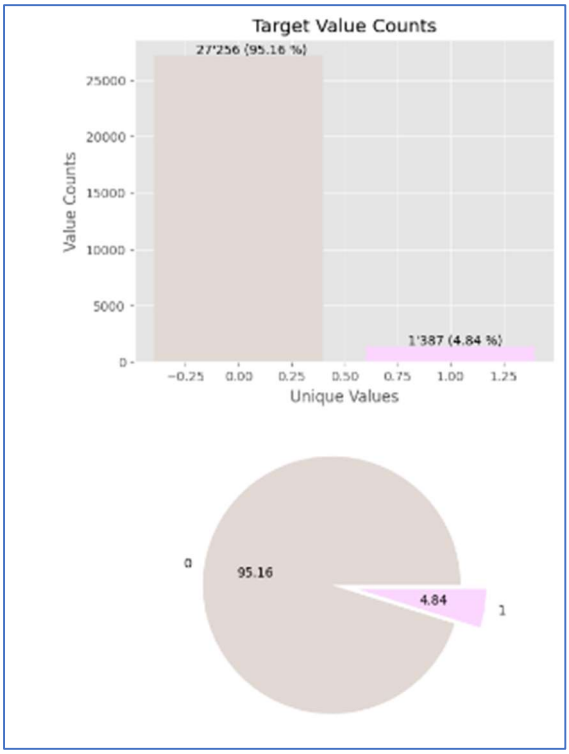

Figure 36: Target variable imbalance visualization

As discussed above, We found class imbalance in our target feature, to prevent bias in our model, we implanted Synthetic Minority Over-sampling Technique (SMOTE). There is more non-toxic data than toxic data which could cause our model to become biased towards non-toxcity. SMOTE is a technique that helps in overcoming the overfitting of the model by generating synthetic data for the minority class.

```
# Apply SMOTE for oversampling
pipeline = make_pipeline(vectorizer,SMOTE(), model)
```

Figure 37: Applying sampling technique to address the imbalance.

Results

We used five different models for both methods (BOW, TF-IDF) such as Nive Bayse, Support Vector Classifier, Logistic Regression, Random Forest and gradient boost classifier. For the vectorization method Gradient boost gave the best results.

1. Best model for Bag of words (Gradient boost):

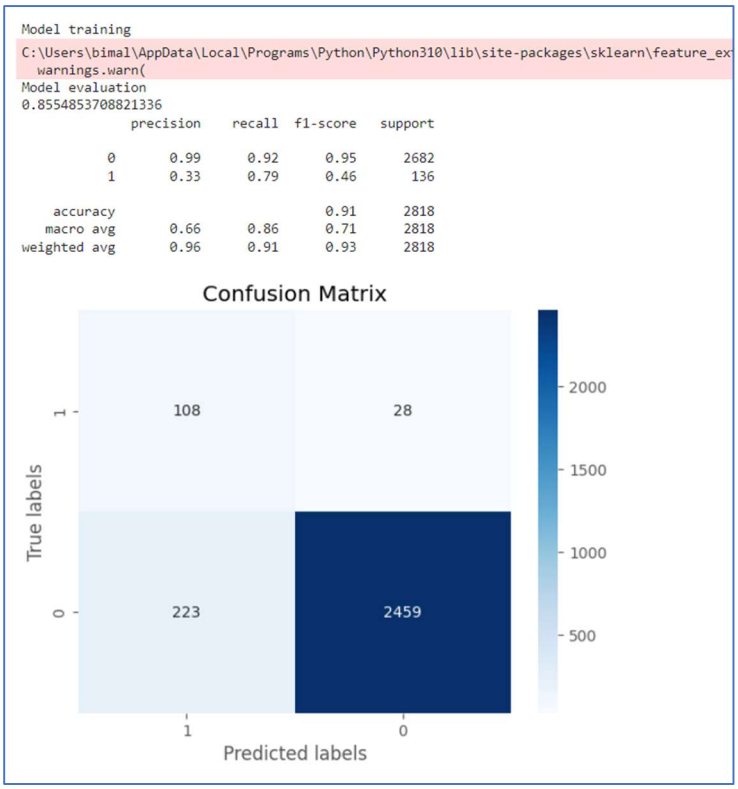

Figure 38: Confusion matrix of Grediant boost (*BOW*).

2. Best model for TF-IDF (Grediant boost):

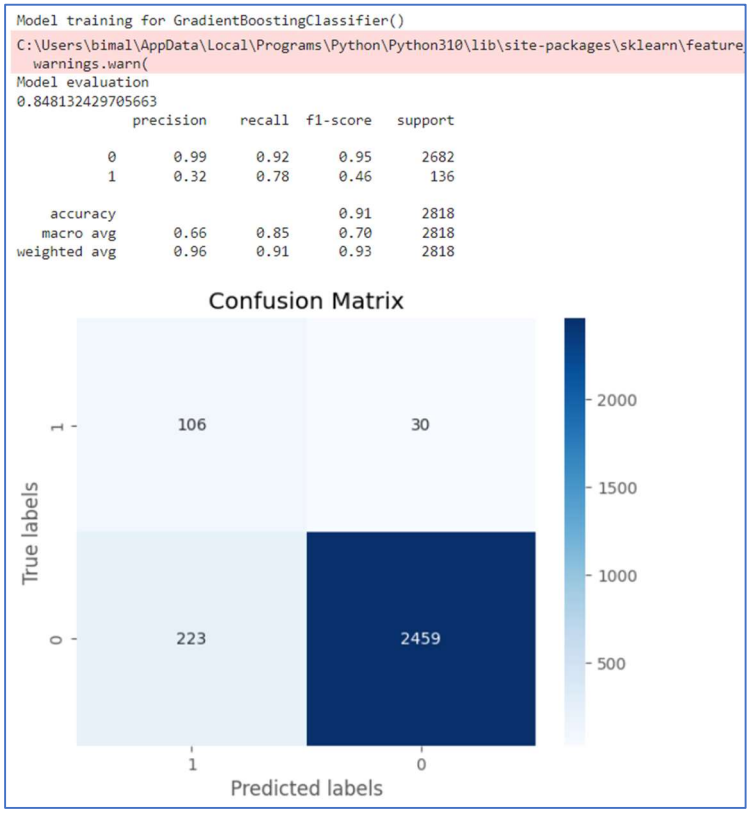

Figure 39: Confusion matrix of Gradient boost *(TF-IDF).*

Comparison of the results (accuracy):

Methods	Bag of Words	TF-IDF
Naive Bayes	80.92 %	82.26 %
Support Vector Classifier	72.24 %	71.38 %
Logistic Regression	82.35 %	83 %
Random Forest	77.44 %	78.49 %
Gradient boosting classifier	85.54 %	84.81 %

The Results we obtained indicate that Bag of Words is outperformed by TF-IDF vectorization method with algorithms such as, Nive Bayse, Logistic regression and Random Forest, however, Bag of Words is more accurate for the algorithms like SVC and Gradient boosting. Despite this, we can conclude that TF-IDF exceeds Bag of Words in overall performance.

Conclusions and Future Work

We employed YouTube data for testing our model and were able to accurately predict the sentiments of the comments of different users on a variety of videos, we are creating an is Toxic column to check whether the comment is toxic or not. We have implemented the fundamental model of both of our vectorization methods (Bag of Words and TF-IDF), but in the future we can fine-tune our best-performing model to achieve better outcomes for each method.

References

[1] Great Learning Team. (2022, October 24). *An Introduction to Bag of Words in NLP using Python | What is BoW?* Great Learning Blog: Free Resources What Matters to Shape Your Career! https://www.mygreatlearning.com/blog/bag-of-words/

[2] Wikipedia contributors. (2023, December 13). *Bag-of-words model*. Wikipedia. https://en.wikipedia.org/wiki/Bag-of-words_model

[3] *TF-IDF — Term Frequency-Inverse Document Frequency – LearnDataSCI*. (n.d.). https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/#:~:text=Term%20Frequency%20%2D%20Inverse%20Document%20Frequency%20(TF%2DIDF)%20is,%2C%20relative%20to%20a%20corpus).

[4] *THREAT: a large annotated corpus for detection of violent threats*. (2019, September 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/8877435

[5] Hammer, H. (2019). *THREAT: a large annotated corpus for detection of violent threats*. https://www.semanticscholar.org/paper/THREAT%3A-A-Large-Annotated-Corpus-for-Detection-of-Hammer-Riegler/32b11247f4232d1262633b3cd48b6de3a2360632

[6] Satpathy, S. (2023, November 17). *SMOTE for Imbalanced Classification with Python*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/#:~:text=its%20related%20counterparts.-,SMOTE%3A%20Synthetic%20Minority%20Oversampling%20Technique,problem%20posed%20by%20random%20oversampling.

[7] *What is Natural Language Processing? | IBM*. (n.d.). https://www.ibm.com/topics/natural-language-processing

[8] *YouTube: hours of video uploaded every minute 2022 | Statista*. (2024, February 13). Statista. https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/

[9] Bullas, J. (2023, December 5). *35 Mind numbing YouTube Facts, Figures and Statistics – infographic*. Jeffbullas's Blog. https://www.jeffbullas.com/35-mind-numbing-youtube-facts-figures-and-statistics-infographic/

Appendix 1: Validation of Target Label

In this appendix, we try to cross validate our annotated labels using NLTK sentiment Analyzer library module. The ultimate reason is to identify the annotations is matching with the positive or negative based on the sentiment intensity. It provided interesting fail proof check on the manually annotated data, where are there few occurrences of wrong labelling is done. It uses the polarity score to get the sentiment value. It works similar to SVM classifier model to differentiate the sentences broadly into three categories, Positive, Negative and Neutral. The null values are termed to be NaN or None.

The below figure highlights the sentiment analysis based label compared with the manually annotated label. It requires further optimization and modification for fully functioning, we will taking this appendix as our future scope.

	text	label	Sentiment_review
2	But it's ok....because Europe will soon rebel ...	1	0
4	ISLAM – A Simple, Humanitarian and Attractive ...	0	1
5	WELCOME TO ISLAM..\n	0	1
6	AND PLEASE READ THE QURAN.\n	0	1
7	"Islam had the power of peacefully conquering ...	0	1
...	...	...	...
28631	EU is US puppet, it's well known fact..\n	0	1
28632	sorry but it's true\n	0	1
28635	dont listen to this man @shiptarpogananacija h...	0	Neutral
28638	yeah we are all monsters..I'm gonna kill u rig...	1	0
28642	the fucking funniest thing is that fucking ame...	0	1

16984 rows × 3 columns

```
df.iloc[28642][0]
```

```
'the fucking funniest thing is that fucking american cunt telling someone is monster?!!\n'
```

Figure 40: Validated label using sentiment compound score