Natural Language Processing HW1 (Text processing)

Pardis Sadat Zahraei

October 27, 2023

Abstract

This report presents the findings from my project, in which I experimented with and explored various methods. The project focused on the task of sentiment analysis on text data, necessitating a range of preprocessing steps.

To enhance the project's robustness, I applied these methods to two distinct datasets. The first dataset, sourced from Kaggle, consisted of "Twitter" data. The second dataset comprised comments from the "Star Wars: The Last Jedi" trailer on YouTube.

The choice of two datasets was deliberate, aimed at introducing an additional layer of complexity to the project. The Twitter dataset from Kaggle included sentiment labels, enabling me to gauge the accuracy of my text processing techniques. Having these labels provided a valuable metric for evaluation.

The second dataset was selected due to the controversial nature of the movie, which resulted in divided fan opinions. This dataset differed significantly from the first one, underscoring the delicate nature of the sentiment analysis task.

Throughout this process, I employed techniques such as lemmatization, stemming, tokenization, Named Entity Recognition (NER), spell checking, bigram checking, and contradiction fixing. These were accompanied by numerous analyses and figures. Additionally, I compiled a useful list of slang terms for use in preprocessing. **clarification** Running the given notebook is estimated to take about 6-7 hours. I have made a slangs dictionary as well and it contains some curse words that I do not claim no disrespect but the text had multiple instances of these curse words and as a researcher I needed to include these curse words in my slangs list as well.

1 Tweet Sentiment Extraction

This is the dataset I have downloaded from kaggle datasets, its called ”tweet-sentiment-extraction” you can download it from kaggle competition section its name is ”train.csv” in there:

	textID	text	selected_text	sentiment
0	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
2	088c60f138	my boss is bullying me...	bullying me	negative
3	9642c003ef	what interview! leave me alone	leave me alone	negative
4	358bd9e861	Sons of ****, why couldn`t they put them on t...	Sons of ****,	negative

Figure 1: First 5 rows of the dataset

We had 27,481 tweets after removing Nan values (I removed them because they were non-numeric) we have 27,480 tweets.

1.1 Analyzing

Here is the distribution of our sentiments. Given the balanced distribution of sentiments

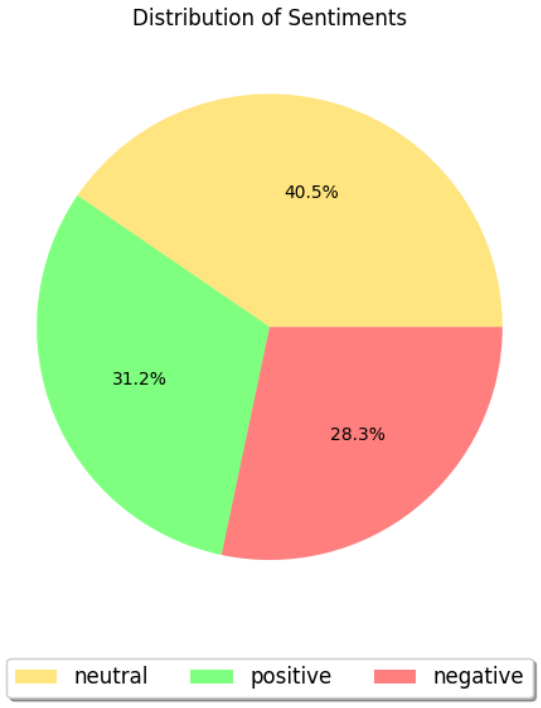


Figure 2: Analysis of Sentiment Distribution

in our data, let’s generate a word cloud to visualize the most frequent words associated with each sentiment. This will provide us with a clearer understanding of the predominant words characterizing each sentiment category.

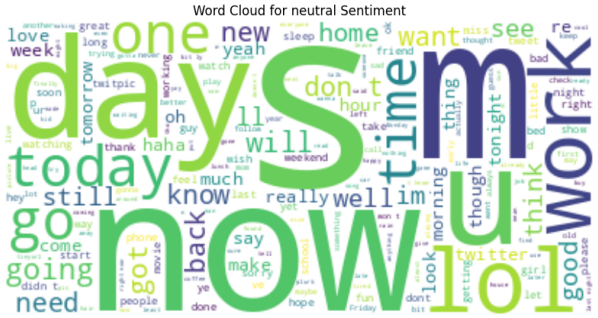


Figure 3: Analysis of neutral Sentiment words

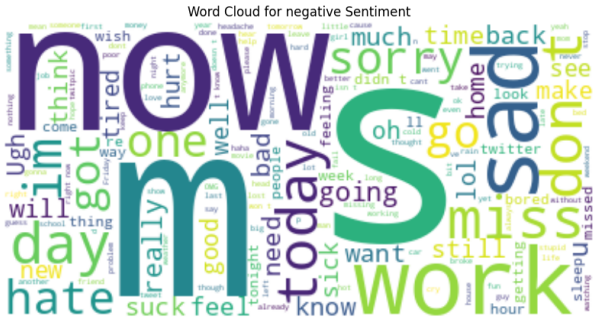


Figure 4: Analysis of negative Sentiment words

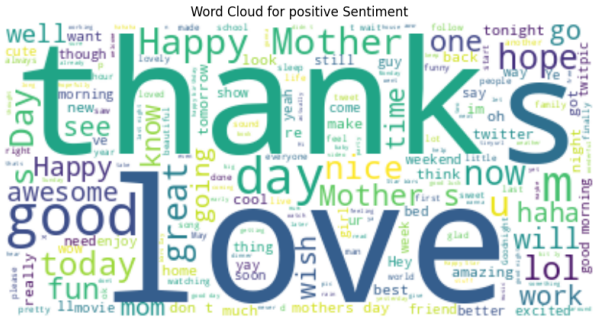


Figure 5: Analysis of positive Sentiment words

These figures show how each sentiment has their own unique words. Now lets see the top 10 frequent word for each sentiment in Figure. 6:

```
Top words for neutral sentiment:
to      4057
I       3365
the     3329
a       2438
my      1800
and     1713
i       1507
in      1501
you     1433
for     1412
dtype: int64

Top words for negative sentiment:
I       2956
to      2838
the     2301
my      1778
a       1758
i       1550
and     1442
is      1245
in      1045
it       889
dtype: int64

Top words for positive sentiment:
to      2914
the     2758
I       2481
a       2305
you     1583
and     1522
my      1354
for     1308
i       1206
is      1065
dtype: int64
```

Figure 6: Top 10 most frequent words

They may appear cluttered due to the lack of preprocessing on the text data. Steps such as normalization, stop word removal, stemming, and lemmatization are crucial for refining our data. However, let's proceed with our EDA for now. We'll examine aspects such as the length of tweets and the number of words to gain further insights into our dataset.

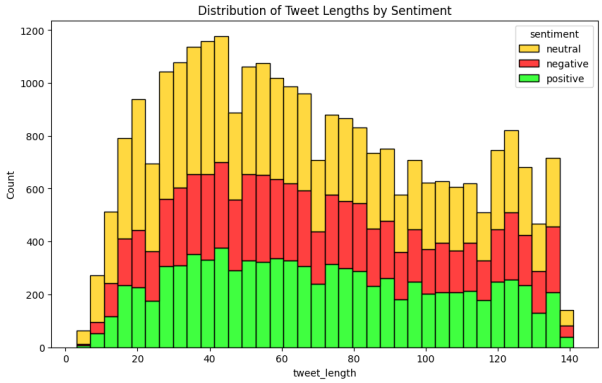


Figure 7: Distribution of Tweet Lengths by Sentiment

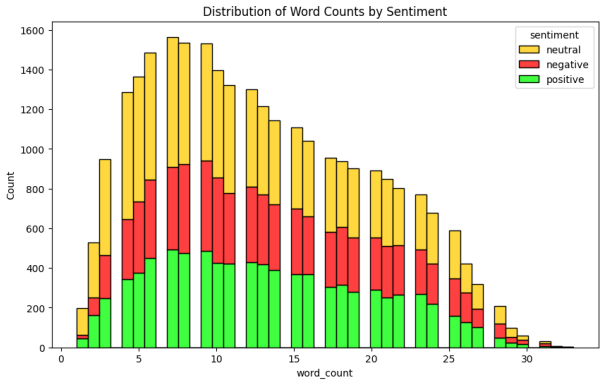


Figure 8: Distribution of Word Counts by Sentiment

Thats really interesting!! but lets see if there is a pattern in here

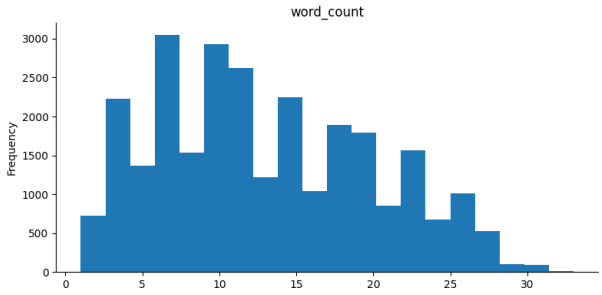


Figure 9: Word Count vs Frequency

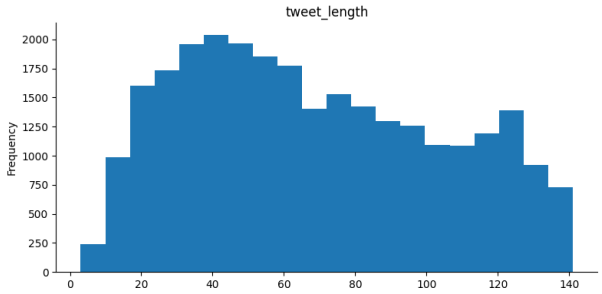


Figure 10: Tweet Length vs Frequency

Upon examining the distribution of both word count and text length in our data, we observe that the highest frequency lies within the range of 3-12 words and a text length of 20-60 characters. A combined analysis of both these metrics can provide a more holistic understanding of our text data. For instance, consider two tweets with an identical word count. If one tweet exhibits a significantly larger text length, it could suggest that this tweet predominantly uses longer words.

Here are the faceted plots for better visualization:

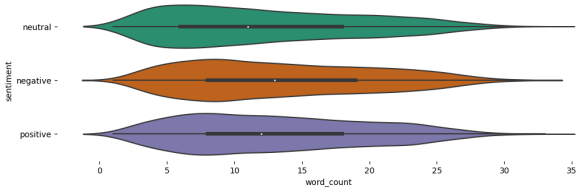


Figure 11: Word Count vs Sentiment

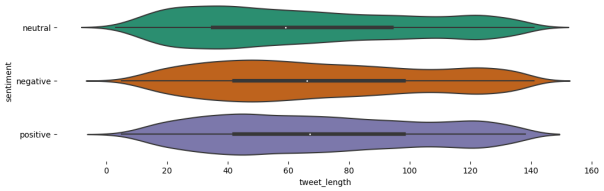


Figure 12: Tweet Length vs Sentiment

Here are the average tweet lengths:

```
Average tweet lengths for each sentiment:
 sentiment
negative    70.488112
neutral     65.206800
positive    70.419133
Name: tweet_length, dtype: float64
```

Figure 13: Tweet Length vs Sentiment

Now we will analyze the effect of Tweet Length on each sentiment:

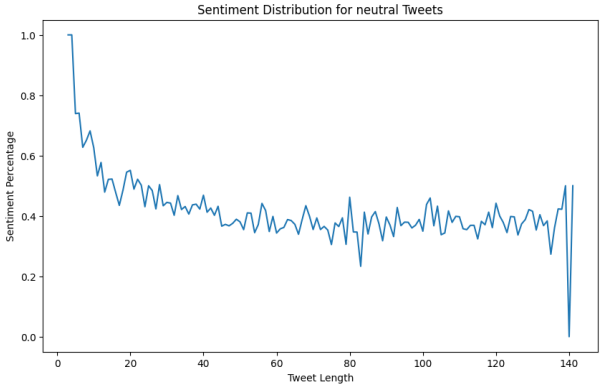


Figure 14: Tweet Length vs Sentiment

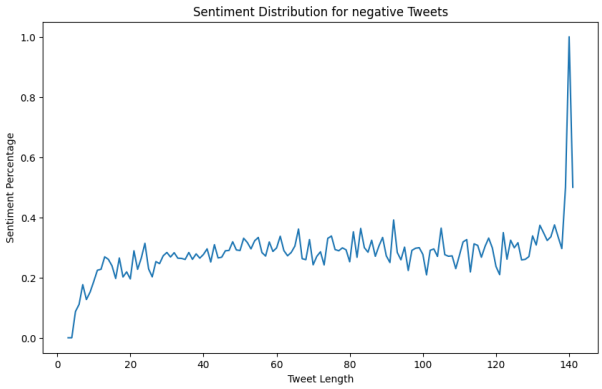


Figure 15: Tweet Length vs Sentiment

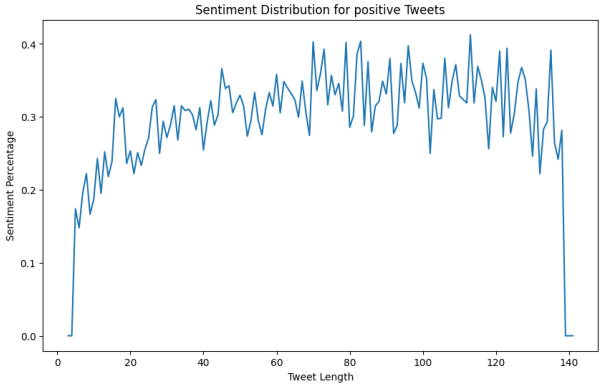


Figure 16: Tweet Length vs Sentiment

Amazing!!!
There is a pattern!!
neutral tweets seems to happen more in smaller tweets but negative tweets are a bit longer. Here is the

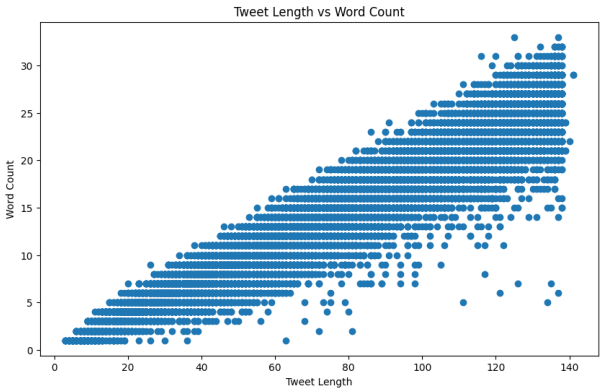


Figure 17: Tweet Length vs Word Count

There are some odd points in our data here are the points with few word counts but is a large tweet:
YESSS, FLASH IS BEING **** TONIGHT!

It.....takes.....time.....to.....open.....a.....webpage.....
THE VIDEO IS FINALLY DONE WOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO!
???? ? ??????, ?????? ????????,Basquash,K-ON,07-Ghost,Sengoku Basara,Valkyria Chronicles,RequiemForThePhantom,EdenOfTheEast.

We definitely need preprocessing :)

Let's include a Sentiment Score as well. Sentiment scores are continuous values that range from -1 to 1. A score of -1 represents extremely negative sentiment, 1 represents extremely positive sentiment, and 0 signifies neutral sentiment. This scoring system allows us to capture varying degrees of positivity or negativity in the text, a nuance that might not be possible with categorical sentiment labels alone.

A sentiment scoring tool, such as TextBlob, is capable of capturing these differences.

TextBlob provides a simple API for performing sentiment analysis. The sentiment property in TextBlob returns a namedtuple of the form Sentiment(polarity, subjectivity). The polarity score is a float within the range [-1.0, 1.0], where -1.0 represents negative sentiment, 1.0 represents positive sentiment, and 0 represents neutral sentiment. The subjectivity score is a float within the range [0.0, 1.0], where 0.0 is very objective and 1.0 is very subjective. This allows TextBlob to capture both the emotional tone (positive, negative, neutral) and the subjectivity (objective, subjective) of the text

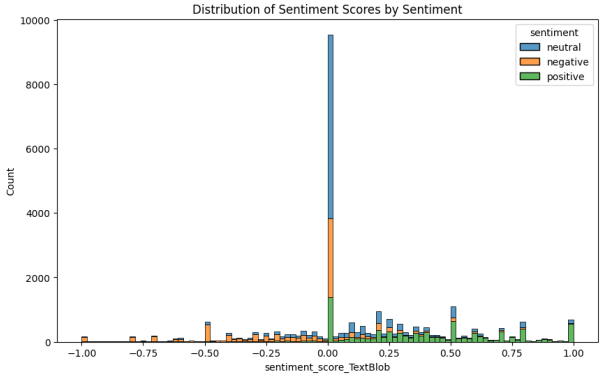


Figure 18: Distribution of Sentiment Scores by Sentiment (TextBlob) on text

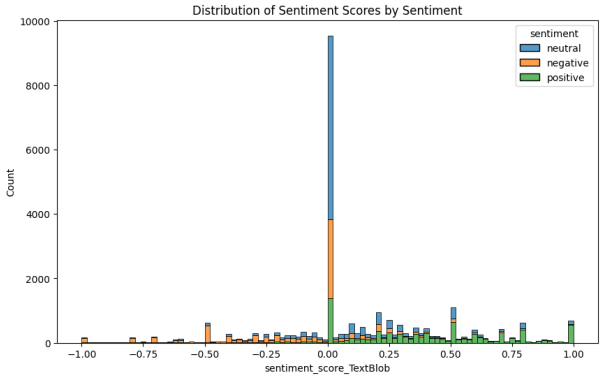


Figure 19: Distribution of Sentiment Scores by Sentiment (TextBlob) on selected text

Funny how Sentiment scores doesnt completly align with our predefined sentiments, lets

see some cases:

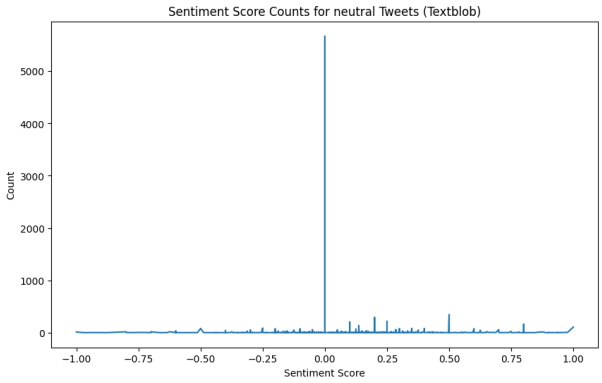


Figure 20: Sentiment Score Counts for neutral Tweets (Textblob) on text

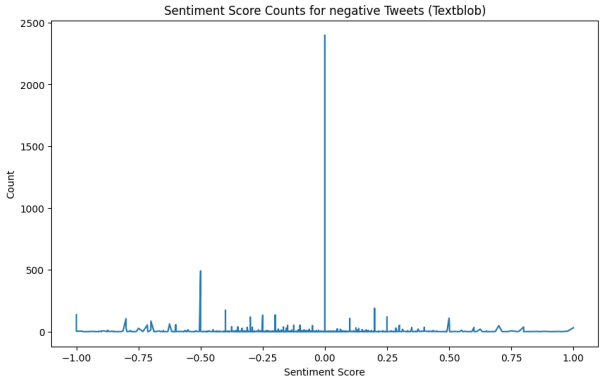


Figure 21: Sentiment Score Counts for negative Tweets (Textblob) on text

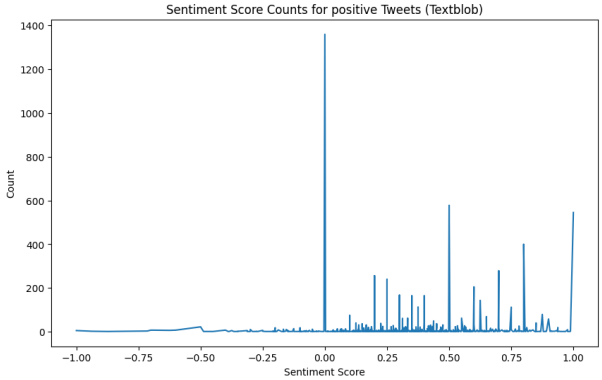


Figure 22: Sentiment Score Counts for positive Tweets (Textblob) on text

here we do it again on selected text to see if there would be any diffrence:

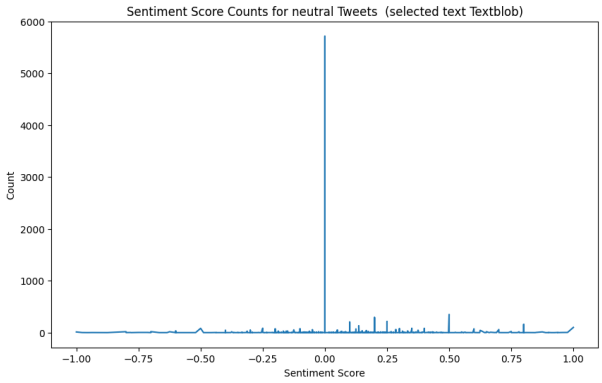


Figure 23: Sentiment Score Counts for neutral Tweets (Textblob) on selected text

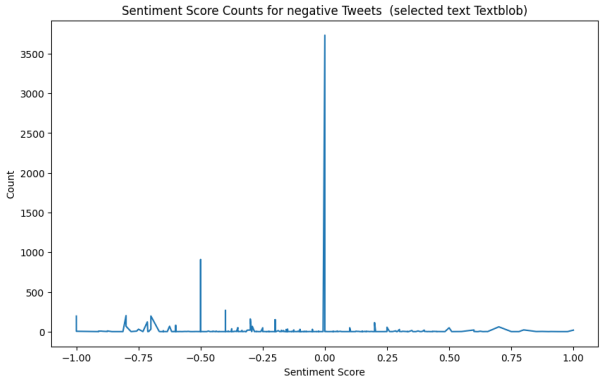


Figure 24: Sentiment Score Counts for negative Tweets (Textblob) on selected text

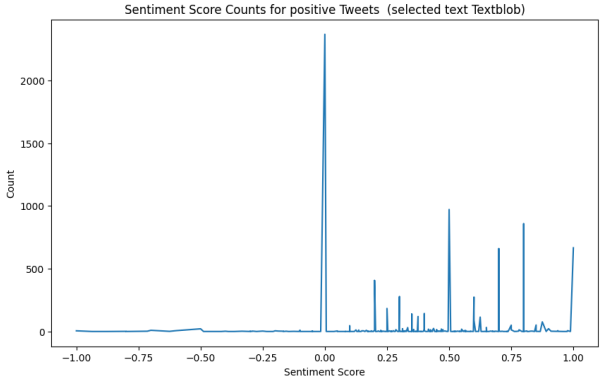


Figure 25: Sentiment Score Counts for positive Tweets (Textblob) on selected text

We can defintly see the diffrence for the selected text vs the whole text and this shows how important extracting the selected text is
We can see Sentiment scores are mostly 0 (detecting as neutral) while the dataset suggest other wise
really interesting how Sentiment score isnt working !!!

Now we will use another tool called Vader.
VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based

sentiment analysis tool. It is specifically attuned to sentiments expressed in social media, but also works well on texts from other domains. VADER is fully open-sourced under the MIT License. It's capable of understanding the sentiment of text by considering factors such as negations, punctuation, capitalization, and modifiers. For example, it can differentiate between degrees of positivity or negativity in phrases like "good" versus "very good" or "excellent". This makes it a powerful tool for analyzing sentiment in various forms of text data:

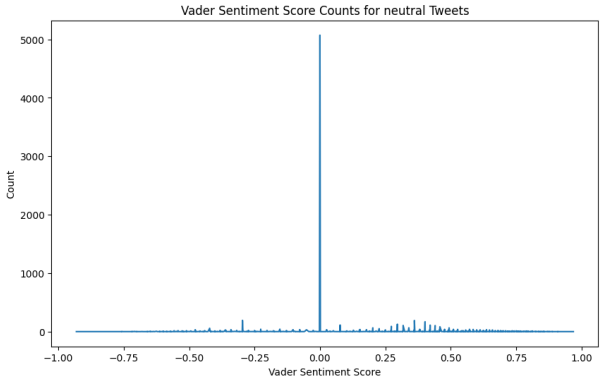


Figure 26: Sentiment Score Counts for neutral Tweets (Vader) on text

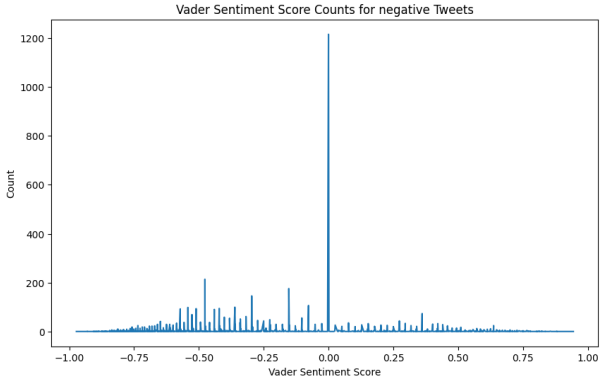


Figure 27: Sentiment Score Counts for negative Tweets (Vader) on text

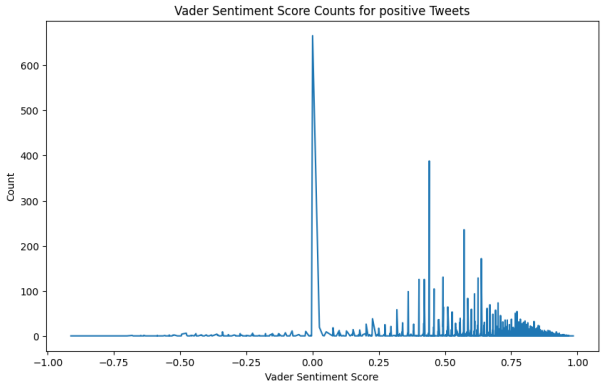


Figure 28: Sentiment Score Counts for positive Tweets (Vader) on text

Here is for the selected text:

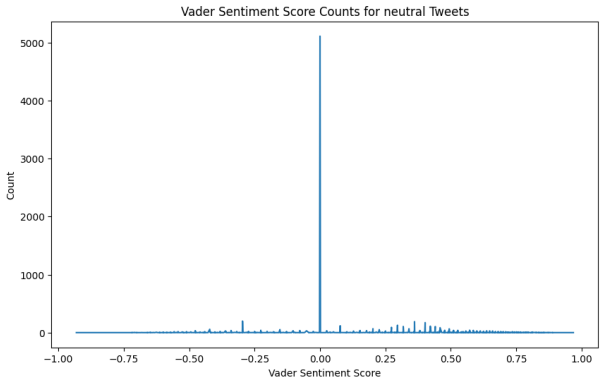


Figure 29: Sentiment Score Counts for neutral Tweets (Vader) on selected text

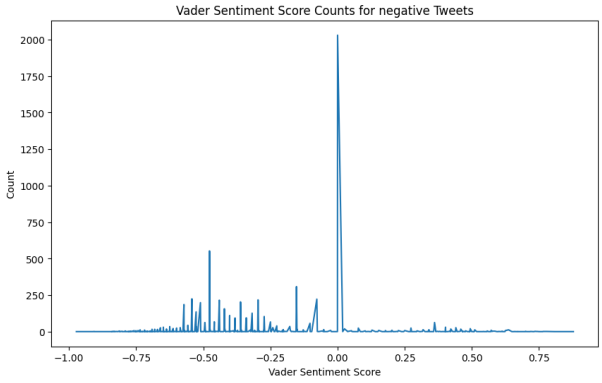


Figure 30: Sentiment Score counts for negative Tweets (Vader) on selected text

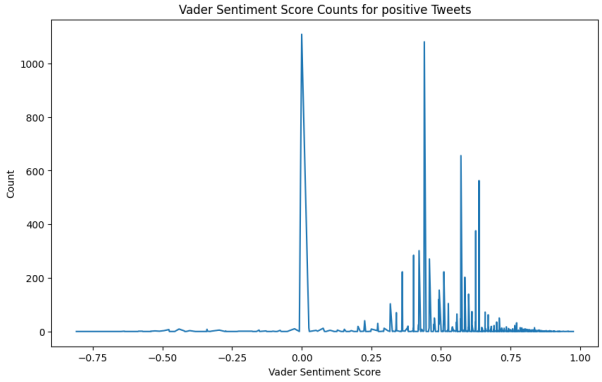


Figure 31: Sentiment Score Counts for positive Tweets (Vader) on selected text

wow!! Vader works much better than textBlob it is interesting to note that Vader is specifically designed for social media sentiment analysis. It is adept at understanding sentiments from content that typically appears on social media, such as emojis, repetitive words, and punctuations. On the other hand, TextBlob is a more general-purpose tool that can handle a variety of Natural Language Processing tasks.
Now lets dig some patterns for these two tools.

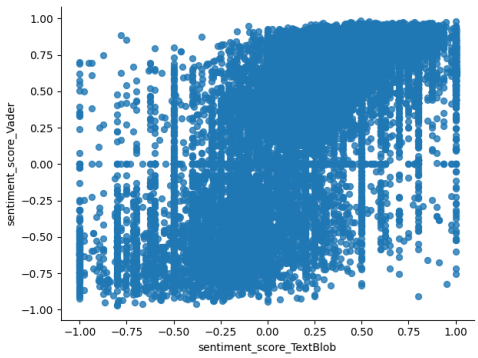


Figure 32: sentiment score TextBlob vs sentiment score Vader on text

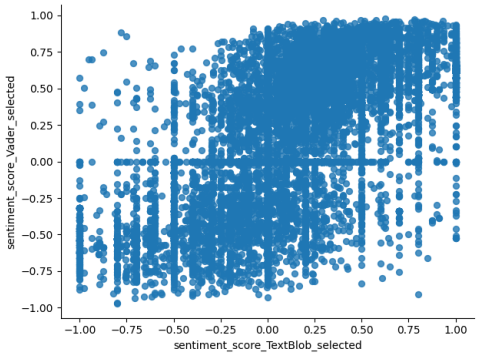


Figure 33: sentiment score TextBlob vs sentiment score Vader on selected text

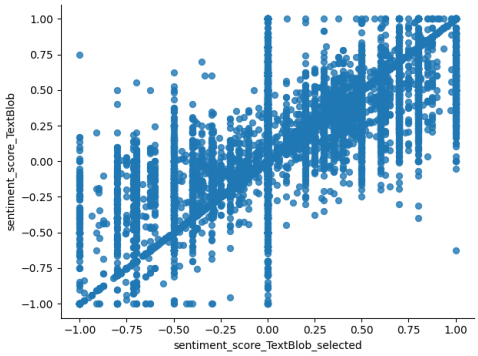


Figure 34: sentiment score TextBlob on text vs selected text

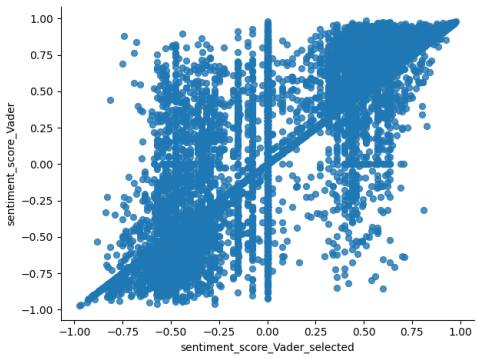


Figure 35: sentiment score Vader on text vs selected text

I am using 3 thresholds, 0.05,0.3,0.5 to detect the sentiment using these scores

- Accuracy of sentiment_score_TextBlob_0.05: 0.5902110625909752
- Accuracy of sentiment_score_TextBlob_0.3: 0.5805312954876274
- Accuracy of sentiment_score_TextBlob_0.5: 0.506513828238719
- Accuracy of sentiment_score_Vader_0.05: 0.6362809315866085
- Accuracy of sentiment_score_Vader_0.3: 0.6445414847161572
- Accuracy of sentiment_score_Vader_0.5: 0.6069141193595342
- Accuracy of sentiment_score_Vader_selected_0.05: 0.6402838427947598
- Accuracy of sentiment_score_Vader_selected_0.3: 0.6485443959243086
- Accuracy of sentiment_score_Vader_selected_0.5: 0.535844250363901
- Accuracy of sentiment_score_TextBlob_selected_0.05: 0.5635371179039301
- Accuracy of sentiment_score_TextBlob_selected_0.3: 0.6019286754002912
- Accuracy of sentiment_score_TextBlob_selected_0.5: 0.5302037845705968

we can see how the accuracy is much better around 0.3 for both tools, indicating how much the model favors neutral responses.

lets look at entry 4, Sons of ****, why couldn‘t they label it as negative? this is negative but all the analyzers put them as neutral with an exact score of 0, how is that possible? we as humans can reason **** is a curse word but the model doesnt, is replacing *** with the word ”bad” help? that would mean all *’s are bad words but as we can see below they arent. We have 1309 tweets with this symbol. Here is the distribution based on sentiment on tweets and selected part of those tweets.

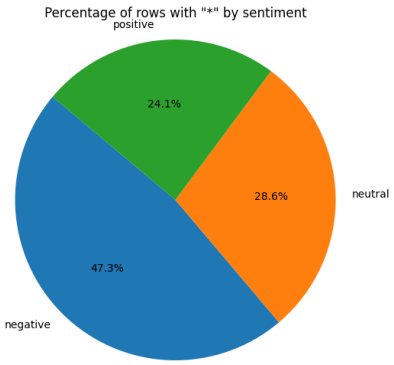


Figure 36: Percentage of texts with "*" by sentiment

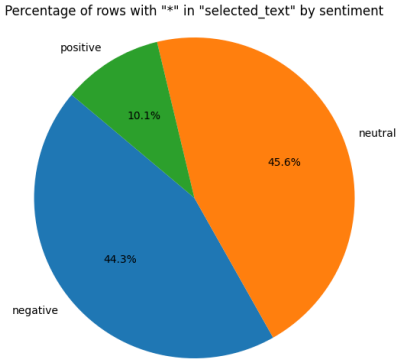


Figure 37: Percentage of selected texts with "*" by sentiment

We cant find any pattern and removing "*" with a bad word is not the case and also it doesnt show what does the words start with so we cant replace it with its counterpart as we did in next section on "star wars" dataset. Lets see the effect of punctuation.

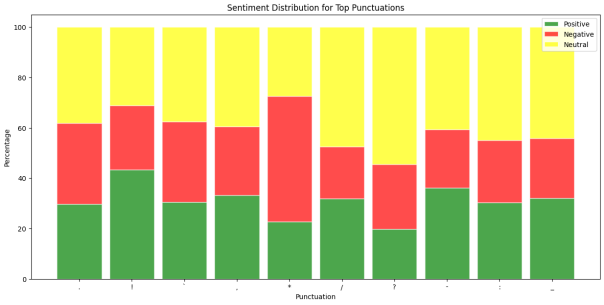


Figure 38: Sentiment Distribution for Top Punctuations

Upon examining the top 10 most frequent punctuation marks, we can ascertain that there is no discernible pattern. Therefore, it is safe to proceed with their removal.

1.2 Preprocessing

I have tried many approaches for this task and here are the results:

1.2.1 Approach 1

```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

def clean_text_A(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # Stemming/Lemmatization
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    return ' '.join(words)

df['cleaned_A_text'] = df['text'].apply(clean_text_A)
df['cleaned_A_selected_text'] = df['selected_text'].apply(clean_text_A)
```

Figure 39: Approach 1

As given comments, we do some preprocess and it cleans the data pretty good but there is a downfall. This is the entry 10:
”Both of you”
after doing the cleaning process it will be ”” so there is a decision here would we want null values or we want all rows? we know these null values are neutral so lets keep them as another neutral word because the task is sentiment analysis for example ”wall”,”unknown”,”hi”...

1.2.2 Approach 2

```python
from nltk.stem import PorterStemmer
# Initialize stemmer
stemmer = PorterStemmer()

def clean_text_B(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # Stemming
    words = [stemmer.stem(word) for word in words]

    return ' '.join(words)

# Apply the function to your text column
df['cleaned_B_text'] = df['text'].apply(clean_text_B)
df['cleaned_B_selected_text'] = df['selected_text'].apply(clean_text_B)
```

Figure 40: Approach 2

This method uses stemming

1.2.3 Approach 3

```python
def clean_text_C(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

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

    return ' '.join(words)

df['cleaned_C_text'] = df['text'].apply(clean_text_C)
df['cleaned_C_selected_text'] = df['selected_text'].apply(clean_text_C)
```

Figure 41: Approach 3

This method doesn't delete stop words

1.2.4 Approach 4

```
# Initialize stemmer
stemmer = PorterStemmer()

def clean_text_D(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Stemming
    words = [stemmer.stem(word) for word in words]

    return ' '.join(words)

# Apply the function to your text column
df['cleaned_D_text'] = df['text'].apply(clean_text_D)
df['cleaned_D_selected_text'] = df['selected_text'].apply(clean_text_D)
```

Figure 42: Approach 4

This method doesn't delete stop words and uses stemming Something is clear. Lemmatization »»>Stemming and removing stopwords is beneficial

1.2.5 Approach 5

It has still more problems here are the new modifictions
Empty Strings: If the cleaning process results in an empty string, replace it with "hi".
Spacing Issues: If there are multiple spaces between words, reduce them to a single space. For instance, convert "hi Emma" to "hi Emma".
URLs: If any URLs or 'http' text are present, they should be removed. Underscores and Slashes: Any underscores, forward slashes, or backslashes should also be eliminated.

```python
def clean_text_E(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # lemmatizer
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    cleaned_text = ' '.join(words)

    # If the cleaned text is an empty string, replace it with "hi"
    if cleaned_text.strip() == "":
        cleaned_text = "hi"

    # Removes underscores
    cleaned_text = cleaned_text.replace('_', ' ')

    # Removes multiple spaces
    cleaned_text = re.sub(' +', ' ', cleaned_text)

    return cleaned_text.strip()
df['cleaned_E_text'] = df['text'].apply(clean_text_E)
df['cleaned_E_selected_text'] = df['selected_text'].apply(clean_text_E)
```

Figure 43: Approach 5

1.2.6 Approach 6

There is still another problem. We need spell checking!!
for example: ”watr” should be replaced with ”water”. I also added another FANTASTIC feature Leetspeak dictionary, There are A LOT of cases which I had ”L0VE” instead of ”LOVE”. so I replaced them as well. Here is the code, This code takes a lot of time to run because of how sophisticated it is right now :)

```python
def clean_text_F(text):
    # Leetspeak dictionary
    leet_dict = {'0': 'o', '1': 'i', '3': 'e', '4': 'a', '5': 's', '7': 't'}

    # Replace leetspeak with corresponding letters
    for k, v in leet_dict.items():
        text = text.replace(k, v)

    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes tokens that contain a number
    words = [word for word in words if not any(char.isdigit() for char in word)]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # lemmatizer
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    cleaned_text = ' '.join(words)

    # If the cleaned text is an empty string, replace it with "hi"
    if cleaned_text.strip() == "":
        cleaned_text = "hi"

    # Removes underscores
    cleaned_text = cleaned_text.replace('_', ' ')

    # Removes multiple spaces
    cleaned_text = re.sub(' +', ' ', cleaned_text)

    # Spell checking
    spell = SpellChecker()
    corrected_words = []
    for word in cleaned_text.split():
        corrected_word = spell.correction(word)
        if corrected_word is not None:
            corrected_words.append(corrected_word)
        else:
            corrected_words.append("")  # or some default value

    return ' '.join(corrected_words)
import pandas as pd
```

Figure 44: Approach 6

1.2.7 Approach 7

****** Here comes the best idea I have had.******
Removing more than two consecutive repeating characters. This approach has all the features from before plus this

```python
# Removes more than two consecutive repeating characters
words = [re.sub(r'(.)\1{2,}', r'\1', word) for word in words]
```

Figure 45: Approach 7

1.2.8 Approach 8

What I have done before has a tiny problem, it falls for words having meaning with single char, but no worries!!! its sentiment analysis only word that matters in ”good” so we check ”god” and ”good” in the begining

```
if 'god' not in text.lower() and 'god' in cleaned_text:
    cleaned_text = cleaned_text.replace('god', 'good')
```

Figure 46: Approach 8

1.3 Analyzing

Approach 8 is the most sophisticated so far so we will analyze its results, here is the word cloud for each of the sentiments:

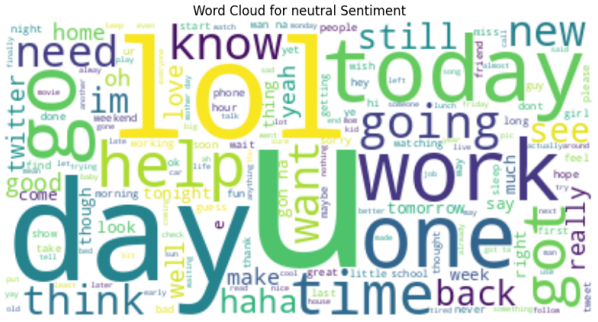


Figure 47: Word Cloud for neutral Sentiment text

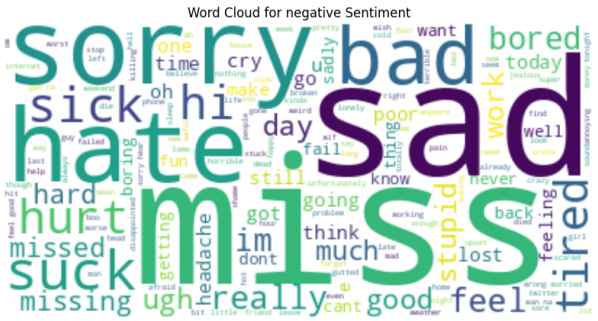


Figure 48: Word Cloud for negative Sentiment text

Figure 49: Word Cloud for positive Sentiment text

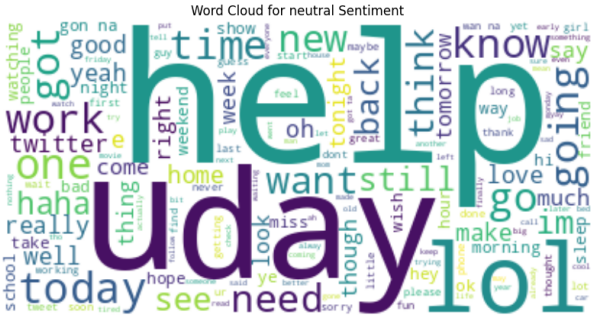


Figure 50: Word Cloud for neutral Sentiment selected text

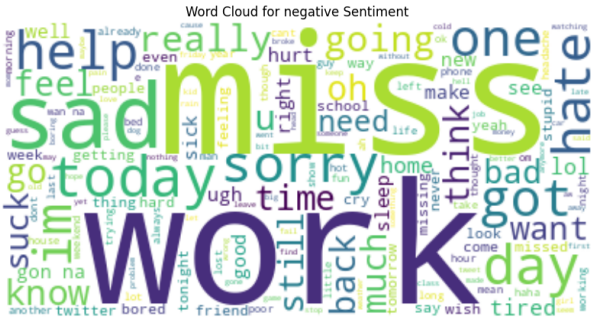


Figure 51: Word Cloud for negative Sentiment selected text

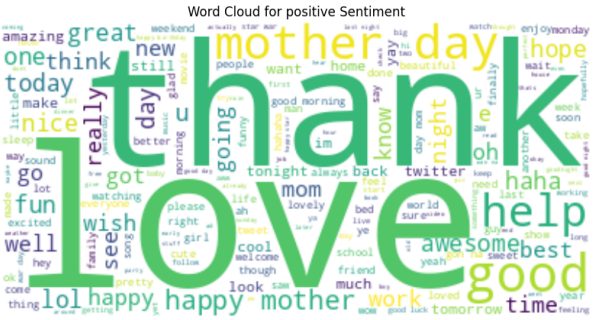


Figure 52: Word Cloud for positive Sentiment selected text

and here are the frequent words for that sentiment:

```
Top words for neutral sentiment:
to      4057
I       3365
the     3329
a       2438
my      1800
and     1713
i       1507
in      1501
you     1433
for     1412
dtype: int64

Top words for negative sentiment:
I       2956
to      2838
the     2301
my      1778
a       1758
i       1550
and     1442
is      1245
in      1045
it       889
dtype: int64

Top words for positive sentiment:
to      2914
the     2758
I       2481
a       2305
you     1583
and     1522
my      1354
for     1308
i       1206
is      1065
dtype: int64
```

Figure 53: TOP 10 for each sentiment from text

```
Top words for neutral sentiment:
day       691
get       670
help      654
go        646
i         581
got       537
work      524
lol       499
u         499
going     481
dtype: int64

Top words for negative sentiment:
like      486
get       470
go        455
miss      447
day       435
work      420
sad       402
got       361
sorry     359
im        358
dtype: int64

Top words for positive sentiment:
day       1396
good      1095
love       979
happy      862
mother     671
thanks     570
great      491
help       474
i          470
like       433
dtype: int64
```

Figure 54: TOP 10 for each sentiment from text

but there is something missing... we need to add another final approach and then we are done

1.4 Final Approach

Here I have defined a slangs dictionary which replaces some slangs ,which are not detected by our spellchecking and functions and sentiment analyzer, with their meanings. for example "lol" will be converted to "laugh out loud" or "gg" will be converted to "good game" this way "ily" which gets a sentiment of 0 now will get 1 because the analyzer will be able to detect it.
Another feature I have added is the contractions fix by NLTK. for example "you'r welcome!!" is now "you are welcome". Its also noteworthy that for stopwords I have seperated a group of negative stop words so they are not removed and no problem with our sentiment analysis.
here is the slang list:
'2day': 'today', '2nite': 'tonight', '4u': 'for you', 'b4': 'before', 'brb': 'be right back', 'btw': 'by the way', 'cya': 'see you', 'fwiw': 'for what it́s worth', 'fyi': 'for your information', 'gg': 'good game', 'gj': 'good job', 'gl': 'good luck', 'gr8': 'great', 'h8': 'hate', 'idk': 'I dont́ know', 'ily': 'I love you', 'jk': 'just kidding', 'l8r': 'later', 'lol': 'laugh out loud', 'np': 'no problem', 'omg': 'oh my god', 'plz': 'please', 'rofl': 'rolling on the floor laughing', 'thx': 'thanks',

'awt': 'adorable', 'bamf': 'bad ass mother f*****r', 'bb': 'baby', 'bf': 'boyfriend', 'bff': 'best friend forever', 'bro': 'brother', 'crush': 'person you are attracted to', 'cwot': 'cutie without the e', 'gf': 'girlfriend', 'hawt': 'hot', 'hubby': 'husband', 'ltr': 'long term relationship', 'ly': 'love you', 'qt': 'cutie', 'ship': 'relationship', 'squad': 'group of friends', 'tbf': 'to be frank', 'ttyl': 'talk to you later', 'wifey': 'wife', 'ze': 'the',

'af': 'as f***', 'biatch': 'b****', 'diss': 'disrespect', 'fml': 'f*** my life', 'fk': 'f***', 'idgaf': 'i dont́ give a f***', 'stfu': 'shut the f*** up', 'wtv': 'whatever', 'pos': 'piece of s***', 'savage': 'ruthless', 'smdh': 'shaking my darn head', 'smh': 'shaking my head', 'tf': 'the f***',

'adn': 'any day now', 'alol': 'actually laughing out loud', 'b4n': 'bye for now', 'cul8r': 'see you later', 'dgaf': 'dont́ give a f***', 'fomo': 'fear of missing out', 'fud': 'fear, uncertainty, doubt', 'fy': 'f*** yeah', 'g2g': 'got to go', 'ianal': 'I am not a lawyer', 'idc': 'I dont́ care', 'idfk': 'I dont́ f***ing know', 'idts': 'I dont́ think so', 'imo': 'in my opinion', 'irl': 'in real life', 'k': 'OK', 'lls': 'laughing like crazy', 'lmk': 'let me know', 'nochill': 'no patience', 'oc': 'of course', 'omdb': 'over my dead body', 'pita': 'pain in the a**', 'protip': 'professional tip', 'rekt': 'wrecked', 'roflcopter': 'rolling on floor laughing uncontrollably', 'rty': 'real talk yaĺl', 'trufax': 'true facts', 'tty': 'talk to you', 'wassap': 'whatś up', 'wtg': 'way to go', 'yaas': 'yes', 'ygtr': 'you got this', 'bae': 'before anyone else', 'asap': 'as soon as possible', 'extra': 'over the top', 'goals': 'aspirations', 'lit': 'exciting', 'flex': 'show off', 'snatched': 'in shape', 'boujee': 'high class', 'v': 'very', 'bet': 'ok', 'Periodt': 'end of discussion', 'and I oop-': 'expression of embarrassment',

'cray': 'crazy in a good way', 'jelly': 'jealous', 'totes': 'totally', 'adorbs': 'adorable', 'amazeballs': 'amazing', 'babe': 'endearing term for a woman', 'baller': 'cool', 'boo': 'romantic partner', 'fire': 'really good', 'legit': 'legitimate', 'litty': 'exciting', 'shipping': 'supporting a relationship', 'slay': 'do really well', 'yaaas': 'expression of excitement', 'hype': 'good hype',

'basic': 'boring', 'bruh': 'expressing frustration', 'fugly': 'extremely ugly', 'hangry': 'angry from hunger', 'ratchet': 'trashy', 'salty': 'bitter', 'thirsty': 'desperate', 'bye felicia':

'goodbye to an annoying person', 'curve': 'reject', 'drama': 'excessive reactions', 'ghost': 'stop contacting suddenly', 'janky': 'dysfunctional', 'woke': 'pretentious',
'airhead': 'stupid', 'bummer': 'disappointing', 'butthurt': 'overly offended', 'dingus': 'fool', 'douche': 'jerk', 'drama queen': 'overly dramatic', 'dumbass': 'stupid', 'epic fail': 'huge failure', 'lameoid': 'lame person', 'loser': 'unsuccessful person', 'nerd': 'overly intellectual', 'noob': 'inexperienced', 'psycho': 'crazy person', 'shady': 'untrustworthy', 'trainwreck': 'disaster', 'try-hard': 'person trying too hard', 'lmao': 'laughing my ass off', 'lmfao': 'laughing my effing ass off', 'lmdao': 'laughing my darn ass off', 'roflol': 'rolling on the floor laughing out loud', 'rotf': 'rolling on the floor', 'rotflmao': 'rolling on the floor laughing my ass off', 'wtf': 'what the heck', 'ftw': 'for the win', 'ftl': 'for the loss', 'imho': 'in my humble opinion', 'aka': 'also known as', 'fic': 'free indirect speech', 'diy': 'do it yourself', 'faq': 'frequently asked questions', 'fos': 'full of shit', 'f2f': 'face to face', 'gth': 'go to hell', 'kma': 'kiss my a**', 'g2h': 'go to hell', 'kys': 'kill yourself', 'stfd': 'shut the f*** up dumbass', 'ez': 'easy', 'roflstomped': 'utterly dominated', 'pwned': 'utterly defeated', 'pwnd': 'utterly humiliated', 'owned': 'dominated', 'n00b': 'newbie', 'stfu n00b': 'shut up newbie', 'wth': 'what the hell', 'bs': 'bullshit', 'e123': 'easy as 1 2 3', 'fol': 'free online learning', 'goat': 'greatest of all time', 'garbage': 'bad garbage'
Here is the final code:

```python
def clean_text_Final(text):
    # Remove standalone numbers
    text = re.sub(r'\b\d+\b', '', text)

    # Handle contractions
    text = contractions.fix(text)

    # Remove HTML tags
    text = BeautifulSoup(text, "html.parser").get_text()

    # Leetspeak dictionary
    leet_dict = {'0': 'o', '1': 'i', '3': 'e', '4': 'a', '5': 's', '7': 't'}
    # Replace leetspeak with corresponding letters
    for k, v in leet_dict.items():
        text = text.replace(k, v)

    # Replace slang words before tokenization
    text = " ".join(slang_dict.get(word.lower(), word.lower()) for word in text.split())

    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes tokens that contain a number
    words = [word for word in words if not any(char.isdigit() for char in word)]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))

    # List of negative words to keep
    negative_words = ['no', 'not', 'nor', 'neither', 'never', 'none']

    # Remove stop words except the negative ones
    words = [word for word in words if not (word in stop_words and word not in negative_words)]

    # Removes more than two consecutive repeating characters
    words = [re.sub(r'(.)\1{2,}', r'\1', word) for word in words]
```

Figure 55: part 1 of the final approach

```
# lemmatizer
lemmatizer = WordNetLemmatizer()
words = [lemmatizer.lemmatize(word) for word in words]
cleaned_text = ' '.join(words)

# Removes underscores
cleaned_text = cleaned_text.replace('_', ' ')

# Removes multiple spaces
cleaned_text = re.sub(' +', ' ', cleaned_text)

# Replace null
if cleaned_text.strip() == "":
    cleaned_text = "wall"

# Spell checking
spell = SpellChecker()
corrected_words = []
for word in cleaned_text.split():
    corrected_word = spell.correction(word)
    if corrected_word is not None:
        corrected_words.append(corrected_word)
    else:
        corrected_words.append(word)
cleaned_text = ' '.join(corrected_words)

# Special case for good
if 'god' not in text.lower() and 'god' in cleaned_text:
    cleaned_text = cleaned_text.replace('god', 'good')

return cleaned_text
```

Figure 56: part 2 of the final approach

1.5 Analyzing Final Approach

Here is the word clouds based on sentiments for text and selected text:

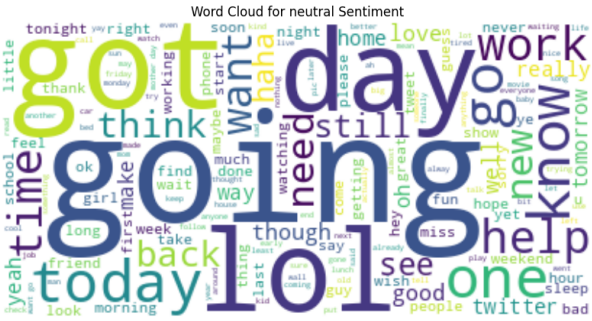


Figure 57: Word Cloud for neutral Sentiment text

Figure 58: Word Cloud for negative Sentiment text

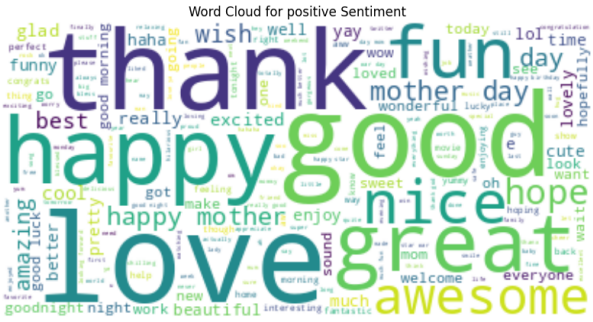


Figure 59: Word Cloud for positive Sentiment text

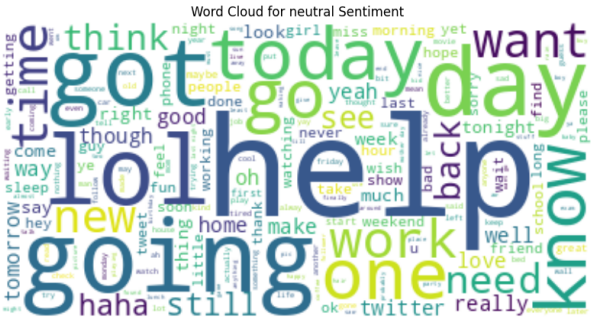


Figure 60: Word Cloud for neutral Sentiment selected text

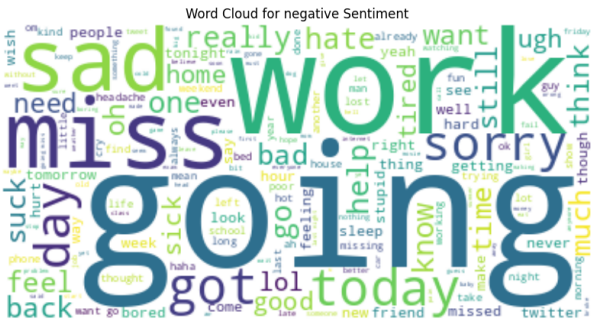


Figure 61: Word Cloud for negative Sentiment selected text

Figure 62: Word Cloud for positive Sentiment selected text

and here are the frequent words for that sentiment:

```
Top words for neutral sentiment:
not      1210
get       671
day       670
help      654
going     652
go        644
no        627
i         575
got       537
work      524
dtype: int64

Top words for negative sentiment:
not      1333
no        617
going     515
like      486
get       470
go        454
miss      444
day       425
work      420
sad       402
dtype: int64

Top words for positive sentiment:
day       1378
good      1099
love      1026
happy      862
mother     671
thanks     599
not        523
great      499
help       474
i          468
dtype: int64
```

Figure 63: TOP 10 for each sentiment from text

```
Top words for neutral sentiment:
not      1195
get       658
day       653
going     643
go        633
no        618
got       525
i         517
work      514
want      487
dtype: int64

Top words for negative sentiment:
not      614
miss     375
sad      346
sorry    303
hate     267
bad      258
no       246
suck     225
feel     173
sick     173
dtype: int64

Top words for positive sentiment:
good       853
love       808
happy      736
day        491
thanks     468
great      374
fun        293
nice       271
mother     270
awesome    256
dtype: int64
```

Figure 64: TOP 10 for each sentiment from text

So much better!!!!
like the frequent words are now relevant and the word cloud is much better now.
Redoing all the analysis before using TextBlob and Vader to see how doing the sentiment analysis on the newly processed text would affect:

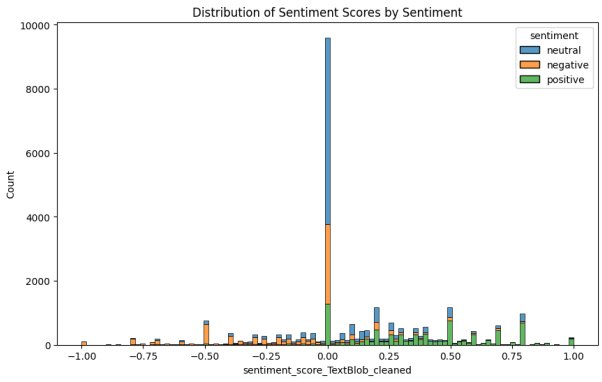


Figure 65: Distribution of Sentiment Scores by Sentiment (TextBlob) on processed text

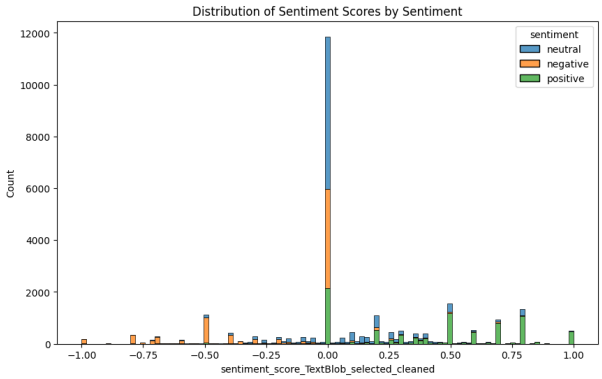


Figure 66: Distribution of Sentiment Scores by Sentiment (TextBlob) on processed selected text

It has improved so much!!!!
its really interesting!!!!
lets see more detail:

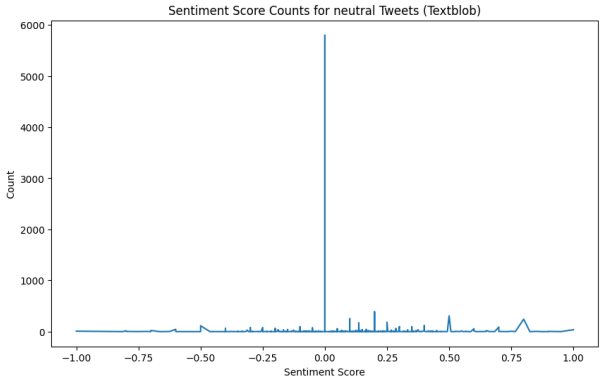


Figure 67: Sentiment Score Counts for neutral Tweets (Textblob) on processed selected text

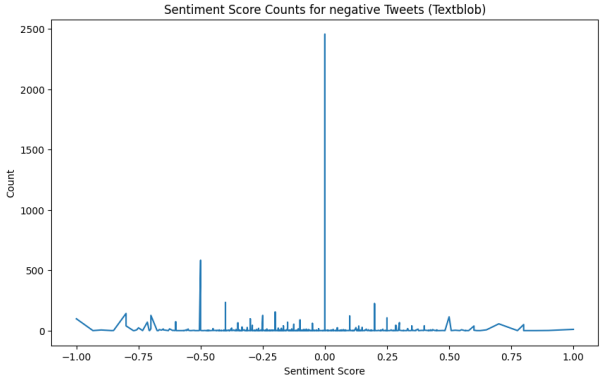


Figure 68: Sentiment Score Counts for negative Tweets (Textblob) on processed selected text

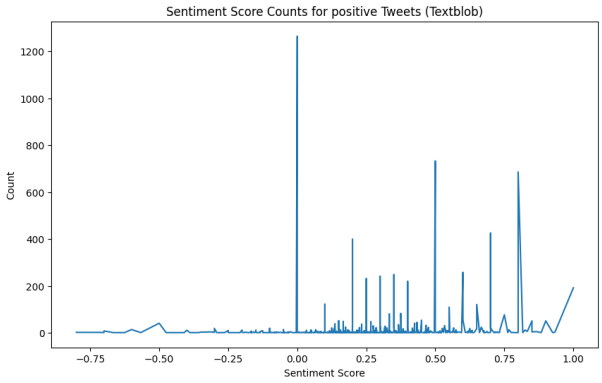


Figure 69: Sentiment Score Counts for positive Tweets (Textblob) on processed selected text

And here is the Vader results:

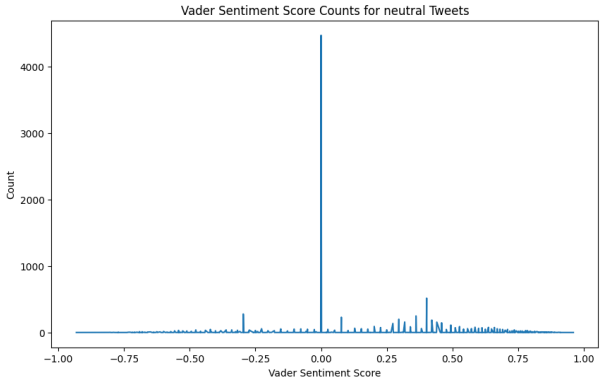


Figure 70: Sentiment Score Counts for neutral Tweets (Vader) on processed selected text

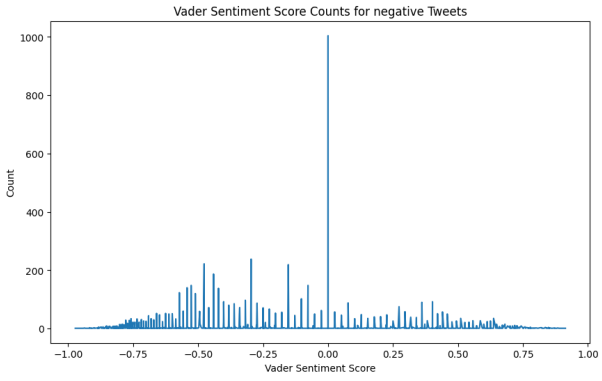


Figure 71: Sentiment Score Counts for negative Tweets (Vader) on processed selected text

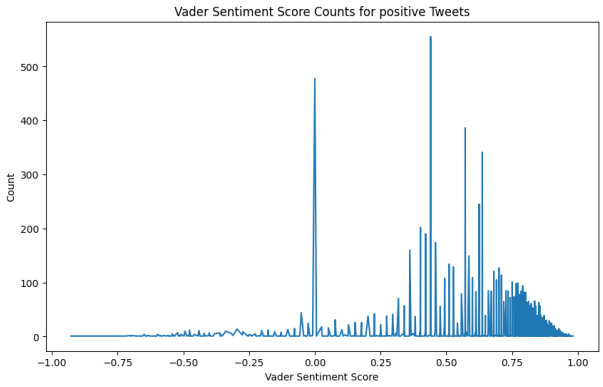


Figure 72: Sentiment Score Counts for positive Tweets (Vader) on processed selected text

Wow!!! Vader works much better and it has enhanced drastically :)
More analysis and detail are in the notebook (check that out if you want more details)
The final accuracy's are like this:

```
Accuracy of sentiment_score_TextBlob_cleaned_0.05: 0.6
Accuracy of sentiment_score_TextBlob_cleaned_0.3: 0.5826419213973799
Accuracy of sentiment_score_TextBlob_cleaned_0.5: 0.49697962154294034
Accuracy of sentiment_score_Vader_cleaned_0.05: 0.6179403202328967
Accuracy of sentiment_score_Vader_cleaned_0.3: 0.6269286754002911
Accuracy of sentiment_score_Vader_cleaned_0.5: 0.5947962154294032
```

Figure 73: Accuracy for text after processing

```
Accuracy of sentiment_score_TextBlob_selected_cleaned_0.05: 0.574745269286754
Accuracy of sentiment_score_TextBlob_selected_cleaned_0.3: 0.6127365356622999
Accuracy of sentiment_score_TextBlob_selected_cleaned_0.5: 0.5303857350800583
Accuracy of sentiment_score_Vader_selected_cleaned_0.05: 0.6349708879184862
Accuracy of sentiment_score_Vader_selected_cleaned_0.3: 0.6508005822416303
Accuracy of sentiment_score_Vader_selected_cleaned_0.5: 0.5348981077147016
```

Figure 74: Accuracy for selected text after processing

1.6 Conclusion

So far we have used many approaches, libraries, Techniques but Why hasnt the final result changed that much? Well here is the deal.
1) The tools I have used would all do the preprocessing as well. So preprocessing the text would make a drastic change if I was to train my own LLM or Neural Network but although it makes these pretrained tools a bit more efficent we didnt need preprocessing for them that much.
2) The selected text and the assigned sentiments werent 100% accurent. The human labelor had some errors so our labeled data needed modifications as well.
3) Overall the final approach I had was comprehensive it had all the necessities needed and could be used on other texts as well. on the next section we will see how we are foing to do it on a completely new dataset.

2 Star Wars: The Last Jedi

"Star Wars: The Last Jedi" is a highly controversial movie due to its subversion of audience expectations and the polarizing reactions it elicited. The film was praised by critics, holding around a 91% Rotten Tomatoes score, but the audience disagreed greatly, giving it 49%. This disparity in sentiment makes it an interesting subject for sentiment analysis. The film's handling of characters like Luke Skywalker and plot threads from "Star Wars: The Force Awakens" sparked debates. Furthermore, the film's perceived dismissal of the Jedi Order and its principles caused further controversy. These elements contribute to a wide range of sentiments, from admiration to disappointment, making "The Last Jedi" a rich dataset for sentiment analysis.

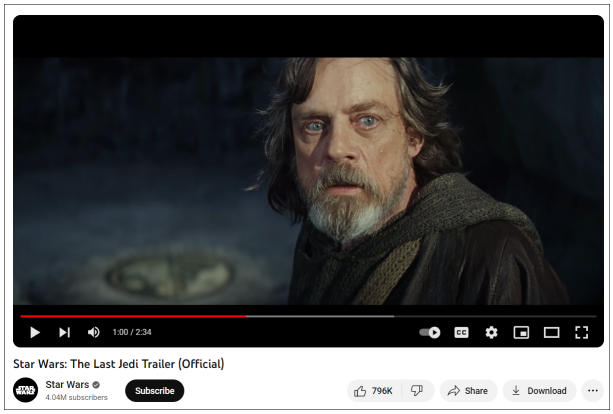


Figure 75: Star Wars he Last Jedi

Thats why I have crawled the data from the youtube comments of this movie, here is the code snippet:

```
"""
function scrapeCommentsWithoutReplies(){
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var result=[['Name','Comment','Time','Likes','Reply Count']];
  var vid = ss.getSheets()[0].getRange(1,1).getValue();
  var nextPageToken=undefined;

  while(1){
    var data = YouTube.CommentThreads.list('snippet', {videoId: vid, maxResults: 100, pageToken: nextPageToken})
    nextPageToken=data.nextPageToken
    //console.log(nextPageToken);
    for (var row=0; row<data.items.length; row++) {
      result.push([data.items[row].snippet.topLevelComment.snippet.authorDisplayName,
                   data.items[row].snippet.topLevelComment.snippet.textDisplay,
                   data.items[row].snippet.topLevelComment.snippet.publishedAt,
                   data.items[row].snippet.topLevelComment.snippet.likeCount,
                   data.items[row].snippet.totalReplyCount]);
    }
    if(nextPageToken =="" || typeof nextPageToken === "undefined"){
      break;
    }
  }
var newSheet=ss.insertSheet(ss.getNumSheets())
newSheet.getRange(1, 1,result.length,5).setValues(result)

}
"""
```

Figure 76: Code snippet for crawling

Here are some examples from the dataset:

Name	Comment	Likes
CSWF	Garbage	2
MikaãƒŸã,	Luke skywalker fought an inter dimensional incarnation of chaos i	0
MÃ©tis m	Just remembering how naive I was thinking â€œthat was fun. Eve	1
VINIÃ‡ÃƒC	Trash.ðŸ˜¢	2
Dean Espo	tbh this trailer was pretty fantastic, it has subtle hints of things in	0
Jackson Su	Gotta give a hand to the people who make the trailers cuz this is s	3
Mason Mc	I was so hyped for this in the months leading up to it!!! It was disa	0
DragonZal	The movie that ended up being the second worst star wars movie	1
gobogoo	best star wars movie in like 30 years	2
ÐÐ¸ÐºÐ¾Ð¼Ð	This movie is part of my childhood!	2
ROGUE TH	Look at this So much potenial 	0
louis winds	Never thought this of all things would break the interet	0
Enzoroni's	The first shot of the trailer is one of my favourites ever	0

Figure 77: Examples from the star wars crawled dataset

it had 50k comments to reduce the size I randomly deleted 25k rows, having a dataset of size 25038, after deleting NANs giving us 25031 comments.

2.1 Analyzing

Here are the top 10 frequent words and the word cloud:

Figure 78: Word cloud

Rank	Word	Frequency
1	REYLO	20127
2	the	13091
3	I	6657
4	to	6596
5	is	5780
6	and	5007
7	a	4852
8	this	4299
9	of	3972
10	in	3280

Figure 79: Top 10 frequent words

From the above analysis we see how important stop word removal and other techniques are, Here is more delicate analysis based on comment length and word count:

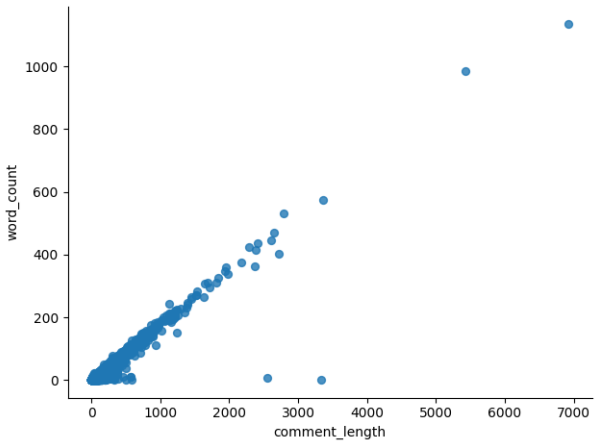


Figure 80: word count vs current length

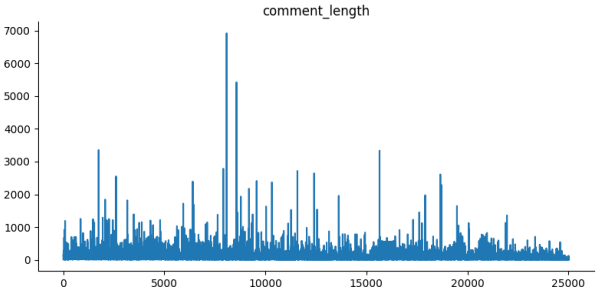


Figure 81: comment length distribution

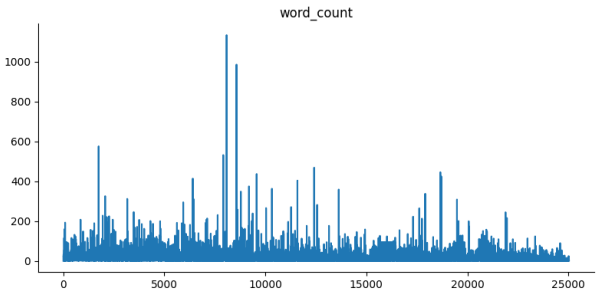


Figure 82: word count distribution

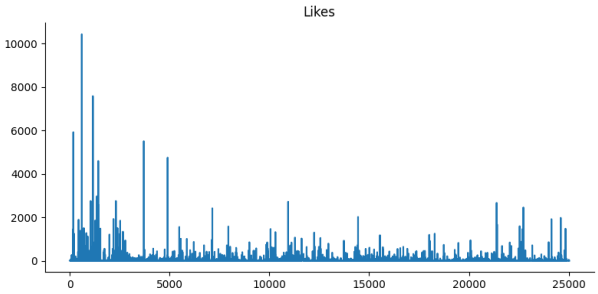


Figure 83: likes distribution

Here is the histogram plot for each sentiment score using TextBlob, (Details about this tool is given in section1) :

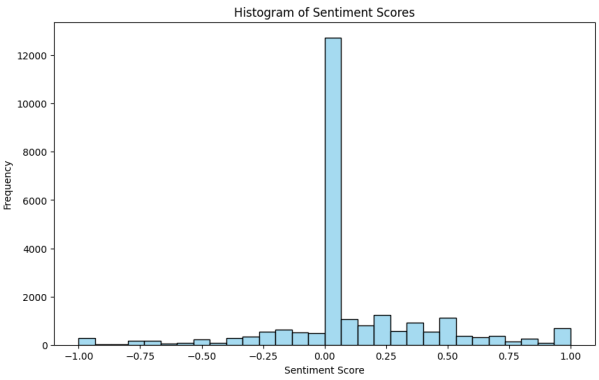


Figure 84: Histogram of Sentiment Scores TextBlob

Here is the Histogram plot for Vader which works much better:

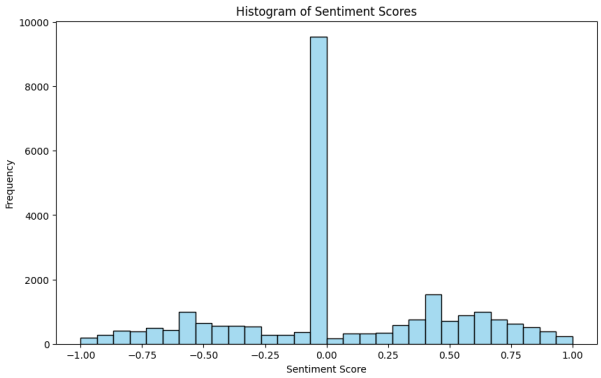


Figure 85: Histogram of Sentiment Scores Vader

This is the diffrence between TextBlob and Vader distribution

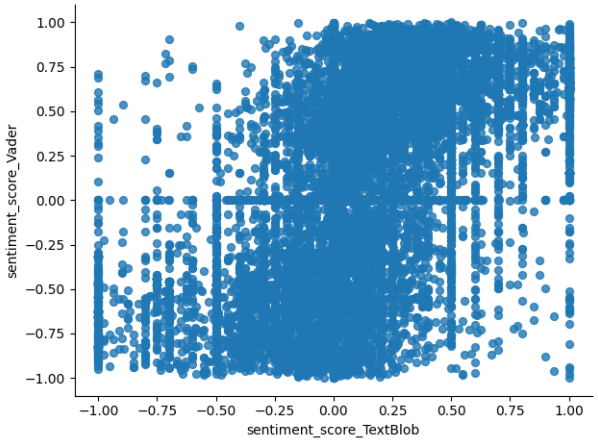


Figure 86: sentiment score TextBlob vs sentiment score Vader

2.2 Preprocessing

Lets extract some features and patterns from our data (it is on Vader with 0.05 threshold):

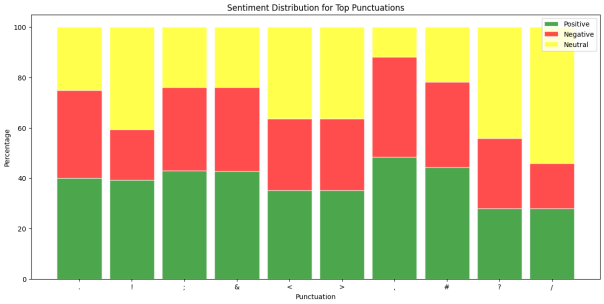


Figure 87: Sentiment Distribution for Top Punctuations

2.3 Approaches

Here like section 1 we have those previous approaches + 3 new approaches which are specialized for this data: (All the code are explained in comments and the code is clean)

2.3.1 Approach1

```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

def clean_text_A(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    return ' '.join(words)

df['cleaned_A_text'] = df['Comment'].apply(clean_text_A)
```

Figure 88: Approach1

2.3.2 Approach2

```python
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')

def clean_text_A1(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # Stemming/Lemmatization
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    return ' '.join(words)

df['cleaned_A1_text'] = df['Comment'].apply(clean_text_A1)
```

Figure 89: Approach2

2.3.3 Approach3

```python
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

def clean_text_B(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # Stemming
    words = [stemmer.stem(word) for word in words]

    return ' '.join(words)
```

Figure 90: Approach3

2.3.4 Approach4

```python
def clean_text_C(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Stemming/Lemmatization
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    return ' '.join(words)

df['cleaned_C_text'] = df['Comment'].apply(clean_text_C)
```

Figure 91: Approach4

2.3.5 Approach5

```
# Initialize stemmer
stemmer = PorterStemmer()

def clean_text_D(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Stemming
    words = [stemmer.stem(word) for word in words]

    return ' '.join(words)
df['cleaned_D_text'] = df['Comment'].apply(clean_text_D)
```

Figure 92: Approach5

2.3.6 Approach6

```
def clean_text_E(text):
    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # lemmatizer
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    cleaned_text = ' '.join(words)

    # If the cleaned text is an empty string, replace it with "hi"
    if cleaned_text.strip() == "":
        cleaned_text = "hi"

    # Removes underscores
    cleaned_text = cleaned_text.replace('_', ' ')

    # Removes multiple spaces
    cleaned_text = re.sub(' +', ' ', cleaned_text)

    return cleaned_text.strip()
df['cleaned_E_text'] = df['Comment'].apply(clean_text_E)
```

Figure 93: Approach6

2.3.7 Approach7

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
import string
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from spellchecker import SpellChecker
from tqdm import tqdm
tqdm.pandas()
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from spellchecker import SpellChecker
```

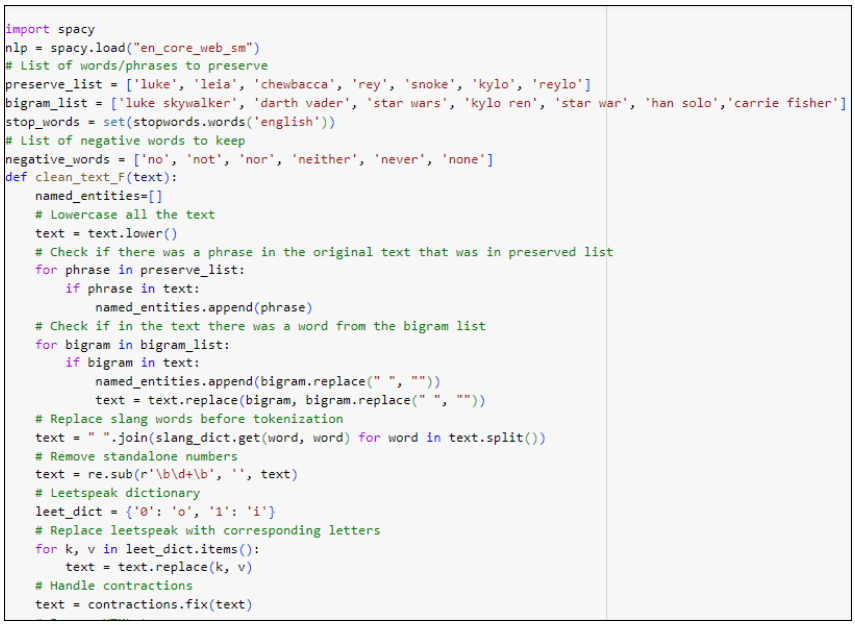
Figure 94: necessary libraries

```python
def clean_text_P(text):
    # Leetspeak dictionary
    leet_dict = {'0': 'o', '1': 'i', '3': 'e', '4': 'a', '5': 's', '7': 't'}

    # Replace leetspeak with corresponding letters
    for k, v in leet_dict.items():
        text = text.replace(k, v)

    # Tokenizes
    words = word_tokenize(text)

    # Make text lowercase
    words = [word.lower() for word in words]

    # Removes hyperlinks
    words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]

    # Removes emojis and other special characters
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Remove punctuation
    words = [re.sub(r'[^\w\s]', '', word) for word in words]

    # Removes numbers
    words = [word for word in words if not word.isdigit()]

    # Removes tokens that contain a number
    words = [word for word in words if not any(char.isdigit() for char in word)]

    # Removes useless words "stopwords"
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if not word in stop_words]

    # lemmatizer
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]

    # Removes more than two consecutive repeating characters
    words = [re.sub(r'(.)\1{2,}', r'\1', word) for word in words]

    cleaned_text = ' '.join(words)

    # If the cleaned text is an empty string, replace it with "hi"
    if cleaned_text.strip() == "":
        cleaned_text = "hi"

    # Removes underscores
    cleaned_text = cleaned_text.replace('_', ' ')

    # Removes multiple spaces
    cleaned_text = re.sub(' +', ' ', cleaned_text)

    # Spell checking
    spell = SpellChecker()
    corrected_words = []
    for word in cleaned_text.split():
        corrected_word = spell.correction(word)
        if corrected_word is not None:
            corrected_words.append(corrected_word)
        else:
            corrected_words.append(word)
    cleaned_text = ' '.join(corrected_words)
    if 'god' not in text.lower() and 'god' in cleaned_text:
        cleaned_text = cleaned_text.replace('god', 'good')
    if cleaned_text.strip() == "":
        cleaned_text = "wall"
    return cleaned_text
```

Figure 95: Approach7

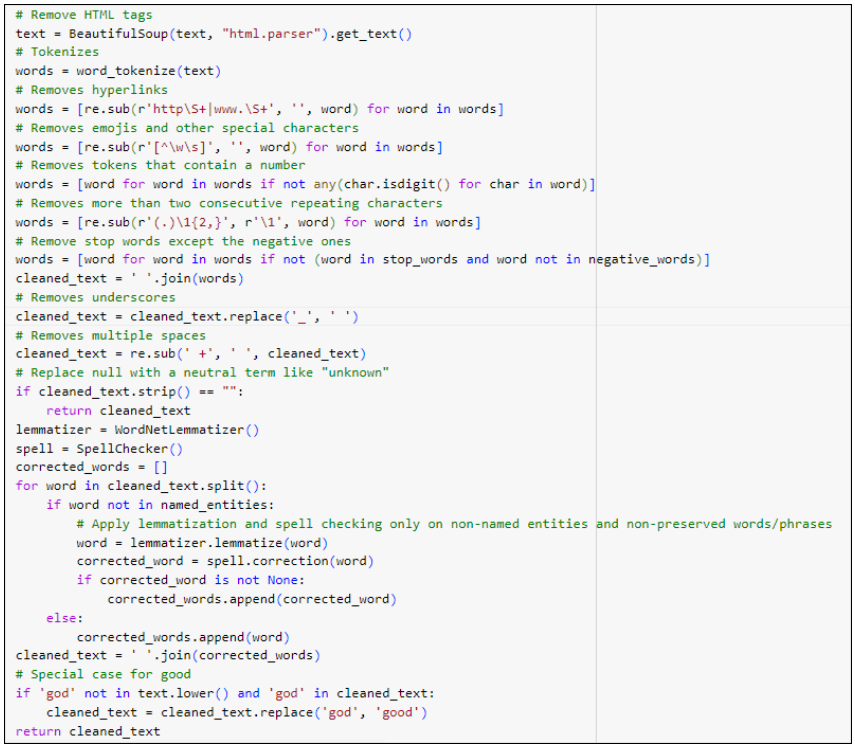
2.3.8 Approach8

This approach is the *BEST* I have come up with, it has some star wars names that would be eliminated by spell checker and also I have a bigram of important phrases for example: ”star wars” should be detected as a name but when I didnt have that bigram check it would have give ”star, war” in the result and ”war” has -1 sentiment so it would be a negative text but its neutral so this bigram helps and also instead of having fill the unknown word with the phrases like ”wall” I dont put that word in the sentence and I have changed the order of the things right now.



```
import spacy
nlp = spacy.load("en_core_web_sm")
# List of words/phrases to preserve
preserve_list = ['luke', 'leia', 'chewbacca', 'rey', 'snoke', 'kylo', 'reylo']
bigram_list = ['luke skywalker', 'darth vader', 'star wars', 'kylo ren', 'star war', 'han solo','carrie fisher']
stop_words = set(stopwords.words('english'))
# List of negative words to keep
negative_words = ['no', 'not', 'nor', 'neither', 'never', 'none']
def clean_text_F(text):
    named_entities=[]
    # Lowercase all the text
    text = text.lower()
    # Check if there was a phrase in the original text that was in preserved list
    for phrase in preserve_list:
        if phrase in text:
            named_entities.append(phrase)
    # Check if in the text there was a word from the bigram list
    for bigram in bigram_list:
        if bigram in text:
            named_entities.append(bigram.replace(" ", ""))
            text = text.replace(bigram, bigram.replace(" ", ""))
    # Replace slang words before tokenization
    text = " ".join(slang_dict.get(word, word) for word in text.split())
    # Remove standalone numbers
    text = re.sub(r'\b\d+\b', '', text)
    # Leetspeak dictionary
    leet_dict = {'0': 'o', '1': 'i'}
    # Replace leetspeak with corresponding letters
    for k, v in leet_dict.items():
        text = text.replace(k, v)
    # Handle contractions
    text = contractions.fix(text)
```

Figure 96: Approach8 part 1



```
# Remove HTML tags
text = BeautifulSoup(text, "html.parser").get_text()
# Tokenizes
words = word_tokenize(text)
# Removes hyperlinks
words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]
# Removes emojis and other special characters
words = [re.sub(r'[^\w\s]', '', word) for word in words]
# Removes tokens that contain a number
words = [word for word in words if not any(char.isdigit() for char in word)]
# Removes more than two consecutive repeating characters
words = [re.sub(r'(.)\1{2,}', r'\1', word) for word in words]
# Remove stop words except the negative ones
words = [word for word in words if not (word in stop_words and word not in negative_words)]
cleaned_text = ' '.join(words)
# Removes underscores
cleaned_text = cleaned_text.replace('_', ' ')
# Removes multiple spaces
cleaned_text = re.sub(' +', ' ', cleaned_text)
# Replace null with a neutral term like "unknown"
if cleaned_text.strip() == "":
    return cleaned_text
lemmatizer = WordNetLemmatizer()
spell = SpellChecker()
corrected_words = []
for word in cleaned_text.split():
    if word not in named_entities:
        # Apply lemmatization and spell checking only on non-named entities and non-preserved words/phrases
        word = lemmatizer.lemmatize(word)
        corrected_word = spell.correction(word)
        if corrected_word is not None:
            corrected_words.append(corrected_word)
    else:
        corrected_words.append(word)
cleaned_text = ' '.join(corrected_words)
# Special case for good
if 'god' not in text.lower() and 'god' in cleaned_text:
    cleaned_text = cleaned_text.replace('god', 'good')
return cleaned_text
```

Figure 97: Approach8 part 2

2.3.9 Approach9

using NLP and NER from spacy is a high computational function so it would be expensive to use it for the whole text and besides we have the phrases so I didnt use this approach previously but I am doing it now and instead of removing the unkown I will put the ”unknown” instead. its still a good approach just I prefer the previous one because it was cheaper.

```
import spacy
nlp = spacy.load("en_core_web_sm")
# List of words/phrases to preserve
preserve_list = ['luke', 'leia', 'chewbacca', 'rey', 'snoke', 'kylo', 'reylo']
bigram_list = ['luke skywalker', 'darth vader', 'star wars', 'kylo ren', 'star war', 'han solo','carrie fisher']
# Removes useless words "stopwords"
stop_words = set(stopwords.words('english'))
nlp = spacy.load("en_core_web_sm")
# List of negative words to keep
negative_words = ['no', 'not', 'nor', 'neither', 'never', 'none']
def clean_text_f(text):
    doc = nlp(text)
    named_entities = [str(i) for i in doc.ents]
    # Lowercase all the text
    text = text.lower()
    # Check if there was a phrase in the original text that was in preserved list
    for phrase in preserve_list:
        if phrase in text:
            named_entities.append(phrase)
    # Check if in the text there was a word from the bigram list
    for bigram in bigram_list:
        if bigram in text:
            named_entities.append(bigram.replace(" ", ""))
            text = text.replace(bigram, bigram.replace(" ", ""))
    # Replace slang words before tokenization
    text = " ".join(slang_dict.get(word, word) for word in text.split())
    # Remove standalone numbers
    text = re.sub(r'\b\d+\b', '', text)
    # Leetspeak dictionary
    leet_dict = {'0': 'o', '1': 'i'}
    # Replace leetspeak with corresponding letters
    for k, v in leet_dict.items():
        text = text.replace(k, v)
    # Handle contractions
    text = contractions.fix(text)
    # Remove HTML tags
    text = BeautifulSoup(text, "html.parser").get_text()
    # Tokenizes
    words = word_tokenize(text)
```

Figure 98: Approach9 part 1

```
# Removes hyperlinks
words = [re.sub(r'http\S+|www.\S+', '', word) for word in words]
# Removes emojis and other special characters
words = [re.sub(r'[^\w\s]', '', word) for word in words]
# Removes tokens that contain a number
words = [word for word in words if not any(char.isdigit() for char in word)]
# Removes more than two consecutive repeating characters
words = [re.sub(r'(.)\1{2,}', r'\1', word) for word in words]
# Remove stop words except the negative ones
words = [word for word in words if not (word in stop_words and word not in negative_words)]
cleaned_text = ' '.join(words)

# Removes underscores
cleaned_text = cleaned_text.replace('_', ' ')

# Removes multiple spaces
cleaned_text = re.sub(' +', ' ', cleaned_text)

# Replace null with a neutral term like "unknown"
if cleaned_text.strip() == "":
    return "unknown"
lemmatizer = WordNetLemmatizer()
spell = SpellChecker()
corrected_words = []

for word in cleaned_text.split():
    if word not in named_entities:
        # Apply lemmatization and spell checking only on non-named entities and non-preserved words/phrases
        word = lemmatizer.lemmatize(word)
        corrected_word = spell.correction(word)
        if corrected_word is not None:
            corrected_words.append(corrected_word)
        else:
            corrected_words.append("unknown")
    else:
        corrected_words.append(word)

cleaned_text = ' '.join(corrected_words)
# Special case for good
if 'god' not in text.lower() and 'god' in cleaned_text:
    cleaned_text = cleaned_text.replace('god', 'good')
return cleaned_text
```

Figure 99: Approach9 part 2

2.4 Analyzing

Both approach 8 & 9 are great but because we delete the unknown terms, some of them would be null so We are sticking with method 9; (for example they are links)

```
NaN values in the DataFrame:
Name                                0
Comment                             0
Likes                               0
comment_length                      0
word_count                          0
sentiment_score_TextBlob            0
sentiment_score_Vader               0
sentiment_score_TextBlob_0.05       0
sentiment_score_TextBlob_0.3        0
sentiment_score_TextBlob_0.5        0
sentiment_score_Vader_0.05          0
sentiment_score_Vader_0.3           0
sentiment_score_Vader_0.5           0
cleaned_A_text                    132
cleaned_A1_text                   185
cleaned_B_text                    185
cleaned_C_text                    132
cleaned_D_text                    132
cleaned_E_text                      0
cleaned_P_text                      0
cleaned_Final_star_text           585
cleaned_Final_star_unknown_text     0
dtype: int64
```

Figure 100: Nullity of data

Here is the word cloud, it has the word "unknown" in it as well which are for those none detectable words.

Figure 101: WordCloud

Here is the wordCloud and top10 word as well which we can see how great it is now:

Rank	Word	Frequency
1	reylo	20249
2	movie	4288
3	unknown	3157
4	not	2921
5	trailer	2651
6	starwars	2477
7	rey	2241
8	like	2075
9	luke	1805
10	good	1732

Figure 102: WordCloud

Rank	Word	Frequency
1	reylo	20249
2	movie	4288
3	unknown	3157
4	not	2921
5	trailer	2651
6	starwars	2477
7	rey	2241
8	like	2075
9	luke	1805
10	good	1732

Figure 103: WordCloud

Rank	Word	Frequency
1	reylo	20249
2	movie	4288
3	unknown	3157
4	not	2921
5	trailer	2651
6	starwars	2477
7	rey	2241
8	like	2075
9	luke	1805
10	good	1732

Figure 104: TOP10

Now we will do a sentiment score and see the results for TextBlob and Vader:

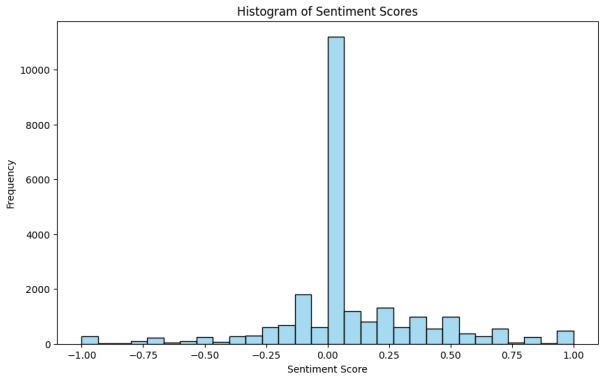


Figure 105: TextBlob

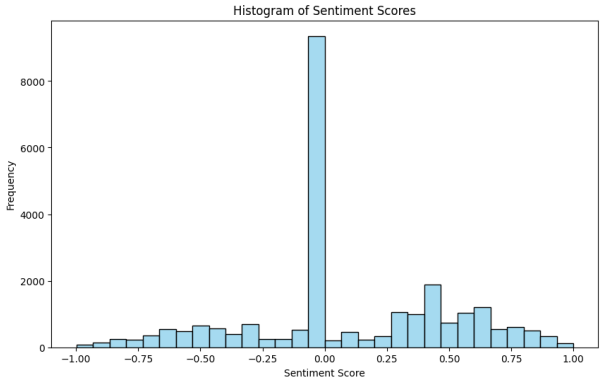


Figure 106: VADER

Finally here is the distribution of sentiments using Vader and 0.05 threshold:

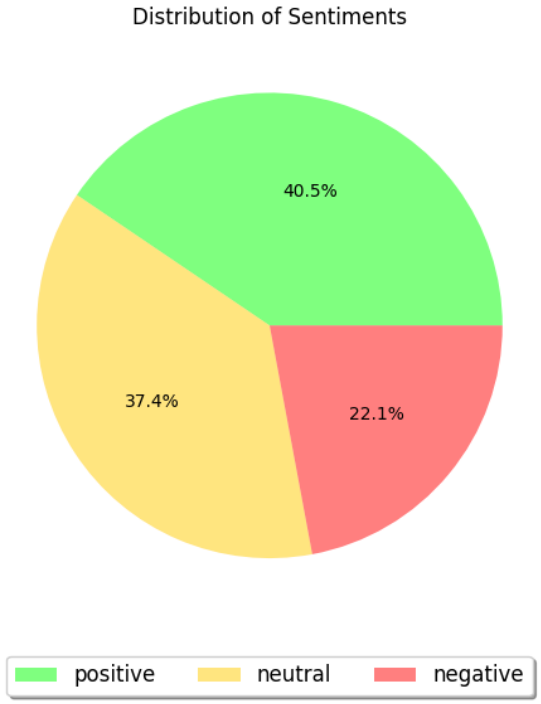


Figure 107: Distribution of sentiments, Positive comments are the most frequent ones indicating public liked this movie.

2.5 Keyword Extraction

Keyword Extraction is a complicated task that can be done using different methods, I choose nltk.Rake function which can be seen in the code below:

```
import pandas as pd
from rake_nltk import Rake
r = Rake()

def extract_keywords(text):
    r.extract_keywords_from_text(text)
    return r.get_ranked_phrases()
```

Figure 108: Keyword Extraction code.

The provided code is using the Rapid Automatic Keyword Extraction (RAKE) algorithm for keyword extraction. RAKE is an unsupervised method for extracting keywords from text. It works by splitting the text into words based on delimiters such as whitespace and punctuation. It then generates candidate keywords by dividing the words into sequences based on stop words and phrase delimiters. Each candidate keyword is assigned a score based on the degree of the word (total number of occurrences in the text) and the frequency of the word (number of times the word appears as a candidate keyword).
The Rake class from the rake-nltk library provides an implementation of this algorithm. The extract_keyword_from_text method is used to extract keywords from the input text, and get_ranked_phrases returns these keywords in descending order of their scores. Here are the results after extracting keywords:

cleaned_keyword	text_keyword
[bad garbage]	[garbage]
[lukeskywalker fought inter dimensional incarn...	[force yet someone cracking, luke skywalker fo...
[remembering naive thinking fun everyone going...	[thinking “, gonna love, everyone ’, ”, talk, ...
[trash]	[trash .😢]
[honest trailer pretty fantastic subtle hint t...	[incredibly well edited, subtle hints, pretty ...
...	...
[han shot first]	[han shot first]
[eyes]	[yyewsssssssssssssss]
[aboard good hype train]	[hype train, aboard]
[miss carriefisher]	[miss carrie fisher :&# 39 ;(]
[love]	[love]

Figure 109: Keyword Extraction Output.

2.6 conclusion

I have made plenty of contributions in this assignment:

- Crawling youtube comments
- A sophisticated list of slangs that I have archived by a lot of effort
- handling MANY challenges in text processing
- doing **2** diffrent tasks on text preocessing to have the best function possible
- NER,Bigrams,Stopwords,lemmatization,leetspeak,slangs,SpellChecking and ...

Just to see the power of my cleaning function look at this example
"I L000000VEEEE StAr Warsss its soooo goooood and I h8 when your lyingggggg & I don't careeee wrte it so you dont forgettttt ,its fire rly its goat :) 435847584 !!!!"
with my function it will be converted to :
"love starwars good hate lying not care write not forget really good really greatest time"
which has all the keywords ad handling the negative emotions in "dont" and converting words like "h8" and "goat" to their counterparts.
Another example :
"lmao your sooo funny thx ___lovely_ she paints beatyful "
my function will convert it to:
"laughing funny thanks lovely paint beautiful"
"beatyful" is spellchecked and space and underscores are deleted and "lmao" is converted to "laughing" and paints is lemmatized to paint.