GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

COMPARISATION OF RESULTS OF SENTIMENTAL ANALYSIS USING DIFFERENT MACHINE LEARNING ALGORITHMS

A Report for the Evaluation 3 of Project 2

Submitted by AKASH

YADAV

(1613101083)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of

Dr. RAVINDER AHUJA ,Assistant Professor

APRIL / MAY- 2020

SCHOOL OF COMPUTING AND SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report "COMPARISON OF RESULTS OF SENTIMENTAL ANALYSIS USING DIFFERENT MACHINE LEARNING ALGORTIHMS" is the bonafide work of "AKASH YADAV (1613101083)" who carried out the project work under my supervision.

SIGNATURE OF HEAD
School of Computing Science &
Engineering

SIGNATURE OF SUPERVISOR
School of Computing Science &
Engineering

ABSTRACT:

Now a days we are seeing in surge of social media used as a platform for marketing and influencing very targets .So understanding the specific behavior of people or individual using his/her tweets or comments is next step of sentiment analysis .We see millions of data shared on social media daily we works on both sides on one side we see various availability of data or opinions and on the other side we challenge to group them in one centroid or domain.in this work I worked on the dataset of sentiment140 from Stanford university by classifying according to polarity of the opinions using extraction features Performance of various machine learning algorithms like Ridge Classifier,LogisticRegression,Perceptron,PassiveAggressiveClassifier,SGDClassifier,LinearSVC, KNeighborsClassifier,NearestCentroid,MultinomialNB,BernoulliNB,AdaBoostClassifier.hence the goal of this work to perform the comparison between performance of these classifiers. Experiment is done Sentiment140 dataset four evaluation measures are recall, precision, f1 -score and accuracy for comparison this research demonstrate which feature will increase the accuracy of sentiment analysis

TABLE OF CONTENT

Abstract VI

List of Tables VII

List of Figures VIII

List of Symbols, Abbreviations and Nomenclature IX

1.Introduction 12

2.Machine learning Background 14

3. Literature Review 15

3.Methodology 15

3.1.Labeled Data 15

3.2.PreProcessing 18

3.2.1.tokenisation 18

3.2.2 Text Cleaning 18

3.2.3.Text Visualizations 20

3.2.4Encoding 23

3.2.5.Data visualization 25

3.2.6 .Splitting of Dataset 36

3.3.feature Extraction 37

3.3.1 bag of words(Count Vectorizer) 37

3.3.2 Tf-idf 38

3.4.Training Classifiers 39

3.4.1 Logistic Regression 39

3.4.2 Ridge Classifier 39

3.4.3 Perceptron 39

3.4.4 Passive Aggressive Classifier 40

3.4.5 Linear Support vector classification 40

3.4.6 SGD Classifier 40

3.4.7 K Neighbours Classifier 40

3.4.8 Nearest Centroid 41

3.4.9 Bernoulli NB 41

3.4.10 Multinomial NB 41

3.4.11 Ada Boost Classifier 41

3.5. Evaluation Metrics 42

4.Experiment 43

5.Result 63

6. Conclusion 70

7.Future Scopes 70

8. References 70

List of Tables:

Table 1 : Sample of sentiment140 dataset 13

Table 2: data Share in sentiment 140 16

Table 3: Example of tweets including URLs, mentions and hashtags 18

Table 4: Content for text cleaning(regular Expression) 19

Table 5: Dataset after cleaning of data 20

Table 6: Inspection of tweet words 23

Table 7: frequency of word in both positive and negative tweets 25

Table 8: Calculated posrate of tweets 31

Table 9: data with posrate and posfreq 32

Table 10: :table with harmonic mean of posword 33

Table 11: posword with cumulative distribution function of historic mean 33

Table 12: combined result of posrate ,posfreq,hmean,hmean_cdf of both positive and negative words 34

Table 13: : Result of performance of Classifier onTf-idf bigram 64

Table 14: Result of performance of Classifier bag of words trigram 65

Table 15: Result of performance of Classifier bag of words bigram 66

List of Figures:

Figure 1:preclean length of tweets graph 16

Figure 2:Boxlength graph of preclean length 17

Figure 3: screen shot of tweet_cleaning Function 19

Figure 4:wordcloud of combined tweets 21

Figure 5:wordcloud of negative tweets 22

Figure 6:wordcloud of positive tweets 22

Figure 7:Zipf’s law plot of tweet tokens 26

Figure 8: log-log graph of rank of taken vs frequency of tokens 27

Figure 9: bar chart of top 50 word in negative tweet 28

Figure 10: bar chart of top 50 word of positive tweet 29

Figure 11: positive vs negative word frequency 30

Figure 12:Plot of neg_hmean vs pos_hmean plot 35

Figure 13: plot of neg_hmeancdf vs pos_neg_hmeancdf 36

Figure 14: formula of precision Parameter 42

Figure 15: formula of recall 43

Figure 16: formula of f1-score 43

Figure 17: function of nfeatures_accuracy_checker 45

Figure 18: Unigram accuracy with or without strop words 45

Figure 19: N-gram(1-3) accuracy of count vectorizer 46

Figure 20 : N-gram (1-3) accuracy of count vectorizer and Tf-idf 59

Figure 21 : bar plot of accuracy vs classifier of bag of words bigram 67

Figure 22 :bar plot of accuracy of classifiers of bag of words trigram 67

Figure 23 :bar plot of accuracy of classifiers of tf-idf bigram 67

Figure 24: line plot of accuracy vs classifier of bag of words bigram 68

Figure 25 : line plot of accuracy of classifiers of bag of words trigram 68

Figure 26: line plot of accuracy of classifiers of tf-idf bigram 69

Figure 27: Accuracy of different Feature Extraction 69

List of Symbols, Abbreviations and Nomenclature:

List of Abbreviations:

1. 1.Nltk Natural Language Toolkit
2. 2.Bgw bag of words
3. Sklearn Scikit Learn
4. CSV comma separated value
5. SGD Stochastic gradient descent
6. KNN k-nearest neighbors
7. MNB multinomial naive bayes
8. SVM Support Vector machine
9. POS part of speech
10. PDT Pacific Daylight Time
11. UTC Universal Time Coordinated
12. www world wide web
13. Https Hypertext Transfer Protocol Secure
14. URL Uniform Resource Locator
15. f function
16. posrate positive word rate
17. posfreq positive word freqeuency
18. pos_hmean harmonic mean of positive words
19. pos_hmean_cdf cumulative distribution function of harmonic mean of positive words
20. cdf cumulative distribution function
21. hmean harmonic mean
22. negrate rate of negative words
23. frac fraction
24. negfreq frequency of neagative word

25. neg_hmean harmonic mean of negative word
26. negrate_cdf cumulative distribution function rate of negative words
27. negfreq_cdf cumulative distribution function of frequency of negative words
28. neg_hmean_cdf cumulative distribution function of frequency harmonic mean of negative words

29. clf Classifiers
30. avg average
31. len length
32. tf term frequency
33. idf inverse document frequecny
34. $\log_e$ logarithm of base e
35. pd pandas package
36. df data frame
37. np numpy
38. max maximum
39. min minimum
40. uni One
41. tri three
42. bi two
43. lr logistic regression
44. Penn University of Pennsylvania
45. UTF Unicode transformation format

List of Symbols:

1..2f float data of two digit after point

2.@ address sign

3 % modulo and percentage

4 & ampersand

5 - hyphen

6 _ underscore

7 ! Excalamation mark

8 # hashtag

9 ≈ approximately equal to

10 Σ summation

11 ^ power

List of Nomenclature:

1.N_ranges name given to define the ranges used in feature vectors

2.Unigram An n-gram an ordered n-tuple of characters when value of n is 1 it is called unigram

3.Bigram An n-gram an ordered n-tuple of characters when value of n is 2 it is called bigram

4.Trigram An n-gram an ordered n-tuple of characters when value of n is 3 it is called trigram

5.TreebankWordTokenizer is the name fiven to a function in tokenizer which is used to convert regular expressions to tokenize text as in Penn Treebank.

6.TreeBank a treebank is a parsed text corpus that annotates syntactic or semantic sentence structure.

7.Seaborn **Seaborn** is a name given a Python data visualization library based on matplotlib

8.UTF-8 UTF-8 is a variable width character encoding capable of encoding all 1,112,064 valid character code points in Unicode using one to four one-byte (**8**-bit) code units

9.latin -1 is an 8-bit character set endorsed by the International Organization for Standardization (ISO) and represents the alphabets of Western European languages.

10/Wordcloud Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud

Abstract:

Now a days we are seeing the surge of using social media as a platform for marketing and for any target .So understanding the specific behaviour of people or individual using his/her tweets or comments is next step of sentiment analysis .We see millions of data shared on social media daily we works on both sides on one side we see various availability of data or opinions and on the other side we challenge to group them in one centroid or domain.in this work I worked on the dataset of sentiment140 from Stanford university by classifying according to polarity of the opinions using extraction features Performance of various machine learning algorithms like Ridge Classifier , LogisticRegression,Perceptron,PassiveAggressiveClassifier,SGDClassifier,LinearSVC,KNeighborsClassifier,NearestCentroid,MultinomialNB,BernoulliNB,AdaBoostClassifier.hence the goal of this work to perform the comparison between performance of these classifiers. Experiment is done Sentiment140 dataset four evaluation measures are recall, precision, f1 -score and accuracy for comparison this research demonstrate which feature will increase the accuracy of sentiment analysis

Introduction:

Now a days we see usage of social media is increasing exponentially and by this various sector are targeting social media platform as their launchpad for example – usage of social media in influence the elections self-promotions etc. Social media have become gold mine to analyze the brand performance .opinion found the social media are casual, honest and informative which can be collected through various surveys .so there is need to analyze the opinion as it draws responses to various responses available on social medias. Twitter is one place where people view their opinions very strongly om different issues ,Daily there is approx. 500 million tweets by which this huge amount of data cannot be analyzed manually. Likewise, the diversity of tweets presumably cannot be captured by fixed set of rules designed by hand. It is worth noting that the task of understanding the sentiment in a tweet is more complex that of any well formatted document. Tweets do not follow any formal language structure, nor they contain words from formal language (i.e. out of vocabulary words). Often, punctuations and symbols are used to express emotions (smileys, emoticons etc.).For examining user thoughts .sentimental analysis has become a major source for purpose of solving hidden pattern in the large number of tweets with help of machine learning algorithms .in this work we have proposed classification system with ten different algorithms to sort out sentiment as negative and positive and finding out the best possible algorithm for sentimental analysis system with the help of natural language processing and machine learning and with help of python language as support system. We have taken various feature extraction and machine learning algorithm as two different entities. Our main contribution is to find out the best classification algorithm to be applied to get maximum potential of sentimental analysis by comparing four major factors of performance of each classification algorithm which as described as F1-score, precision, accuracy and recall

As for the collection of data from twitter with have taken help from the sentiment140 dataset provided by the Standard University. The table below describe the sample of the information provided by the sentiment140

First of all the data is CSV format is described as follows :

0 - the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)

1 - the id of the tweet (2087)

2 - the date of the tweet (Sat May 16 23:58:44 UTC 2009)

3 - the query (lyx). If there is no query, then this value is NO_QUERY.

4 - the user that tweeted (robotickilldozr)

5 - the text of the tweet (Lyx is cool)

sentiment	Id	Date	Query	User	text
0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
4	1467822272	Mon Apr 06 22:22:45 PDT 2009	NO_QUERY	ersle	I LOVE @Health4UandPets u guys r the best!!

Table 1: Sample of sentiment140 dataset

This work will be structed on these data first we discuss the process of analyzing the dataset and through data visualizations through wordcloud and various other graphs for data visualization .Data preprocessing is one the first step taken in sentimental analysis .then we discuss the procedure of building machine learning model and explain basics of machine learning techniques then we summarize the finding in the literature review conducted to understand the research field and identify the gap in knowledge .then we discuss our procedure of building the method for applying different algorithms and compare the result and compute tables for different feature extraction with ten different machine learning algorithm with four parameters-f1-score,recall,precision and accuracy

2.Machine Learning Background:

Before understanding research conducted for the work we need the distinguish the procedures into three equals parts which can be described as follows:

1.first the dataset of the label is compiled to according to text length and extracting only the sentiment and text from the dataset and then text cleaning process is implemented on the dataset during text preprocessing of natural language processing

2.Then we feature extractor generator used for finding value of vector should characterize the sentiment .once feature vectors from dataset and then popular classification algorithms like Ridge Classifier, Logistic Regression, Perceptron , Passive Aggressive Classifier , SGD Classifier, Linear Classifier ,KNN Nearest Neighbor Classifier, Nearest Centroid , Multinomial Navies Bayes, Bernoulli Navies Bayes ,Ada Boost Classifier

3.After applying the classification algorithms we conduct the experiment of comparison of the Four parameters which are known as F1-score ,Accuracy ,Precision and Recall and publish the result in Tabular form.

3.Literature Review:

The sentiment140 data set used in our experiments was created using an automated sentiment labeling method [6]. Go et al. [6] created an automated labeling method which took advantage of emoticons found in tweets. Emoticons are a combination of symbols that express an emotion, usually forming a facial representation, such as ":)" which depicts a positive emotion. Tweets were collected and labeled based on the emotion assigned to each emoticon, resulting in a data set of 1.6 million positive and negative tweets. They performed sentiment prediction using three machine learning algorithms: MNB, Support Vector Machines (SVM), and Maximum Entropy. They used unigrams, bigrams, unigrams and bigrams, and unigrams with Part-Of-Speech (POS) tags as features. Their results show the use of unigrams and unigrams with bigrams have the highest performance. SVM had the highest performance when using unigrams, while Maximum

Entropy had the highest performance when using unigrams with bigrams. The difference in performance between the classifiers when using unigrams and unigrams with bigrams is smaller than 2%, and they conducted no tests to determine if this difference was significant. The authors mentioned bigram features alone did not perform well due to the length of tweets. As they are shorter posts, 140 characters or less, a bigram feature space becomes very sparse.

Our study is unique in that we provide a comprehensive comparison of Performance of various classifiers using four performance parameters or metrices .We compare the performance of the best model built using our sentiment140 data set against models built using tweets with help of different classifiers. Models are built with Multinomial Naive Bayes and evaluated across 1.6 million distinct tweets. Our classifiers are trained on large data sets, consisting of 100,000 instances, including Amazon product review data, which consists of many diverse product domains, and sentiment140 data. We evaluate our models on a sentiment140 data set.

3.Methodology:

3.1.Labeled Data:

We have used dataset Sentiment140 which can be described in the specific format defined by the publisher of dataset which in this case is Stanford university. Which are described in following ways:

First of all the data is CSV format is described as follows :

0 - the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)

1 - the id of the tweet (2087)

2 - the date of the tweet (Sat May 16 23:58:44 UTC 2009)

3 - the query (lyx). If there is no query, then this value is NO_QUERY.

4 - the user that tweeted (robotickilldozr)

5 - the text of the tweet (Lyx is cool)

sentiment	Id	Date	Query	User	text
0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
4	1467822272	Mon Apr 06 22:22:45 PDT 2009	NO_QUERY	ersle	I LOVE @Health4UandPets u guys r the best!!

Table: Sample of sentiment140 dataset

The next table shows the share of positive and negative class distribution in dataset

Sentiment	Data in numbers
Positive	800000
Negative	800000

Table 2:Data share in dataset

Datatype is integer 64 in dataset from the table we have gather information that there are 800000 positive tweets and 800000 negative tweets. There are no neutral tweets in the dataset. The text in tweets are variable lengths containing various mentions ,usernames ,escapes ,URLs links, hashtag s and negations so there is diversity in text of dataset below figure will show how variable length of data is present in dataset

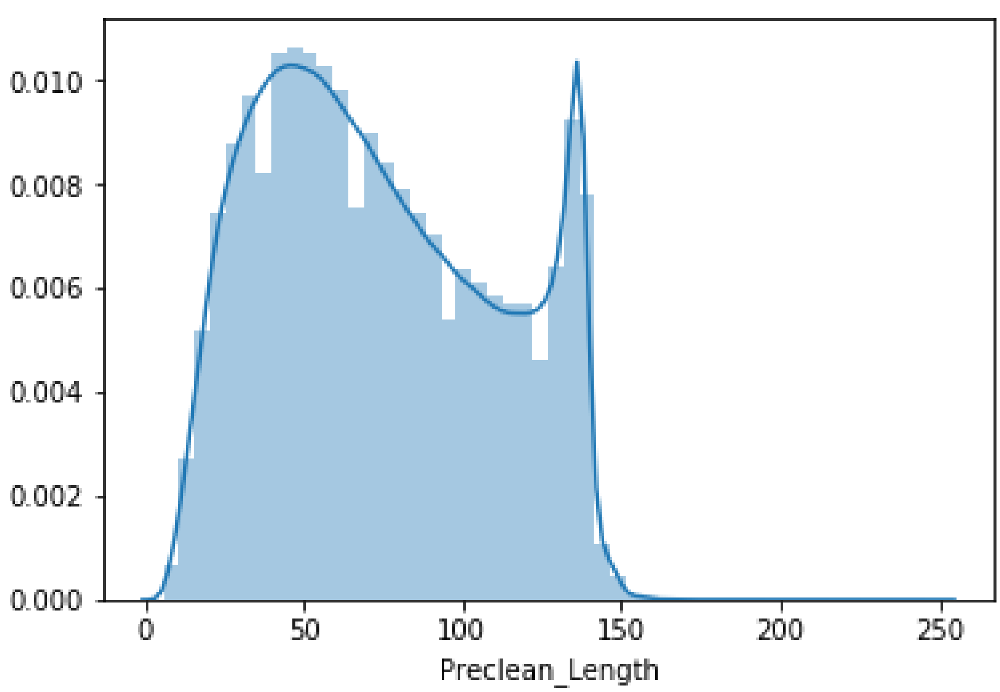

Figure 1:preclean length of text of tweets

Above figure is drawn with the help of **seaborn** by which have developed a scatter plot to show the number of texts with their precleaned length with help of distplot which is used to visualize histogram show the number of tweets with their respective length. Another figure of preclean length is shown below with the help of **Boxplot of seaborn**

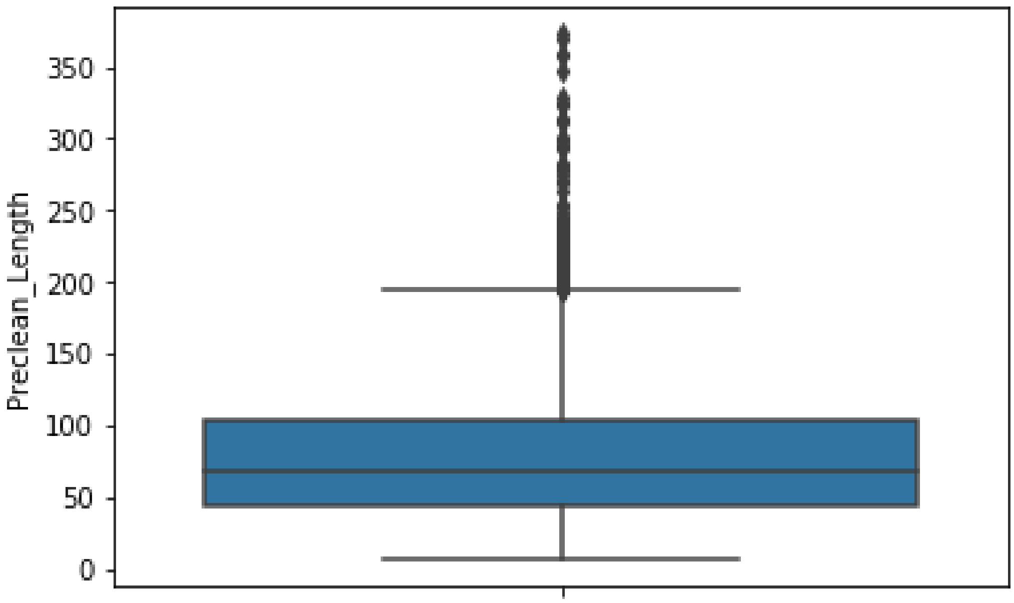

Figure: 2 Boxplot graph of preclean length

UTF-8 can limit with 128 characters by which by seeing above figures we see there are more than 128 characters so we have to convert into **latin-1** encoding. After encoding in latin-1 we further begin our process of finding the irregularity in text which will be elaborates further in this segment:

Firstly, we see a text which contains lot of space which is unnecessary and various special characters ,single quotes , double quotes etc.

"Awwh babs... you look so sad underneith that shop entrance of "Yesterday's Musik " O-: I like the look of the new transformer movie "

Other type of text includes links, URL mentions etc.:

Text
"@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D"

"@machineplay I'm so sorry you're having to go through this. Again. #therapyfail"

Table 3:Example of tweets including urls,mentions and hastags

3.2 Preprocessing:

Before the process of applying machine learning algorithms on our work we need to clean the data .cleaning the data is one of the initial step of data pre-processing and these step helps in converting the text into processable elements with information added that can be utilized by feature extractor:

3.2.1.**Tokenisation:** Tokenization is the process of converting text as a string into processable elements called tokens. In the context of a tweet, these elements can be words, emoticons, URL links, hashtags or punctuations. These elements are often separated by spaces. However, punctuation ending the sentence like exclamation marks or full stop are often not separated by a space. On the other hand, hashtags with "#" preceding the tag needs to be retained since a word as a hashtag may have different sentiment value than a word used regularly in the text.

"@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D"

with the help of **nltk** package of machine learning have a functionality of tokenize which contains **TreebankWordTokenizer** which is used to convert regular expressions to tokenize texts in **Penn treebank .**it assumes that the text already been segmented into sentences

```
from nltk.tokenize import TreebankWordTokenizer

token = TreebankWordTokenizer()
```

3.2.2.text Cleaning:

We have already discussed type of data we have collected from our dataset we finally take a major step in preprocessing which cleaning the texts to decreasing the length of the text to help them to convert into processable element. The above-mentioned exception or unwanted text which occur in our dataset can be categorized into four elements which is as follows:

Unwanted words	Regular Expression
mentions	r'@[A-Za-z0-9]+'
URL HTTPs	'https?://[A-Za-z0-9./]+'
url_www	r'www.[^]+'

Table 4:Content for text cleaning(regular Expression)

A list is created to eliminate the negation words which occur in text and changing into its full form does not change the value of sentence for example "isn’t”: “is not", "aren’t”: “are not", "wasn’t”: “was not", "weren’t”: “were not", "haven’t”: “have not","hasn't":"has not".

A function is created to replace these words.

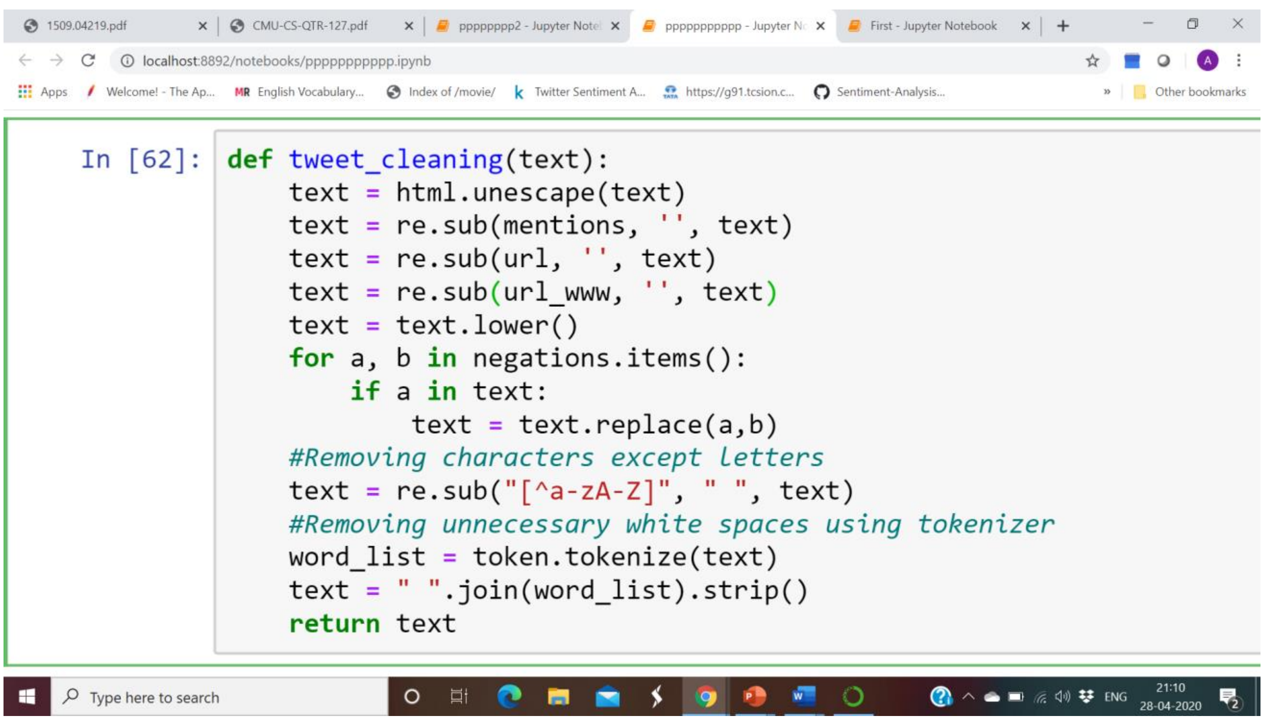

Figure 3: screen shot of tweet_cleaning Function

After applying the function on data new dataset is created which only contain processable element.

New dataset is look like below table:

Sentiment	Post clean Text
0	awww that s a bummer you shoulda got david car...
0	is upset that he can not update his facebook b...
0	i dived many times for the ball managed to sav...
0	my whole body feels itchy and like its on fire40no
0	its s not behaving at all i m mad why am i h...

Table 5:Dataset after cleaning of data

3.2.3.**Text Visualizations**:

This type analyses of data can be done with help of **wordcloud** which used to get relation between sentiment text which is achieved with help of wordcloud which shows the relation between these text with each other words below figure consists of cloud of words which have the highest frequencies in the dataset.

Figure 4:worldcloud for both the sentiment combined

Now we have seen the most occurring word in our dataset are drinking, thanks etc.

By this we can analyze how our data is defined in the dataset and how people opinions generally contain similar words which are used to define both the sentiments -positive and negative.

For further text visualization we will apply wordcloud individually on positive sentiment as well as on negative sentiment. So below figures will explain the frequencies of words in both classes of classifications

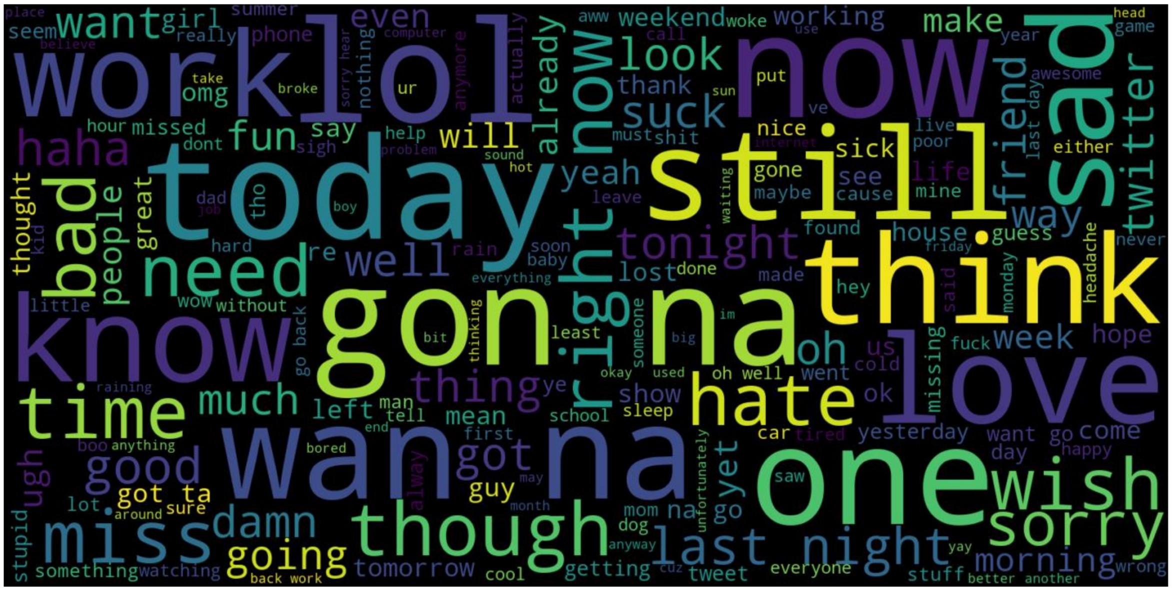

Figure 5: wordcloud of negative tweets

Some words, like, "today", "one", "still" can be termed as neutral. Words like, "sad", "bad", "hate", "suck", "wish" etc. make sense as negative words.

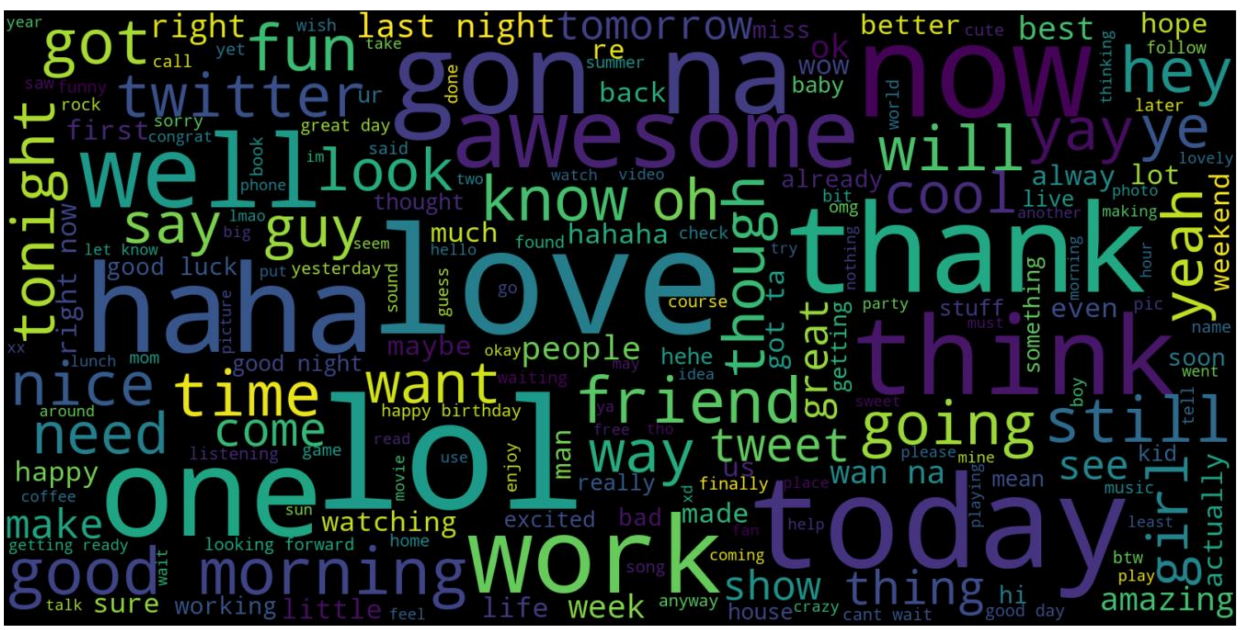

Figure 6: wordcloud of positive tweets

In this wordcloud of positive tweets, neutral words, like "today", "tonight", "still", etc are present.

Also, words like "thank", "haha", "awesome", "good", etc stand out as the positive words.

Words like "today", "lol", "tonight", "still", "work" etc are common in both the positive and negative tweets. Hence, it can be concluded that people have both positive and negative response towards work and their day.

What we have found surprising is the presence of "lol" and "love" in both the positive and the negative tweets wordclouds. So, now, I am going to inspect this.

For inspection I tried to search “love” word in negative tweets and find out the count of tweets containing the word “love” same has been done to find out the word “lol” in both positive and negative tweets and the below will surely describe the result of the inspection carried.

Word	Count of tweets in which word is available
“love” in negative tweets	21548
“lol” in positive tweets	35780
“lol” in negative tweets	22754

Table 6: inspection of word in tweets

there are 21.5k negative tweets where the word 'love' is used. But one thing I observed is that love is used with negative words like sad, loss, no, leave, etc or it is used sarcastically.

I also inspected the use of 'lol' in tweets of both, positive and negative sentiments. In positive tweets, lol is used as an expression for joy, fun and laughter. And in negative tweets, 'lol' is used with words that convey negative emotion like 'sad', 'crying', 'slap', 'no', 'bored' etc.

3.2.4.Encoding:

for data visualisation firstly we have to prepare the text for the data visualisation Here we taken **CountVectorizer** The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. With the use of countvectiozer we have built vectors of words of tweets and length of the corpus of words is 271304 and then we have encoded the document into two different matrix or vector for the positive and negative tweets .

After encoding we have found out the corpus are defined as following:

```
aa                                                                265
aaa                                                               152
aaaa                                                               74
aaaaa                                                              38
aaaaaa                                                             28
                                                                 ...
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz                                     1
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz                                  3
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz                                 1
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz              1
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz           1
Name: total, Length: 271304, dtype: int64
```

After finding the corpus of encoded data we have calculated term frequency of the word which will be shown below table:

	negative	positive	total
to	313185	252600	565785
the	257953	266161	524114
my	190805	125981	316786
it	157491	147841	305332
and	153985	149692	303677
you	103865	198274	302139
not	196637	87771	284408
is	133533	111325	244858
in	115628	101297	216925
for	99044	117389	216433
of	92837	91218	184055
on	84227	84231	168458
that	82734	83070	165804

	negative	positive	total
me	92188	72247	164435
so	88534	65627	154161
have	88400	65586	153986
but	84896	48600	133496
just	64006	62946	126952
do	68206	48506	116712
with	50156	65187	115343

Table 7 :frequency of words in both positive and negative tweets

by examine the above table we have find out that most of the term are stop words

3.2.5.Data visualisation:

Data visualisation is done using zipf's law which states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. Thus, the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.: the rank-frequency distribution is an inverse relation.

Suppose a word occurs f times and that in the list of word frequencies it has a certain rank, r. Then if Zipf's Law holds we have

$\boldsymbol{f=a/rb}$**f=a/rb**

where a and b are constants and $b\approx 1$b≈1.

Let's see how the tweet tokens and their frequencies look on a plot

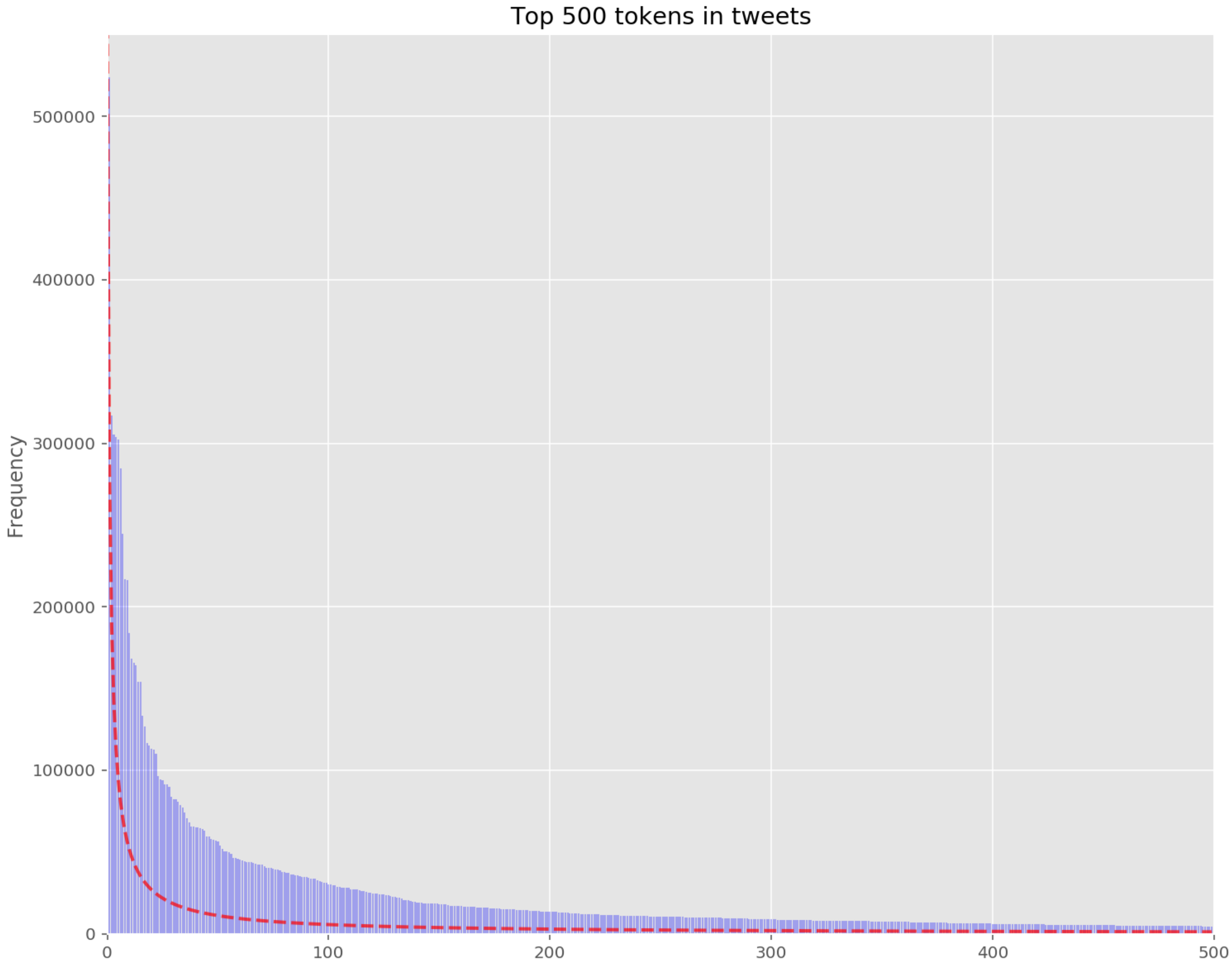

Figure 7:Zipf's law plot of tweet tokens

On the X-axis are the top 500 tokens of the corpus with the highest rank in the left and 500th rank in the right. Y-axis consists of the frequencies of the top 500 words most frequent words in the Sentiment140 corpus. The curve here is not the exact Zipfian curve, rather a near Zipfian distribution curve. Even though we can see the plot follows the trend of Zipf's Law, but it looks like it has more area above the expected Zipf curve in higher ranked words. We can also plot a log-log graph, with X-axis being log(rank), Y-axis being log(frequency). By plotting, the result will be a roughly linear line.

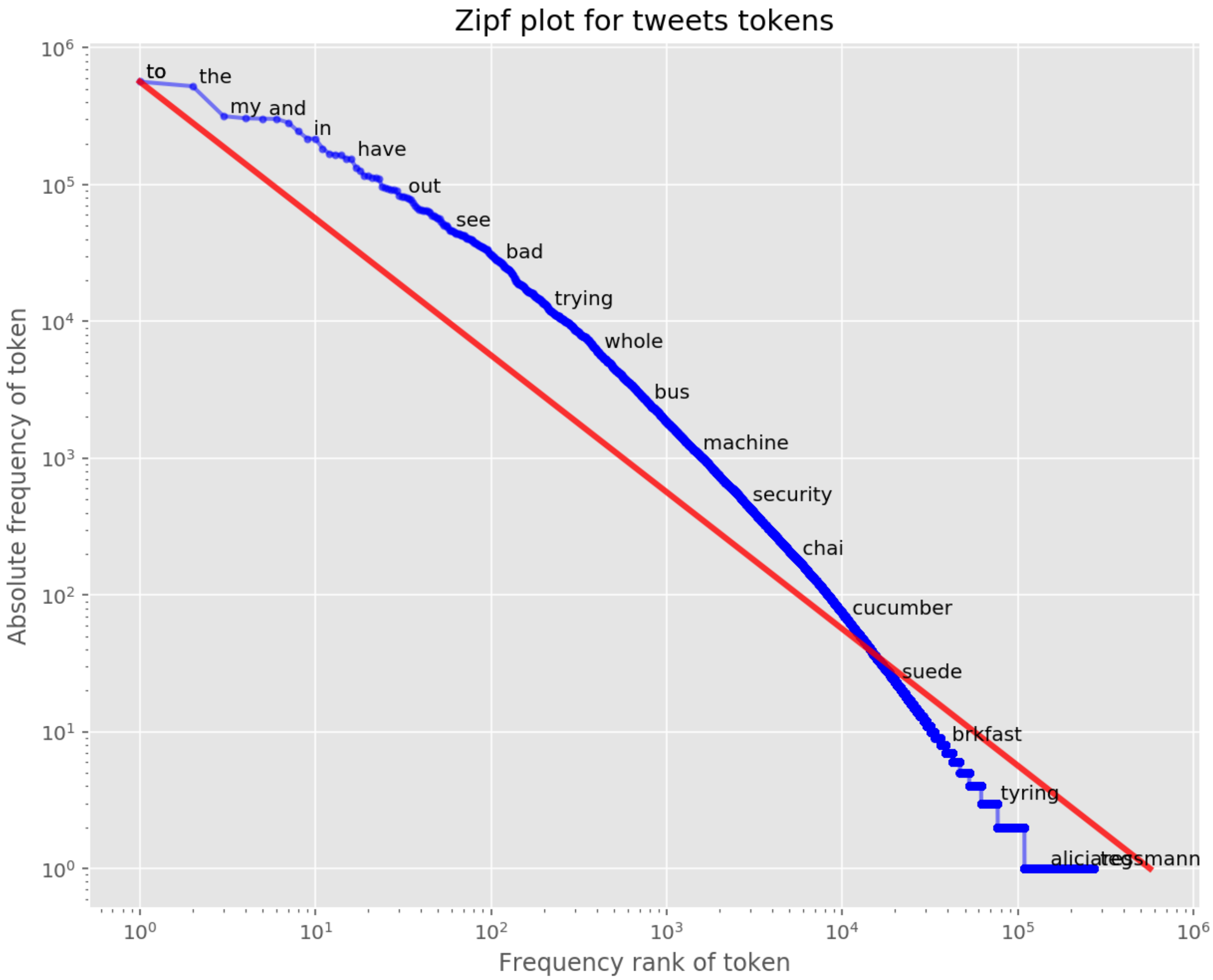

Figure 8: log-log graph of rank of taken vs frequency of tokens

Here, we see a roughly linear curve, but deviating above the expected line on higher ranked words and deviating below the expected line on lower ranked words

From the previous step of encoding using Count vectorizer we have prepared corpus of tweets from that result we will see top 50 words in negative in form of bar chart figure:

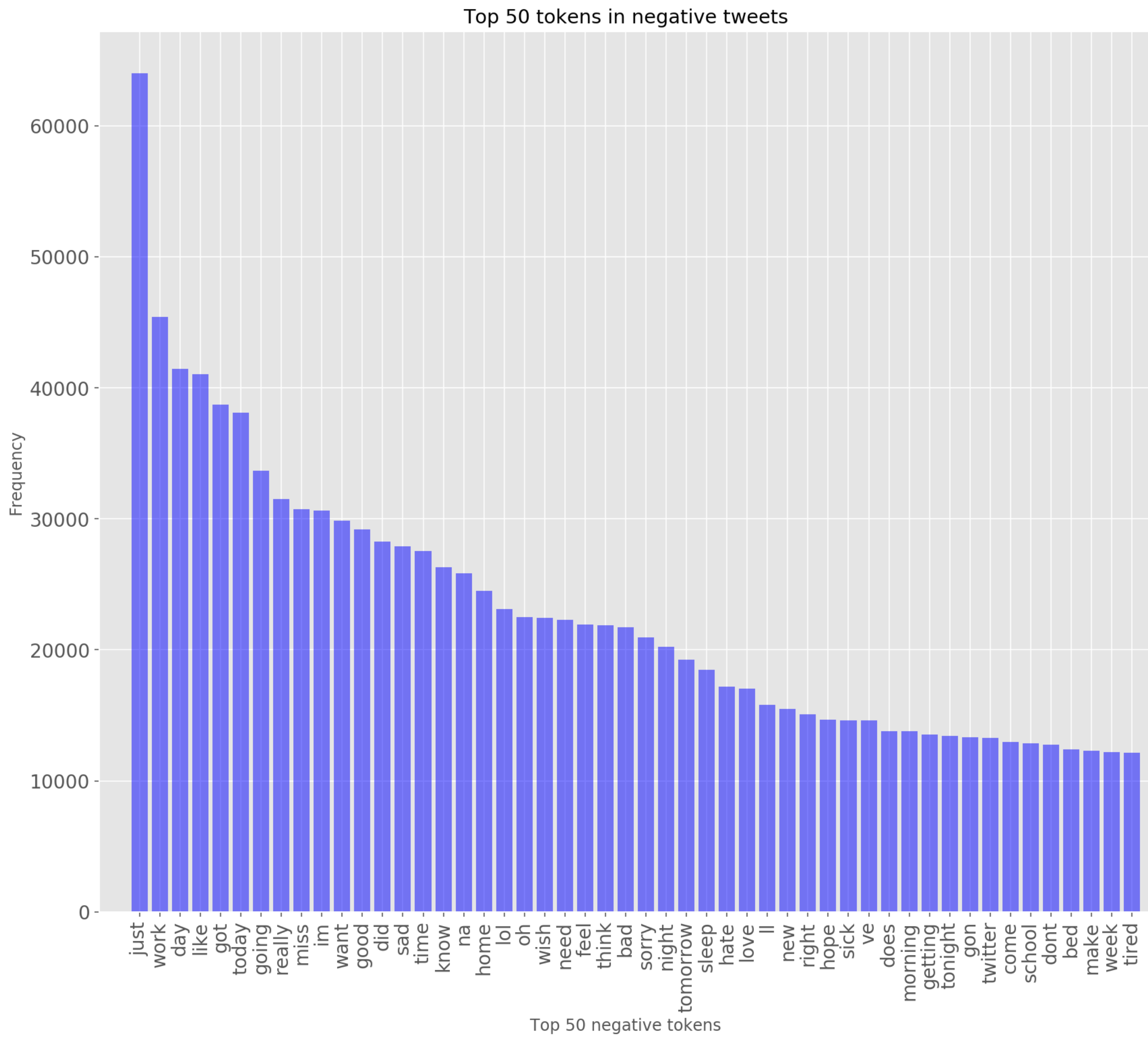

Figure 9 :bar chart of top 50 word in negative tweet

The most frequent words like "just", "work", "day", "got", "today" etc. do little to convey negative sentiment. It's difficult to comment about their importance in characterising negative tweets. On the other hand, words like, "miss", "sad", bad", "sorry", "hate" etc. convey clear negative sentiment.

Let's see the top 50 words in positive tweets on a bar chart.

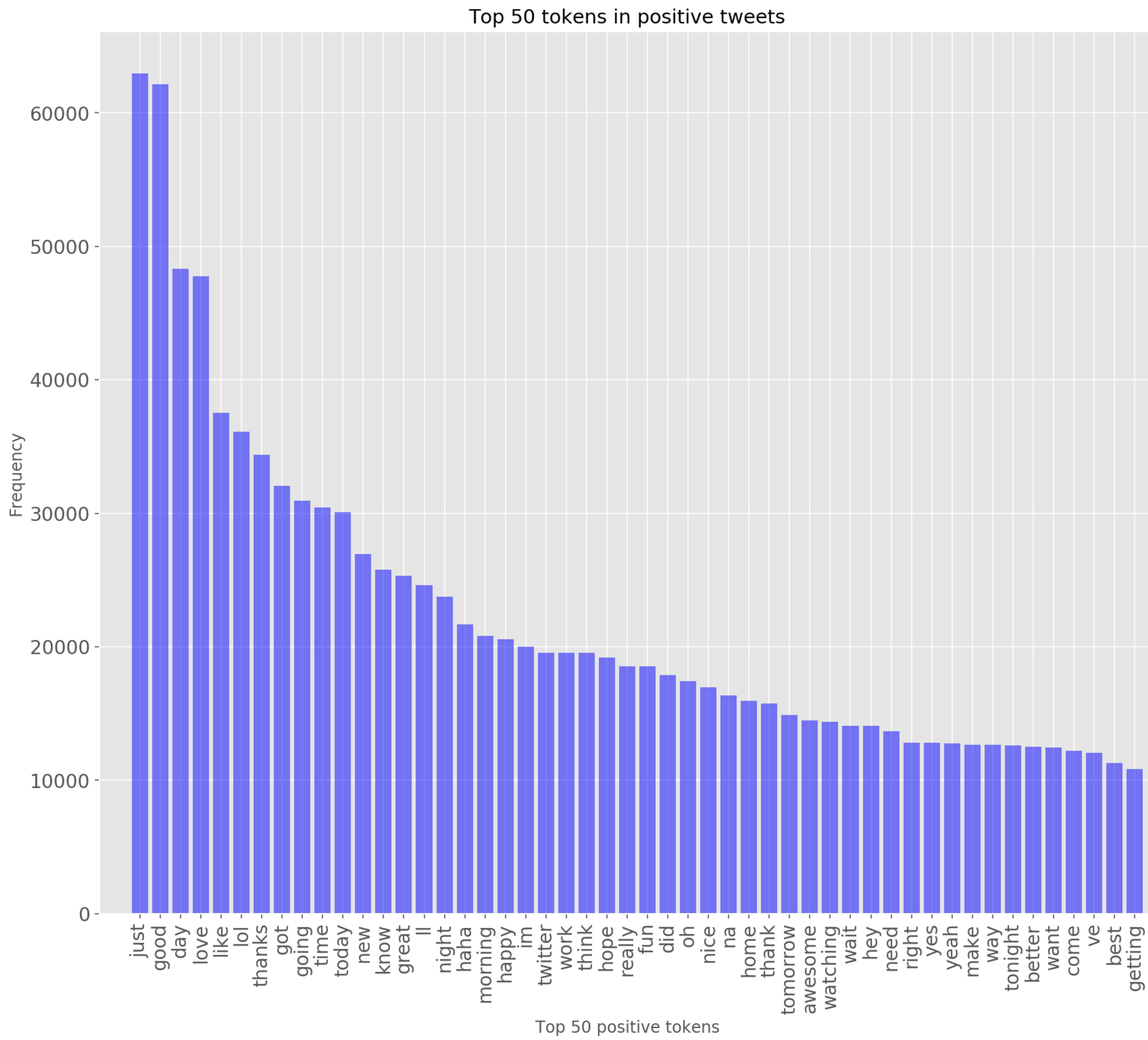

Figure 10: bar chart of top 50 word of positive tweet

The most frequent words like "just", "day", "got", "today", "time" etc. do little to convey positive sentiment. It's difficult to comment about their importance in characterising positive tweets. On the other hand, words like, "good", "love", "like", "thanks", "new" etc. convey clear positive sentiment.

Let's plot the negative frequency of a word on x-axis and the positive frequency on y-axis.

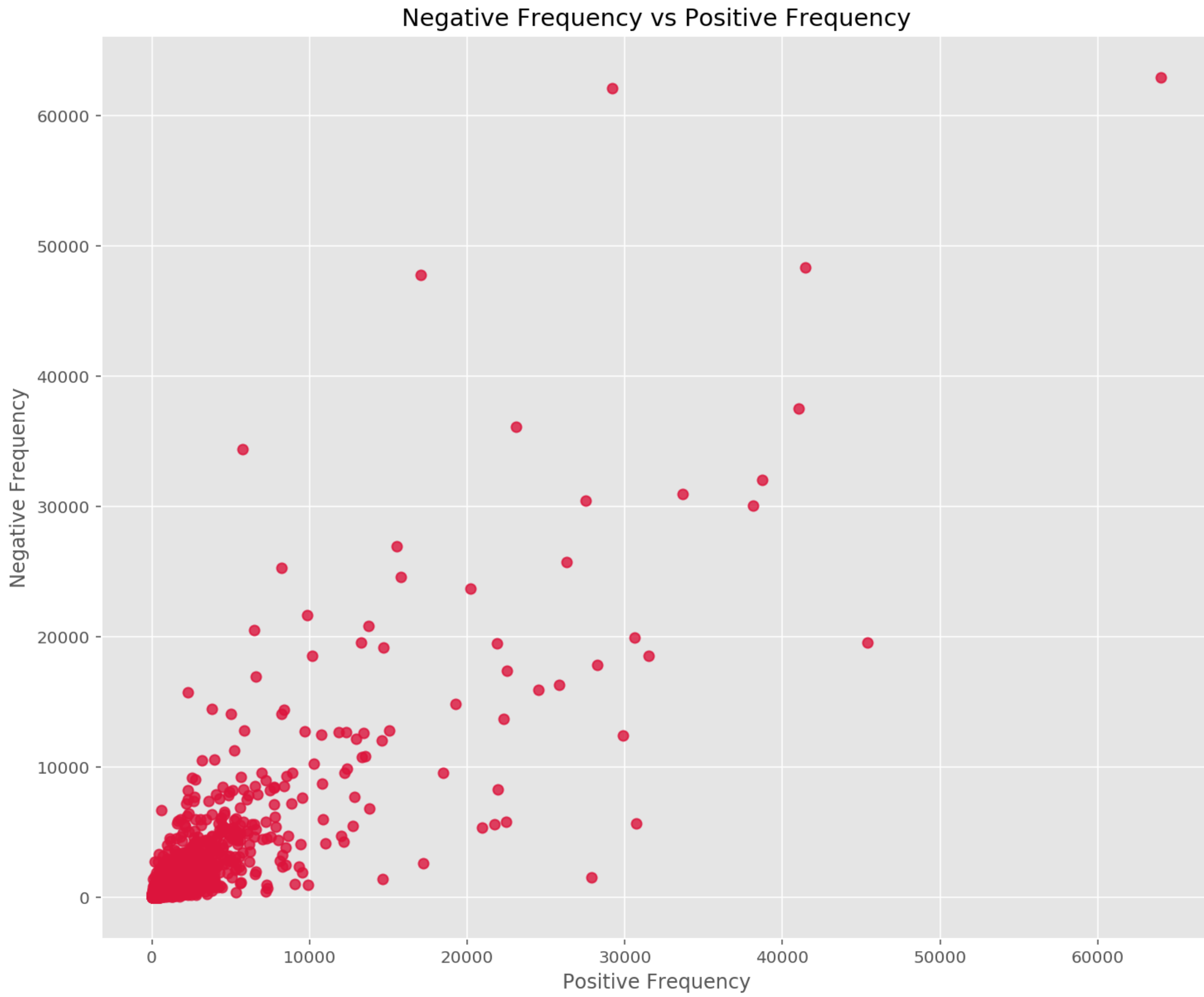

Figure 11: positive vs negative word frequency

Most of the words are below 10000 on both Y and X-axis, hence, we can find any meaningful relation between positive and negative frequency.

The next metric has been taken from Jason Kessler's talk in Pydata 2017 in Seattle, where he introduced Scatter text.

If a word appears more in one class as compared to the other, we can use it as a measure of how much important the word is to characterise the class. Let's call it posrate.

$$posrate=(positivefrequency)/(positivefrequency+negativefrequency)$$

Below the data show the scatter text data which calculated from above formula:

Word	Negative	Positive	Total	posrate
dividends	0	83	83	1.00000
emailunlimited	0	100	100	1.00000
mileymonday	0	161	161	1.00000
shareholder	1	80	81	0.987654
fuzzball	2	99	101	0.980198
recommends	3	109	112	0.973214
delongeday	6	162	168	0.964286
atcha	3	80	83	0.963855

Table 8 : Calculated posrate of tweets

Words with highest posrate have 0 frequency in negative class. But the frequency of these words is quite low to use them as a measure to characterise positive tweets that's why we use another metrices which is the frequency a word occurs in the class. This is defined as

$$posfreq=positivefrequency/\Sigma(positivefrequency)$$

The below table will show both the posfreq and posrate:

posword	Negative	Positive	Total	Posrate	posfreq
just	64006	62946	126952	0.495825	0.014264
good	29213	62122	91335	0.680155	0.014077
day	41456	48319	89775	0.538223	0.010949
love	17061	47780	64841	0.736879	0.010827
like	41052	37527	78579	0.477570	0.008504
lol	23123	36120	59243	0.609692	0.008185
thanks	5768	34378	40146	0.856324	0.007790
got	38715	32030	70745	0.452753	0.007258
going	33690	30939	64629	0.478717	0.007011

time	27532	30438	57970	0.525065	0.006897

Table 9: data with posrate and posfreq

Since posfreq is just the frequency scaled over the total sum of the frequency, the rank of posfreqpct is exactly same as just the positive frequency.

The maximum value from posfreq is 0.01426404088907219 and maximum and minimum value Are 0 and 1 so we need to come up with a metric which combines posrate and posfreq. The range of posrate is 0 to 1. The range of posfreq is 0 to ~0.015. If we take the average of posrate and posfreq, posrate will be too dominant and will not reflect the two metrics properly.

Hence, instead of arithmetic mean, we use harmonic mean. It increases the effect of the small values and reduces the effect of the larger ones. The harmonic mean H of positive real numbers x1, x2,...... xn is defined as

H = \frac{n}{Σ_{i=1}^{n}\frac{1}{x_{i}}}

We have added harmonic mean to our previous table which shown us the posfreq and posrate .the table id shown below:

posword	Negative	Positive	Total	Posrate	posfreq	Pos_hmean
just	64006	62946	126952	0.495825	0.014264	0.027730
good	29213	62122	91335	0.680155	0.014077	0.027584
day	41456	48319	89775	0.538223	0.010949	0.021462
love	17061	47780	64841	0.736879	0.010827	0.021341
like	41052	37527	78579	0.477570	0.008504	0.016710
lol	23123	36120	59243	0.609692	0.008185	0.016153
thanks	5768	34378	40146	0.856324	0.007790	0.015440
got	38715	32030	70745	0.452753	0.007258	0.014287
going	33690	30939	64629	0.478717	0.007011	0.013820

time	27532	30438	57970	0.525065	0.006897	0.013616

Table 10:table with harmonic mean of posword

The harmonic mean rank seems just like the posfreq rank. Here, the impact of the posfreq significantly increased and dominated the mean value. Hence, we still can't come to a meaningful conclusion.

Now, we will try the Cumulative Distribution Function. The cumulative distribution function (CDF) of a real-valued random variable X, evaluated at x, is the probability that X will take a value less than or equal to x. Now, we do calculate harmonic mean of these 2 CDF values.

	negative	positive	total	posrate	posfreq	pos_hmean	posrate_cdf	posfreq_cdf	pos_hmean_cdf
welcome	620	6702	7322	0.915324	0.001519	0.003032	0.995710	0.999312	0.997508
thank	2282	15737	18019	0.873356	0.003566	0.007103	0.990923	1.000000	0.995441
thanks	5768	34378	40146	0.856324	0.007790	0.015440	0.987920	1.000000	0.993923
awesome	3829	14475	18304	0.790811	0.003280	0.006533	0.967084	1.000000	0.983266
glad	2273	8255	10528	0.784100	0.001871	0.003732	0.963836	0.999967	0.981569
follow	2553	9154	11707	0.781925	0.002074	0.004138	0.962729	0.999996	0.981009
enjoy	1642	5876	7518	0.781591	0.001332	0.002659	0.962556	0.997262	0.979602
sweet	1621	5653	7274	0.777151	0.001281	0.002558	0.960202	0.996135	0.977838
yay	3171	10508	13679	0.768185	0.002381	0.004748	0.955079	1.000000	0.977023
hi	2176	7219	9395	0.768387	0.001636	0.003265	0.955200	0.999734	0.976960

Table 11:psoword with cumulative distribution function of historic mean

Now we have seen calculation of posrate,posfreq ,hmean and hmaeanofcdfs in of positive words we have to calculate exact things for the negative words and after both positive and negative words are combined in table and printed below:

	negative	**positive**	**total**	**posrate**	**posfreq**	**pos_hmean**	**posrate_cdf**	**posfreq_cdf**	**pos_hmean_cdf**	**negrate**	**negfreq**	**neg_hmean**	**negrate_cdf**	**negfreq_cdf**	**neg_hmean_cdf**
sad	27911	1510	29421	0.051324	0.000342	0.000680	0.002221	0.707535	0.004428	0.948676	0.006038	0.012000	0.997779	1.000000	0.998888
hurts	7204	456	7660	0.059530	0.000103	0.000206	0.002611	0.502999	0.005196	0.940470	0.001558	0.003112	0.997389	0.999690	0.998538

	miss	headache	hate	missing	ugh	poor	sucks	sick
negative	30737	5317	17208	7282	9056	7333	9902	14620
positive	5710	421	2618	991	1000	719	982	1420
total	36447	5738	19826	8273	10056	8052	10884	16040
posrate	0.156666	0.073371	0.132049	0.119787	0.099443	0.089295	0.090224	0.088529
posfreq	0.001294	0.000095	0.000593	0.000225	0.000227	0.000163	0.000223	0.000322
pos_hmean	0.002567	0.000191	0.001181	0.000448	0.000452	0.000325	0.000444	0.000641
posrate_cdf	0.014717	0.003412	0.009809	0.007949	0.005541	0.004602	0.004681	0.004537
posfreq_cdf	0.996457	0.495863	0.867034	0.610624	0.612387	0.556433	0.608859	0.691533
pos_hmean_cdf	0.029005	0.006778	0.019398	0.015694	0.010982	0.009128	0.009291	0.009014
negrate	0.843334	0.926629	0.867951	0.880213	0.900557	0.910705	0.909776	0.911471
negfreq	0.006649	0.001150	0.003723	0.001575	0.001959	0.001586	0.002142	0.003163
neg_hmean	0.013195	0.002298	0.007414	0.003145	0.003910	0.003167	0.004274	0.006304
negrate_cdf	0.985283	0.996588	0.990191	0.992051	0.994459	0.995398	0.995319	0.995463
negfreq_cdf	1.000000	0.993145	1.000000	0.999732	0.999994	0.999757	0.999999	1.000000
neg_hmean_cdf	0.992587	0.994863	0.995071	0.995877	0.997219	0.997573	0.997653	0.997726

Table 12: combined result of posrate ,posfreq,hmean,hmean_cdf of both positive and negative words

Now we have calculated the pos_hmean and neg_hmean we have visualized by drawing a plot where pos_hmean is X-axis and neg_hmean is neg_hmean

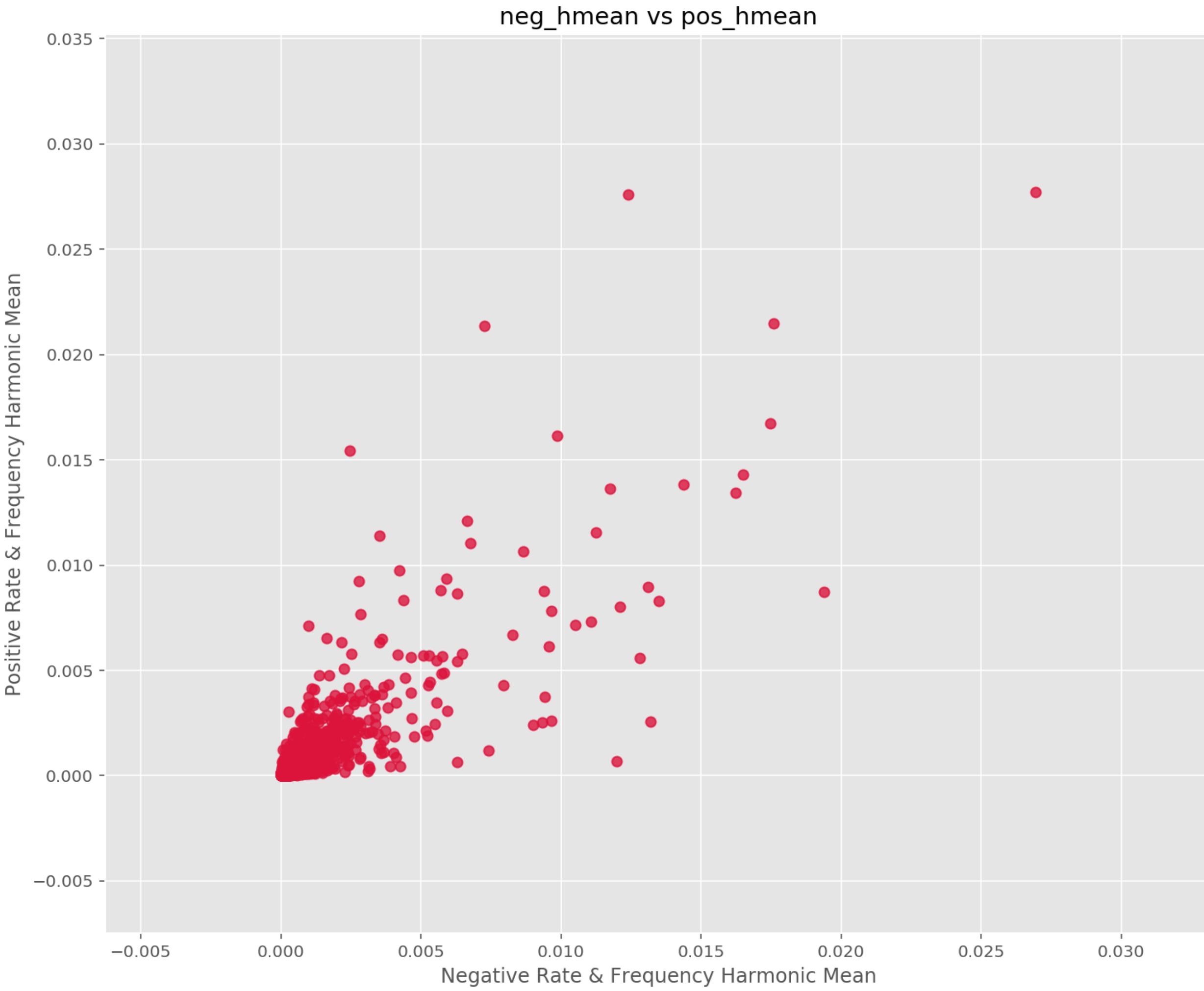

Figure 12:neg_hmean vs pos_hmean plot

After plotting neg_hmean_cdf (X-axis) vs pos_hmean_cdf (Y-axis), we find that if a data point is near the upper left, it is more positive. And if a data point is near the bottom right, it is more nega tive.

Now we have same plot using pos_hmean_cdf instead of pos_hmean and the result of graph is shown below:

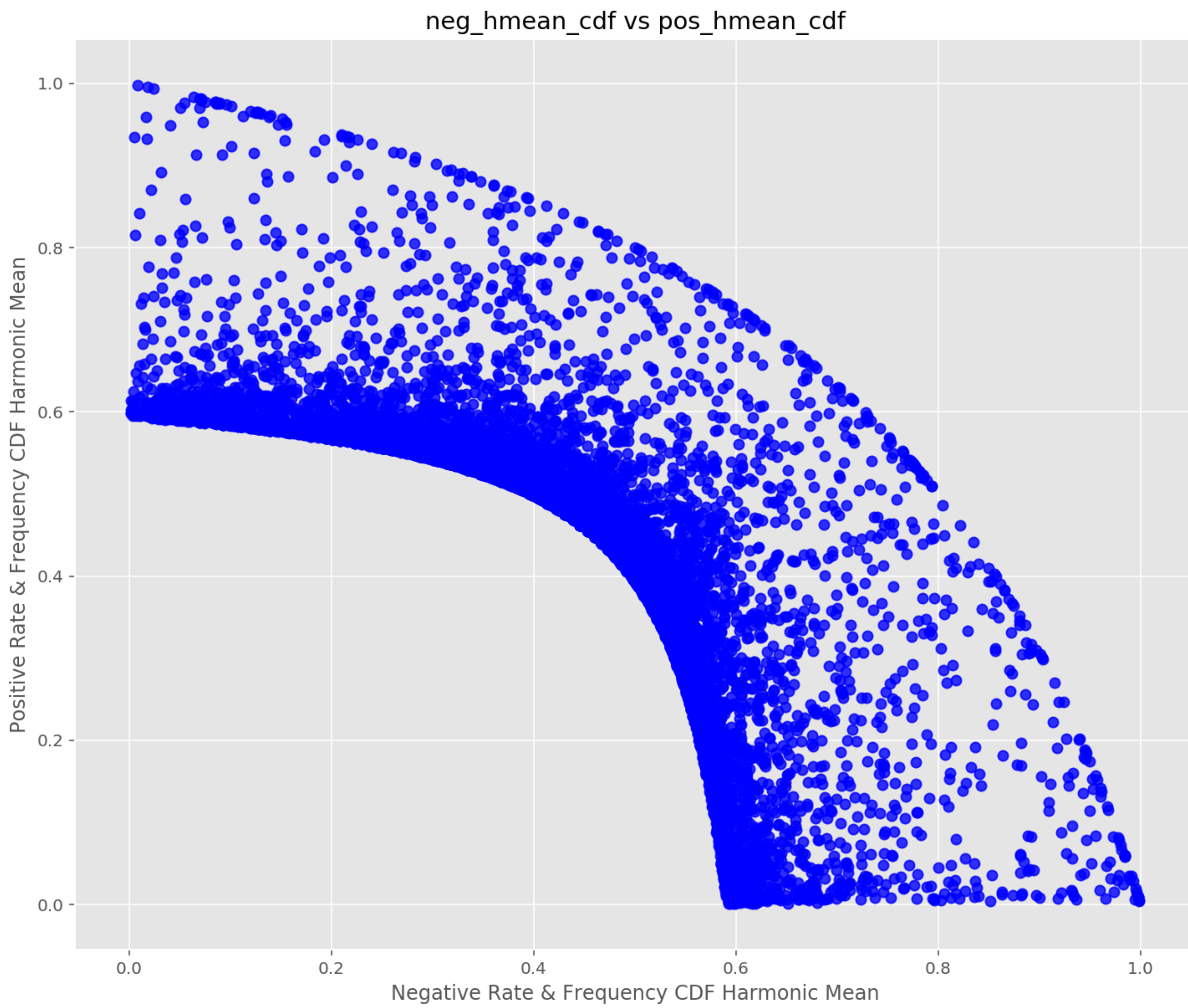

Figure 13: plot of neg_hmeancdf vs pos_neg_hmeancdf

3.2.6 Splitting of Dataset:

We will split the dataset into three sets which will be defined as follows

- Train set: The dataset used for learning
- Development Set: A validation/development dataset is a sample of data held back from training your model that is used to give an estimate of model skill while tuning model's hyperparameters.
- Test Set: The dataset used to assess the performance of a model

Our chosen ratio is 98/1/1 i.e. 98% for the training set, 1% for the development set and 1% for the testing set.

Using sklearn.model_selection import train_test_split we splited the data using the below code

```
x =df['text']#define all other columns except the target variable
y = df['sentiment'] #define the target variable
x_train, x_validation_and_test, y_train, y_validation_and_test =
train_test_split(x, y, test_size = 0.02, random_state = 42)

x_validation, x_test, y_validation, y_test = train_test_split(x_
validation_and_test, y_validation_and_test,test_size = 0.5, rando
m_state = 42)
```

After the splitting the dataset the entries are as follows:

Training set has 1564779 entries, where 49.99 are positive and 50.01 are negative

Validation set has 15967 entries, where 49.82 are positive and 50.18 are negative

Testing set has 15968 entries, where 50.33 are positive and 49.67 are negative

3.3 Feature Extraction:

Feature extraction is the process of building feature vector from a given tweet. Each entry in a feature vector is an integer that has a contribution on attributing a sentiment class to a tweet. This contribution can vary from strong, where the value of a feature entry heavily influences the true sentiment class; to negligible, where there is no relationship between feature value and sentiment class. It is often the job of classification algorithm to identify the dependency strength between features and classes, making use of strong correlated features and avoiding the use of ‘noisy features .

In our project we have taken two feature Extraction which are known as follows:

3.3.1 Bag of Word or Count Vectorizer: Bag of Words (unigrams) is a set of features where the frequency of tokens (or in our case, presence of a token) is indicated in a feature vector. From our study of past work, this feature set was unanimously chosen by researchers to be included in the feature vector. An entry in the feature vector is assigned to each unique token found in the labelled training set. If the respective token occurs in a tweet, it is assigned a binary value of 1 otherwise it is 0. Note that the grammar structure or ordering of token sequence is not preserved. Instead, only the independent presence of a token preserved .in this project we taken three types of gram to ana lyse our result which are as follows: **unigram, bigram and trigram** .In **sklearn** we use Count V ectorizer with the argument **n_ranges** which help in achieving different grams

3.3.2 Feature Extraction Using TF-IDF:

In a large text corpus, some words will be very present (e.g. “the”, “a”, “is” in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms.

Term Frequency measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length or the total number of terms in the document as a way of normalization:

$TF(t)$=Number of times term t appears in a document Total number of terms in the document

Inverse Document Frequency measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$IDF(t)$=$\log_e$Total number of documents Number of documents with term t in it

Combining these two, we get TF-IDF.

$$TF-IDF(t)=TF(t)\times IDF(t)$$

The higher the TFIDF score, the rarer the term and vice versa.

Tf-idf is computed by sklearn,featureExtracter feature which is tfidfVectorizer which have various arguments

3.4 Training Classifier:

We have taken **textBlob** as baseline for the sentimental analysis .it will provide as a point of reference for our future models.

Textblob provides a common text processing operation like sentiment analysis ,tokenisation etc.

3.4.1 Logistic Regression is first classifier which we will use for training logistic regression is used to model the probability of certain class it is very efficient and does not requires too many computational resources that why we use logistic regression classifier.

As we have discussed earlier that we are going to train our model with ten classifiers to compare the result of these classifier using specific parameters .So to compute these classifiers together we have taken use of **pipeline** Pipeline class allows sticking multiple processes into a single scikit-learn estimator. Classifier used in this progress are described below:

3.4.2 Ridge Classifier: this classifier is mainly use ridge regression for classification of multi class outputs. Ridge regression simply addresses the problems of ordinary least squares by imposing penalties on size of the coefficients.

3.4.3 Perceptron :perceptron is one of classifier in package of sklearn.linearmodel . the **perceptron** is an algorithm for supervised learning of binary classifiers. A binary classifier is a function which can decide whether or not an input, represented by a vector of numbers, belongs to some specific class. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

It does not require any learning rate and does not have any regularisation .its update its model only on mistakes.

3.4.4 Passive Aggressive Classifier: The passive-aggressive algorithms are a family of algorithms for large-scale learning. They are similar to the Perceptron in that they do not require a learning rate. However, contrary to the Perceptron, they include a regularization parameter C.

3.4.5 SGD Classifier: Stochastic gradient descent is used in large scale learning in text classification and NLP .its advantage are its efficiency and ease of implementation .it implements a plain stochastic gradient learning routine which supports different loss functions

3.4.6 Linear Support vector classification : Linear SVC is created from support vector machine method of machine learning Advantages of using SVM are

Effective in high dimensional spaces.

Still effective in cases where number of dimensions is greater than the number of samples.

Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

3.4.7 K Neighbour Classifier: Neighbors-based classification is a type of *instance-based learning* or *non-generalizing learning*: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point

3.4.8 Nearest Centroid: Nearest Centroid classifier is a simple algorithm that represent each class by the centroid of its members. In effect, this makes it similar to the label updating phase of the **sklearn.cluster.KMeans** algorithm. It also has no parameters to choose, making it a good baseline classifier.

3.4.9 Bernoulli Naive Bayes: implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, Boolean) variable.

The decision rule for Bernoulli naive Bayes is based on

$$P(x_i \mid y) = P(i \mid y) x_i + (1 - P(i \mid y)) (1 - x_i)$$

3.4.10 Multinomial NB: implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice).

3.4.11.AdaBoostClassifier: The core principle of AdaBoost is to fit a sequence of weak learners (i.e., models that are only slightly better than random guessing, such as small decision trees) on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The data modifications at each so-called boosting iteration consist of applying weights $w1,w2,\ldots wn$ to each of the training samples. Initially, those weights are all set to , so that the first step simply trains a weak learner on the original data. For each successive iteration, the sample weights are individually modified and the learning algorithm is reapplied to the reweighted data. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence

3.5 Evaluation Metrices:

for our project we have taken four parameter which were previously described in this report which are F1-score ,precision ,accuracy and Recall.

Accuracy :accuracy simply in machine learning means division between the number of correct predictions by total number of input samples

Precision : Precision talks about how precise/accurate your model is out of those predicted positive, how many of them are actual positive.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Figure 14: formula of precision

Precision is a good measure to determine, when the costs of False Positive is high.

Recall: Recall is defined by the below formula. Recall actually calculates how many of the Actual Positives our model capture through labelling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

Figure 15:formula of recall

F1-score: the **F_1 score** (also **F-score** or **F-measure**) is a measure of a test's accuracy. It considers both the precision *p* and the recall *r* of the test to compute the score: *p* is the number of correct positive results divided by the number of all positive results returned by the classifier, and *r* is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

The F_1 score is the harmonic mean of the precision and recall, where an F_1 score reaches its best value at 1 (perfect precision and recall)

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Figure 16:formula of f1-score

4.Experiment:

To achieve the target of the project we have implemented various step in implementation which will be described in this Experiment section starting with the training of feature vectors using various algorithms to find the best algorithm for sentimental analysis .Together with development set ,testing set and training set first of experiment performed will be discussed below :

As previously discussed, we have taken textBlob as a baseline so performing sentimental analysis function on dataset, we have applied some line of code to achieve accuracy score

```
conmat = np.array(confusion_matrix(y_validation, tbpred,
labels=[1,0]))

confusion = pd.DataFrame(conmat, index=['positive', 'negative'],
columns=['predicted_positive', 'predicted_negative'])

print("Accuracy score: {0:.2f}
%".format(accuracy_score(y_validation, tbpred)*100))
```

After applying implementation of code for accuracy score of model is 61.41 % as well as other parameter results are :

```
Confusion Matrix
          predicted_positive  predicted_negative
positive                7136                 818
negative                5344                2669
```

Classification report consists of our Evaluation

```
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.33      0.46      8013
           1       0.57      0.90      0.70      7954

   micro avg       0.61      0.61      0.61     15967
   macro avg       0.67      0.62      0.58     15967
weighted avg       0.67      0.61      0.58     15967
```

We have created function which will help in calculating nfeatures checker to find the maximum accuracy at n feature

```python
def nfeature_accuracy_checker(vectorizer = cvec, n_features = n_features, stop_words = None,
                              ngram_range = (1,1), classifier = lr):
    result = []
    print(classifier, "\n")
    for n in n_features:
        vectorizer.set_params(stop_words = stop_words, max_features = n, ngram_range=ngram_range)
        checker_pipeline = Pipeline([('vectorizer', vectorizer), ('classifier', classifier)])
        print("Validation result for {0} features".format(n))
        nfeature_accuracy, ttime = accuracy_summary(checker_pipeline, x_train, y_train, x_validation, y_validation)
        result.append((n, nfeature_accuracy, ttime))
    return result
```

Figure 17 :function of nfeatures_accuracy_checker

After doing the sentiment analysis using textBlob now we use bag of word for nfeature accuracy checker with various ngram for examples :Unigram without Stop words ,unigram with stop word without custom stop word after calculating we have created plot to showcase the result in below figure:

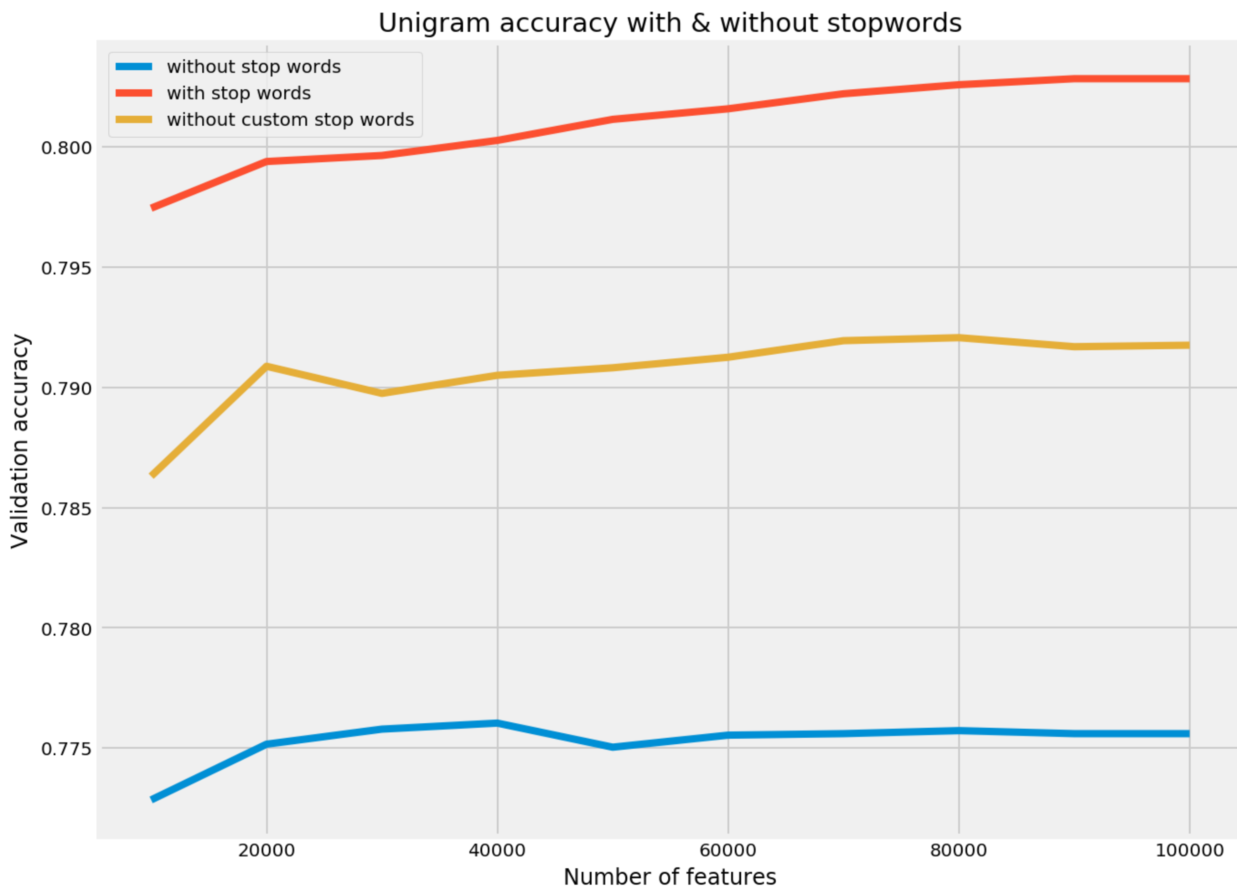

Figure 18:Unigram accuracy with or without strop words

The above graph shows the removal of stopword does not help in improvement of the model. In this setting, keeping the stopwords improve the model performance.

Now we have decided from our calculation that keeping the stopword helps in attaining the maximum result for the model now we calculate the nfeatureCheckers for bigram and trigram

After calculating for uni ,bi and tri gram the below figure will show the result :

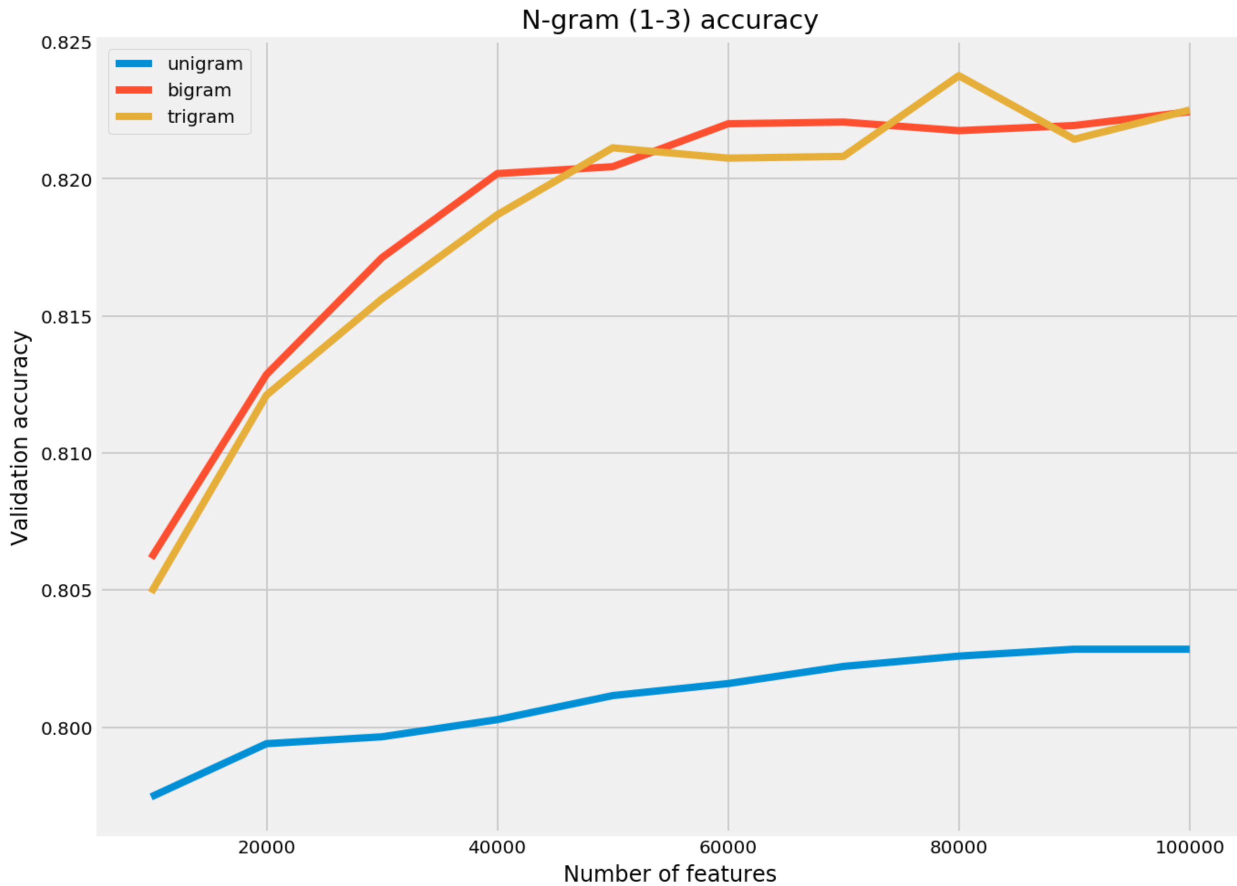

Figure 19:N-gram(1-3) accuracy

Here, unigram has maximum accuracy at 100000 features, bigram has maximum accuracy at 70000 features and trigram has maximum accuracy at 80000 features.

So, we calculate our result of model using logistic regression using the above result which shows maximum accuracy at specific features and find the result in our expected parameter

Result of Unigram at 100000 feature using Count Vectorizer with Logistic Regression model

```
Null accuracy: 50.18%
Accuracy: 80.28%
Model is 30.10% more accurate than null accuracy
--------------------------------------------------
CONFUSION MATRIX

          predicted_negative  predicted_positive
negative                2669                5344
positive                 818                7136
--------------------------------------------------
              precision    recall  f1-score   support

    negative       0.81      0.79      0.80      8013
```

```
    positive       0.79      0.81      0.80      7954

    accuracy                           0.80     15967
   macro avg       0.80      0.80      0.80     15967
weighted avg       0.80      0.80      0.80     15967
```

Result of bigram at 70000 feature using Countvectorizer with Logistic Regression model

```
Null accuracy: 50.18%
Accuracy: 82.21%
Model is 32.02% more accurate than null accuracy
--------------------------------------------------
CONFUSION MATRIX

          predicted_negative  predicted_positive
negative                2669                5344
positive                 818                7136
--------------------------------------------------
              precision    recall  f1-score   support

    negative       0.83      0.81      0.82      8013
    positive       0.81      0.83      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967
```

Trigram at 80000 features using Countvectorizer with Logistic Regression model

```
Null accuracy: 50.18%
Accuracy: 82.38%
Model is 32.19% more accurate than null accuracy
--------------------------------------------------
CONFUSION MATRIX

          predicted_negative  predicted_positive
negative                2669                5344
positive                 818                7136
--------------------------------------------------
              precision    recall  f1-score   support

    negative       0.83      0.81      0.82      8013
    positive       0.82      0.83      0.83      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967
```

So now we have seen the result of Count Vectorizer using logistic regression. Now we apply other classification algorithms on these feature vectors for this we have created function which automatically train the set and return classification report.

```python
null_accuracy = 0
def accuracy_summary(pipeline, x_train, y_train, x_test, y_test):
    if len(x_test[y_test==0])/len(x_test)>0.5:
        null_accuracy = len(x_test[y_test==0])/len(x_test)
    else:
        null_accuracy = 1 - len(x_test[y_test==0])/len(x_test)
    t0 = time()
    sentiment_fit = pipeline.fit(x_train, y_train)
    y_pred = sentiment_fit.predict(x_test)
    train_test_time = time() - t0
    report=classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    print("Null accuracy: {0:.2f}%".format(null_accuracy*100))
    print("Accuracy: {0:.2f}%".format(accuracy*100))
    if accuracy>null_accuracy:
        print("Model is {0:.2f}% more accurate than null accuracy".format((accuracy-null_accuracy)*100))
    elif accuracy==null_accuracy:
        print("Model has the same accuracy as null accuracy")
    else:
        print("Model is {0:.2f}% less accurate than null accuracy".format((null_accuracy-accuracy)*100))
    print("Train and test time: {0:.2f}s".format(train_test_time))
    print("-"*50)
    print(report)
    return accuracy, train_test_time
```

So, to calculate to train the model using classification algorithm we have come to conclusion that to create a function for this scenario

```
def classifier_comparator(vectorizer = cvec, n_features=10000,
stop_words=None, ngram_range=(1,1), classifier=zipped_clf):

    result = []

    vectorizer.set_params(stop_words=stop_words, ngram_range=ngram_range,
max_features=n_features)

    for n, c in classifier:

        pipeline = Pipeline([('vectorizer', vectorizer), ('classifier', c)])

        print('Validation result for {}'.format(n), c)
```

```
        clf_accuracy, ttime = accuracy_summary(pipeline, x_train,
y_train,x_validation, y_validation)

        result.append((n, clf_accuracy, ttime))

    return result
```

Using the above function, we get the result of classification comparator trigram using CountVectorizor with max_feature argument at 80000 and the result is as follows:

```
Validation result for Ridge Classifier RidgeClassifier(alpha=1.0,
class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, normalize=False, random_state=None,
                solver='auto', tol=0.001)
Null accuracy: 50.18%
Accuracy: 81.86%
Model is 31.67% more accurate than null accuracy
Train and test time: 580.60s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.80      0.82      8013
           1       0.81      0.84      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for Logistic Regression LogisticRegression(C=1.0,
class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
Null accuracy: 50.18%
Accuracy: 82.38%
Model is 32.19% more accurate than null accuracy
Train and test time: 611.43s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.81      0.82      8013
           1       0.82      0.83      0.83      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967
```

```
Validation result for Perceptron Perceptron(alpha=0.0001, class_weight=None,
early_stopping=False, eta0=1.0,
           fit_intercept=True, max_iter=1000, n_iter_no_change=5, n_jobs=None,
           penalty=None, random_state=0, shuffle=True, tol=0.001,
           validation_fraction=0.1, verbose=0, warm_start=False)
Null accuracy: 50.18%
Accuracy: 75.76%
Model is 25.57% more accurate than null accuracy
Train and test time: 750.20s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.77      0.73      0.75      8013
           1       0.74      0.78      0.76      7954

    accuracy                           0.76     15967
   macro avg       0.76      0.76      0.76     15967
weighted avg       0.76      0.76      0.76     15967

Validation result for Passive-Agressive Classifier
PassiveAggressiveClassifier(C=1.0, average=False, class_weight=None,
                            early_stopping=False, fit_intercept=True,
                            loss='hinge', max_iter=1000, n_iter_no_change=5,
                            n_jobs=None, random_state=None, shuffle=True,
                            tol=0.001, validation_fraction=0.1, verbose=0,
                            warm_start=False)
Null accuracy: 50.18%
Accuracy: 75.93%
Model is 25.75% more accurate than null accuracy
Train and test time: 186.06s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.73      0.82      0.77      8013
           1       0.79      0.70      0.74      7954

    accuracy                           0.76     15967
   macro avg       0.76      0.76      0.76     15967
weighted avg       0.76      0.76      0.76     15967

Validation result for Stochastic Gradient Descent SGDClassifier(alpha=0.0001,
average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
Null accuracy: 50.18%
Accuracy: 81.39%
Model is 31.20% more accurate than null accuracy
Train and test time: 183.68s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.79      0.81      8013
           1       0.80      0.84      0.82      7954

    accuracy                           0.81     15967
   macro avg       0.81      0.81      0.81     15967
weighted avg       0.81      0.81      0.81     15967

Validation result for LinearSVC LinearSVC(C=1.0, class_weight=None, dual=True,
fit_intercept=True,
```

```
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
Null accuracy: 50.18%
Accuracy: 82.06%
Model is 31.88% more accurate than null accuracy
Train and test time: 827.04s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.84      0.80      0.82      8013
           1       0.81      0.84      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for L1 based LinearSVC Pipeline(memory=None,
         steps=[('feature_selection',
                 SelectFromModel(estimator=LinearSVC(C=1.0, class_weight=None,
                                                     dual=False,
                                                     fit_intercept=True,
                                                     intercept_scaling=1,
                                                     loss='squared_hinge',
                                                     max_iter=1000,
                                                     multi_class='ovr',
                                                     penalty='l1',
                                                     random_state=None,
                                                     tol=0.0001, verbose=0),
                                 max_features=None, norm_order=1,
prefit=False,
                                 threshold=None)),
                ('classification',
                 LinearSVC(C=1.0, class_weight=None, dual=True,
                           fit_intercept=True, intercept_scaling=1,
                           loss='squared_hinge', max_iter=1000,
                           multi_class='ovr', penalty='l2', random_state=None,
                           tol=0.0001, verbose=0))],
         verbose=False)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
Null accuracy: 50.18%
Accuracy: 82.14%
Model is 31.95% more accurate than null accuracy
Train and test time: 1220.14s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.84      0.80      0.82      8013
           1       0.81      0.84      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for KNN KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
```

```
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
Null accuracy: 50.18%
Accuracy: 71.87%
Model is 21.69% more accurate than null accuracy
Train and test time: 1968.04s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.75      0.66      0.70      8013
           1       0.69      0.78      0.73      7954

    accuracy                           0.72     15967
   macro avg       0.72      0.72      0.72     15967
weighted avg       0.72      0.72      0.72     15967

Validation result for Nearest Centroid NearestCentroid(metric='euclidean',
shrink_threshold=None)
Null accuracy: 50.18%
Accuracy: 63.78%
Model is 13.60% more accurate than null accuracy
Train and test time: 287.85s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.66      0.57      0.61      8013
           1       0.62      0.70      0.66      7954

    accuracy                           0.64     15967
   macro avg       0.64      0.64      0.64     15967
weighted avg       0.64      0.64      0.64     15967

Validation result for Multinomial NB MultinomialNB(alpha=1.0,
class_prior=None, fit_prior=True)
Null accuracy: 50.18%
Accuracy: 79.73%
Model is 29.55% more accurate than null accuracy
Train and test time: 251.46s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      8013
           1       0.80      0.79      0.80      7954

    accuracy                           0.80     15967
   macro avg       0.80      0.80      0.80     15967
weighted avg       0.80      0.80      0.80     15967

Validation result for Bernoulli NB BernoulliNB(alpha=1.0, binarize=0.0,
class_prior=None, fit_prior=True)
Null accuracy: 50.18%
Accuracy: 79.38%
Model is 29.19% more accurate than null accuracy
Train and test time: 259.98s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.81      0.77      0.79      8013
           1       0.78      0.82      0.80      7954

    accuracy                           0.79     15967
   macro avg       0.79      0.79      0.79     15967
weighted avg       0.79      0.79      0.79     15967
```

```
Validation result for Adaboost AdaBoostClassifier(algorithm='SAMME.R',
base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None)
Null accuracy: 50.18%
Accuracy: 70.23%
Model is 20.05% more accurate than null accuracy
Train and test time: 540.20s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.74      0.62      0.68      8013
           1       0.67      0.78      0.72      7954

    accuracy                           0.70     15967
   macro avg       0.71      0.70      0.70     15967
weighted avg       0.71      0.70      0.70     15967
```

Now we have seen in figure that bigram have its maximum accuracy at the 70000 features so we classification comparator on bigram of bag of words at maxfeatures equals to 70000

```
Validation result for Ridge Classifier RidgeClassifier(alpha=1.0, class_weight
=None, copy_X=True, fit_intercept=True,
                max_iter=None, normalize=False, random_state=None,
                solver='auto', tol=0.001)
Null accuracy: 50.18%
Accuracy: 81.77%
Model is 31.59% more accurate than null accuracy
Train and test time: 737.55s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.80      0.81      8013
           1       0.80      0.84      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for Logistic Regression LogisticRegression(C=1.0, class_weig
ht=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
```

```
  FutureWarning)
Null accuracy: 50.18%
Accuracy: 82.21%
Model is 32.02% more accurate than null accuracy
Train and test time: 843.42s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.81      0.82      8013
           1       0.81      0.83      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for Perceptron Perceptron(alpha=0.0001, class_weight=None, e
arly_stopping=False, eta0=1.0,
           fit_intercept=True, max_iter=1000, n_iter_no_change=5, n_jobs=None,
           penalty=None, random_state=0, shuffle=True, tol=0.001,
           validation_fraction=0.1, verbose=0, warm_start=False)
Null accuracy: 50.18%
Accuracy: 73.78%
Model is 23.59% more accurate than null accuracy
Train and test time: 95.94s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.79      0.64      0.71      8013
           1       0.70      0.83      0.76      7954

    accuracy                           0.74     15967
   macro avg       0.75      0.74      0.74     15967
weighted avg       0.75      0.74      0.74     15967

Validation result for Passive-Agressive Classifier PassiveAggressiveClassifier
(C=1.0, average=False, class_weight=None,
                            early_stopping=False, fit_intercept=True,
                            loss='hinge', max_iter=1000, n_iter_no_change=5,
                            n_jobs=None, random_state=None, shuffle=True,
                            tol=0.001, validation_fraction=0.1, verbose=0,
                            warm_start=False)
Null accuracy: 50.18%
Accuracy: 75.54%
Model is 25.36% more accurate than null accuracy
Train and test time: 91.51s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.74      0.78      0.76      8013
```

```
           1       0.77      0.73      0.75      7954

    accuracy                           0.76     15967
   macro avg       0.76      0.76      0.76     15967
weighted avg       0.76      0.76      0.76     15967

Validation result for Stochastic Gradient Descent SGDClassifier(alpha=0.0001,
average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
Null accuracy: 50.18%
Accuracy: 80.85%
Model is 30.67% more accurate than null accuracy
Train and test time: 89.56s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.78      0.80      8013
           1       0.79      0.84      0.81      7954

    accuracy                           0.81     15967
   macro avg       0.81      0.81      0.81     15967
weighted avg       0.81      0.81      0.81     15967

Validation result for LinearSVC LinearSVC(C=1.0, class_weight=None, dual=True,
fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
Null accuracy: 50.18%
Accuracy: 81.81%
Model is 31.62% more accurate than null accuracy
Train and test time: 748.96s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.84      0.79      0.81      8013
           1       0.80      0.84      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967
```

```
Validation result for L1 based LinearSVC Pipeline(memory=None,
         steps=[('feature_selection',
                 SelectFromModel(estimator=LinearSVC(C=1.0, class_weight=None,
                                                     dual=False,
                                                     fit_intercept=True,
                                                     intercept_scaling=1,
                                                     loss='squared_hinge',
                                                     max_iter=1000,
                                                     multi_class='ovr',
                                                     penalty='l1',
                                                     random_state=None,
                                                     tol=0.0001, verbose=0),
                                 max_features=None, norm_order=1, prefit=False
,
                                 threshold=None)),
                ('classification',
                 LinearSVC(C=1.0, class_weight=None, dual=True,
                           fit_intercept=True, intercept_scaling=1,
                           loss='squared_hinge', max_iter=1000,
                           multi_class='ovr', penalty='l2', random_state=None,
                           tol=0.0001, verbose=0))],
         verbose=False)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929: Convergenc
eWarning: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
Null accuracy: 50.18%
Accuracy: 81.84%
Model is 31.66% more accurate than null accuracy
Train and test time: 1018.56s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.84      0.79      0.81      8013
           1       0.80      0.84      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for KNN KNeighborsClassifier(algorithm='auto', leaf_size
=30, metric='minkowski',
                      metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                      weights='uniform')
Null accuracy: 50.18%
Accuracy: 72.13%
Model is 21.95% more accurate than null accuracy
Train and test time: 2430.90s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.76      0.66      0.70      8013
```

```
           1       0.69      0.79      0.74      7954

    accuracy                           0.72     15967
   macro avg       0.73      0.72      0.72     15967
weighted avg       0.73      0.72      0.72     15967

Validation result for Nearest Centroid NearestCentroid(metric='euclidean',
shrink_threshold=None)
Null accuracy: 50.18%
Accuracy: 63.70%
Model is 13.52% more accurate than null accuracy
Train and test time: 158.58s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.66      0.57      0.61      8013
           1       0.62      0.70      0.66      7954

    accuracy                           0.64     15967
   macro avg       0.64      0.64      0.64     15967
weighted avg       0.64      0.64      0.64     15967

Validation result for Multinomial NB MultinomialNB(alpha=1.0, class_prior=
None, fit_prior=True)
Null accuracy: 50.18%
Accuracy: 79.78%
Model is 29.59% more accurate than null accuracy
Train and test time: 126.78s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.79      0.81      0.80      8013
           1       0.80      0.79      0.80      7954

    accuracy                           0.80     15967
   macro avg       0.80      0.80      0.80     15967
weighted avg       0.80      0.80      0.80     15967

Validation result for Bernoulli NB BernoulliNB(alpha=1.0, binarize=0.0, cl
ass_prior=None, fit_prior=True)
Null accuracy: 50.18%
Accuracy: 79.67%
Model is 29.49% more accurate than null accuracy
Train and test time: 123.32s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.81      0.78      0.79      8013
           1       0.79      0.81      0.80      7954

    accuracy                           0.80     15967
   macro avg       0.80      0.80      0.80     15967
weighted avg       0.80      0.80      0.80     15967

Validation result for Adaboost AdaBoostClassifier(algorithm='SAMME.R', bas
e_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None)
Null accuracy: 50.18%
Accuracy: 70.23%
Model is 20.05% more accurate than null accuracy
```

```
Train and test time: 556.87s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.74      0.62      0.68      8013
           1       0.67      0.78      0.72      7954

    accuracy                           0.70     15967
   macro avg       0.71      0.70      0.70     15967
weighted avg       0.71      0.70      0.70     15967
```

Now we have seen bag of words feature vectors results then our next experiment is on Tf-idf feature Extraction as discussed earlier in feature Extraction section .All the procedures used in experiment of bag of words are followed in case of Tf-idf So we performed the nfeature_accuracy_checker function on unigram, bigram and trigram using tfidf feature extraction and outcome figure is as follows:

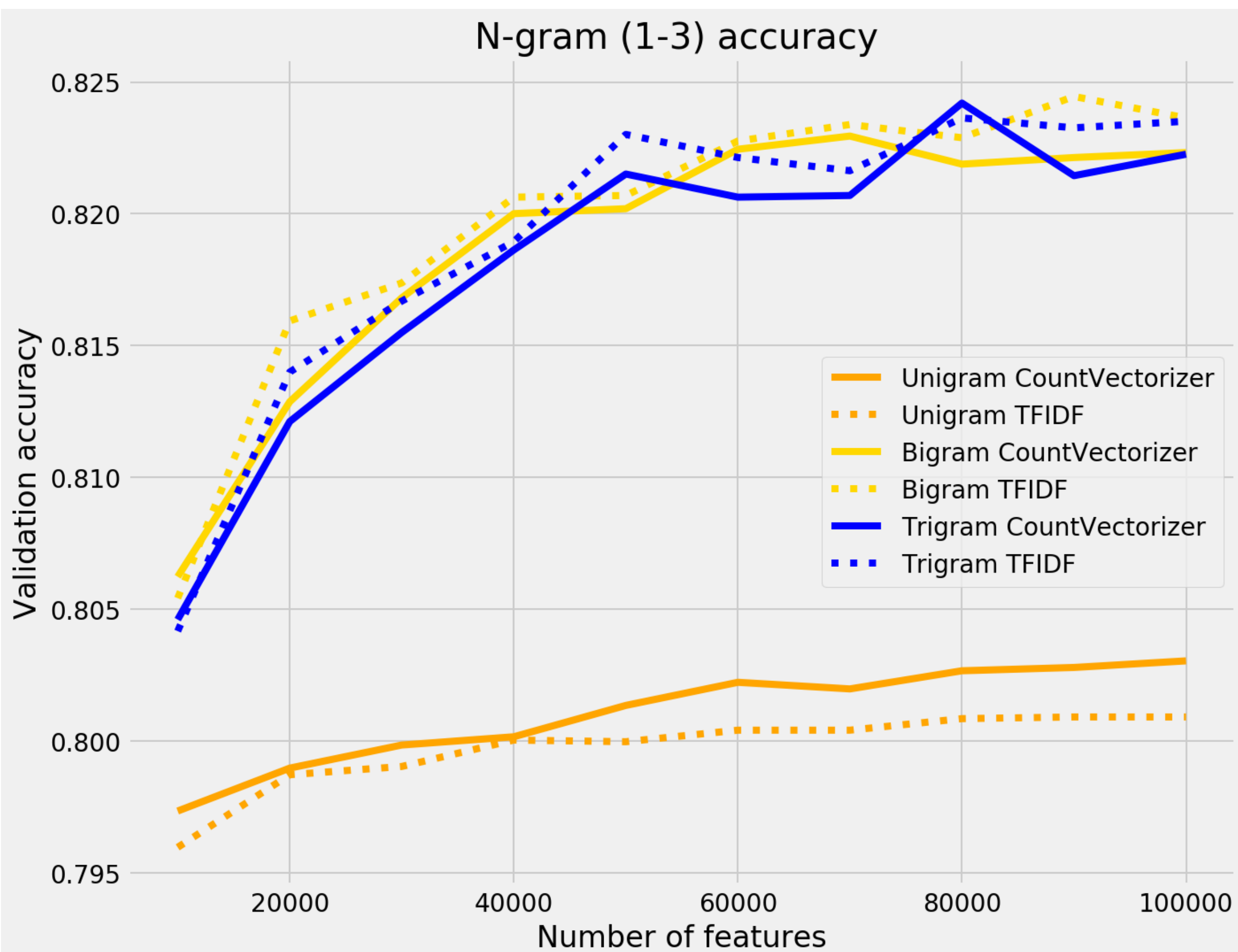

Figure 20:N-gram (1-3) accuracy

Hence, we can clearly see that using bigram and trigrams boosts the performance of the model in Count Vectorizer and Tfidf Vectorizer both. Also, for bigram and trigram, Tfidf Vectorizer gives better performance than Count Vectorizer. Bigram Tfidf Vectorizer at 90000 features gives the highest validation accuracy at 82.45%.

After applying classification comparator, the following outcome came:

```
Validation result for Ridge Classifier RidgeClassifier(alpha=1.0,
class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, normalize=False, random_state=None,
                solver='auto', tol=0.001)
Null accuracy: 50.18%
Accuracy: 82.29%
Model is 32.10% more accurate than null accuracy
Train and test time: 177.09s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.81      0.82      8013
           1       0.81      0.84      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for Logistic Regression LogisticRegression(C=1.0,
class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
C:\ProgramData\Anaconda3\lib\site-
packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
Null accuracy: 50.18%
Accuracy: 82.43%
Model is 32.24% more accurate than null accuracy
Train and test time: 178.62s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.82      0.82      8013
           1       0.82      0.83      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967
```

```
Validation result for Perceptron Perceptron(alpha=0.0001, class_weight=None,
early_stopping=False, eta0=1.0,
           fit_intercept=True, max_iter=1000, n_iter_no_change=5, n_jobs=None,
           penalty=None, random_state=0, shuffle=True, tol=0.001,
           validation_fraction=0.1, verbose=0, warm_start=False)
Null accuracy: 50.18%
Accuracy: 76.39%
Model is 26.20% more accurate than null accuracy
Train and test time: 113.19s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.77      0.76      0.76      8013
           1       0.76      0.76      0.76      7954

    accuracy                           0.76     15967
   macro avg       0.76      0.76      0.76     15967
weighted avg       0.76      0.76      0.76     15967

Validation result for Passive-Agressive Classifier
PassiveAggressiveClassifier(C=1.0, average=False, class_weight=None,
                            early_stopping=False, fit_intercept=True,
                            loss='hinge', max_iter=1000, n_iter_no_change=5,
                            n_jobs=None, random_state=None, shuffle=True,
                            tol=0.001, validation_fraction=0.1, verbose=0,
                            warm_start=False)
Null accuracy: 50.18%
Accuracy: 79.86%
Model is 29.68% more accurate than null accuracy
Train and test time: 115.51s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.80      0.81      0.80      8013
           1       0.80      0.79      0.80      7954

    accuracy                           0.80     15967
   macro avg       0.80      0.80      0.80     15967
weighted avg       0.80      0.80      0.80     15967

Validation result for Stochastic Gradient Descent SGDClassifier(alpha=0.0001,
average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=None, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
Null accuracy: 50.18%
Accuracy: 78.71%
Model is 28.53% more accurate than null accuracy
Train and test time: 111.60s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.80      0.77      0.78      8013
           1       0.78      0.80      0.79      7954

    accuracy                           0.79     15967
   macro avg       0.79      0.79      0.79     15967
weighted avg       0.79      0.79      0.79     15967
```

```
Validation result for LinearSVC LinearSVC(C=1.0, class_weight=None, dual=True,
fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
Null accuracy: 50.18%
Accuracy: 82.26%
Model is 32.08% more accurate than null accuracy
Train and test time: 838.67s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.82      0.83      0.82      8013
           1       0.83      0.81      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967

Validation result for L1 based LinearSVC Pipeline(memory=None,
         steps=[('feature_selection',
                SelectFromModel(estimator=LinearSVC(C=1.0, class_weight=None,
                                                    dual=False,
                                                    fit_intercept=True,
                                                    intercept_scaling=1,
                                                    loss='squared_hinge',
                                                    max_iter=1000,
                                                    multi_class='ovr',
                                                    penalty='l1',
                                                    random_state=None,
                                                    tol=0.0001, verbose=0),
                                max_features=None, norm_order=1,
prefit=False,
                                threshold=None)),
                ('classification',
                LinearSVC(C=1.0, class_weight=None, dual=True,
                          fit_intercept=True, intercept_scaling=1,
                          loss='squared_hinge', max_iter=1000,
                          multi_class='ovr', penalty='l2', random_state=None,
                          tol=0.0001, verbose=0))],
         verbose=False)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
Null accuracy: 50.18%
Accuracy: 82.41%
Model is 32.22% more accurate than null accuracy
Train and test time: 989.29s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.82      0.83      0.83      8013
           1       0.83      0.81      0.82      7954

    accuracy                           0.82     15967
   macro avg       0.82      0.82      0.82     15967
weighted avg       0.82      0.82      0.82     15967
```

```
Validation result for KNN KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
Null accuracy: 50.18%
Accuracy: 62.55%
Model is 12.37% more accurate than null accuracy
Train and test time: 1872.75s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.69      0.45      0.55      8013
           1       0.59      0.80      0.68      7954

    accuracy                           0.63     15967
   macro avg       0.64      0.63      0.61     15967
weighted avg       0.64      0.63      0.61     15967

Validation result for Nearest Centroid NearestCentroid(metric='euclidean',
shrink_threshold=None)
Null accuracy: 50.18%
Accuracy: 72.55%
Model is 22.36% more accurate than null accuracy
Train and test time: 120.74s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.72      0.74      0.73      8013
           1       0.73      0.71      0.72      7954

    accuracy                           0.73     15967
   macro avg       0.73      0.73      0.73     15967
weighted avg       0.73      0.73      0.73     15967

Validation result for Multinomial NB MultinomialNB(alpha=1.0,
class_prior=None, fit_prior=True)
Null accuracy: 50.18%
Accuracy: 80.15%
Model is 29.97% more accurate than null accuracy
Train and test time: 106.64s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.80      0.81      0.80      8013
           1       0.81      0.79      0.80      7954

    accuracy                           0.80     15967
   macro avg       0.80      0.80      0.80     15967
weighted avg       0.80      0.80      0.80     15967

Validation result for Bernoulli NB BernoulliNB(alpha=1.0, binarize=0.0,
class_prior=None, fit_prior=True)
Null accuracy: 50.18%
Accuracy: 79.91%
Model is 29.73% more accurate than null accuracy
Train and test time: 106.89s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.81      0.78      0.80      8013
           1       0.79      0.81      0.80      7954

    accuracy                           0.80     15967
```

```
   macro avg       0.80      0.80      0.80     15967
weighted avg       0.80      0.80      0.80     15967

Validation result for Adaboost AdaBoostClassifier(algorithm='SAMME.R',
base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=None)
Null accuracy: 50.18%
Accuracy: 70.23%
Model is 20.05% more accurate than null accuracy
Train and test time: 765.85s
--------------------------------------------------
              precision    recall  f1-score   support

           0       0.75      0.62      0.68      8013
           1       0.67      0.79      0.72      7954

    accuracy                           0.70     15967
   macro avg       0.71      0.70      0.70     15967
weighted avg       0.71      0.70      0.70     15967
```

5. Result:

The result obtained from these Experiment are kept in form of table as we have used two feature vector which are bag of words and term frequency and inverse Document frequency and it showed the which ngram is best for the sentimental model and three tables below showcase our results

Classification algorithm	Negative or positive	Accuracy	F1-score	precision	recall
Ridge Classifier	Negative	82.29%	0.82	0.83	0.81
	positive		0.82	0.81	0.84
Logistic Regression	Negative	82.43%	0.82	0.83	0.82
	positive		0.82	0.82	0.83
Perceptron	Negative	76.39%	0.76	0.77	0.76
	positive		0.76	0.76	0.76
Passive-Aggressive Classifier	Negative	79.86%	0.80	0.80	0.81
	positive		0.80	0.80	0.79
Stochastic Gradient Descent	Negative	78.71%	0.78	0.80	0.77
	positive		0.79	0.78	0.80
LinearSVC	Negative	82.26%	0.82	0.82	0.83

	positive		0.82	0.83	0.81
L1 based LinearSVC	Negative	82.41%	0.83	0.82	0.83
	positive		0.82	0.83	0.81
KNN KNeighborsClassifier	Negative	62.55%	0.55	0.69	0.45
	positive		0.68	0.59	0.80
Nearest Centroid	Negative	72.55%	0.73	0.72	0.74
	positive		0.72	0.73	0.71
Bernoulli NB	Negative	79.91%	0.80	0.81	0.78
	positive		0.80	0.79	0.81
AdaBoostClassifier	Negative	70.23%	0.68	0.75	0.62
	positive		0.72	0.67	0.79
Multinomial NB	Negative	80.15%	0.80	0.80	.81
	positive		0.80	0.81	0.79

Table 13: Result of performance of Classifier onTf-idf bigram

Classification algorithm	Negative or positive	Accuracy	F1-score	precision	recall
Ridge Classifier	Negative	81.86%	0.82	0.83	0.80
	positive		0.82	0.81	0.84
Logistic Regression	Negative	82.38%	0.82	0.83	0.81
	positive		0.83	0.82	0.83
Perceptron	Negative	75.76%	0.75	0.77	0.73
	positive		0.76	0.74	0.78
Passive-Aggressive Classifier	Negative	75.93%	0.77	0.73	0.82
	positive		0.74	0.79	0.70
Stochastic Gradient Descent	Negative	81.39%	0.81	0.83	0.79
	positive		0.82	0.80	0.84
LinearSVC	Negative	82.06%	0.82	0.84	0.80
	positive		0.82	0.81	0.84
L1 based LinearSVC	Negative	82.14%	0.82	0.84	0.80

	positive		0.82	0.81	0.84
KNN KNeighborsClassifier	Negative	71.87%	0.70	0.75	0.66
	positive		0.73	0.69	0.78
Nearest Centroid	Negative	63.78%	0.61	0.66	0.57
	positive		0.66	0.62	0.70
Bernoulli NB	Negative	79.38%	0.79	0.81	0.77
	positive		0.80	0.78	0.82
AdaBoostClassifier	Negative	70.23%	0.68	0.74	0.62
	positive		0.72	0.67	0.78
Multinomial NB	Negative	79.73%	0.80	0.79	.81
	positive		0.80	0.80	0.79

Table 14 : Result of performance of Classifier bag of words trigram

Classification algorithm	Negative or positive	Accuracy	F1-score	precision	recall
Ridge Classifier	Negative	81.77%	0.81	0.83	0.80
	positive		0.82	0.80	0.84
Logistic Regression	Negative	82.21%	0.82	0.83	0.81
	positive		0.82	0.81	0.83
Perceptron	Negative	73.78%	0.71	0.79	0.64
	positive		0.76	0.70	0.83
Passive-Aggressive Classifier	Negative	75.54%	0.76	0.74	0.78
	positive		0.75	0.77	0.73
Stochastic Gradient Descent	Negative	80.85%	0.80	0.83	0.78
	positive		0.81	0.79	0.84
LinearSVC	Negative	81.81%	0.81	0.84	0.79
	positive		0.82	0.80	0.84
L1 based LinearSVC	Negative	81.84%	0.81	0.84	0.79

	positive		0.82	0.80	0.84
KNN KNeighborsClassifier	Negative	72.13%	0.70	0.76	0.66
	positive		0.74	0.69	0.79
Nearest Centroid	Negative	63.70%	0.61	0.66	0.57
	positive		0.66	0.62	0.70
Bernoulli NB	Negative	79.67%	0.79	0.81	0.78
	positive		0.80	0.79	0.81
AdaBoostClassifier	Negative	70.23%	0.68	0.74	0.62
	positive		0.72	0.67	0.78
Multinomial NB	Negative	79.78%	0.80	0.79	.81
	positive		0.80	0.80	0.79

Table 15: Result of performance of Classifier bag of words bigram

Above table show the result of classifier performance on various feature vectors with performance parameter results

Below diagrams shows that how the accuracy of the of all the classification fared they have been seen using bar plot where x axis is defined as Classification algorithms and Accuracy of classifier As we have discussed before that we have created three different table in our results so we have three different bar graph to show case the result and then we have the a figure combined to show case the result

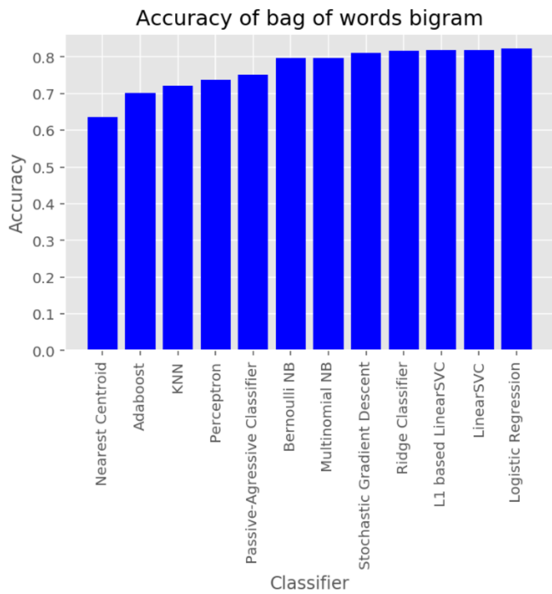

Figure 21:bar plot of accuracy vs classifier

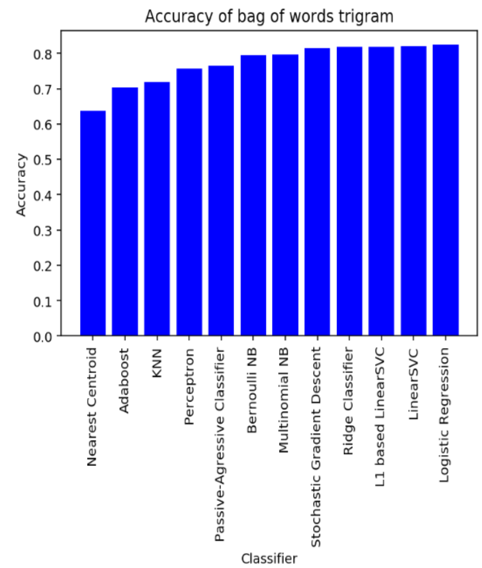

figure 22:bar plot of Accuracy vs

Classifier in bgw trigram

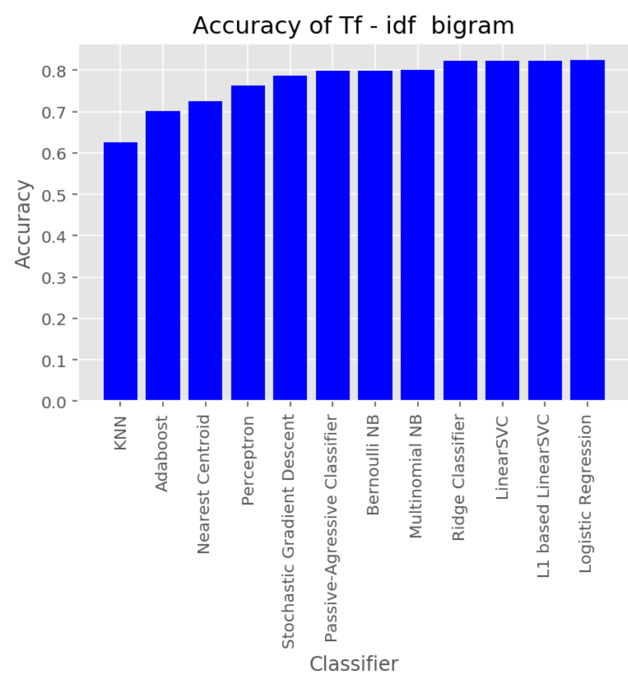

Figure 23:bar plot of Accuracy vs classifier of tf-idf bigram

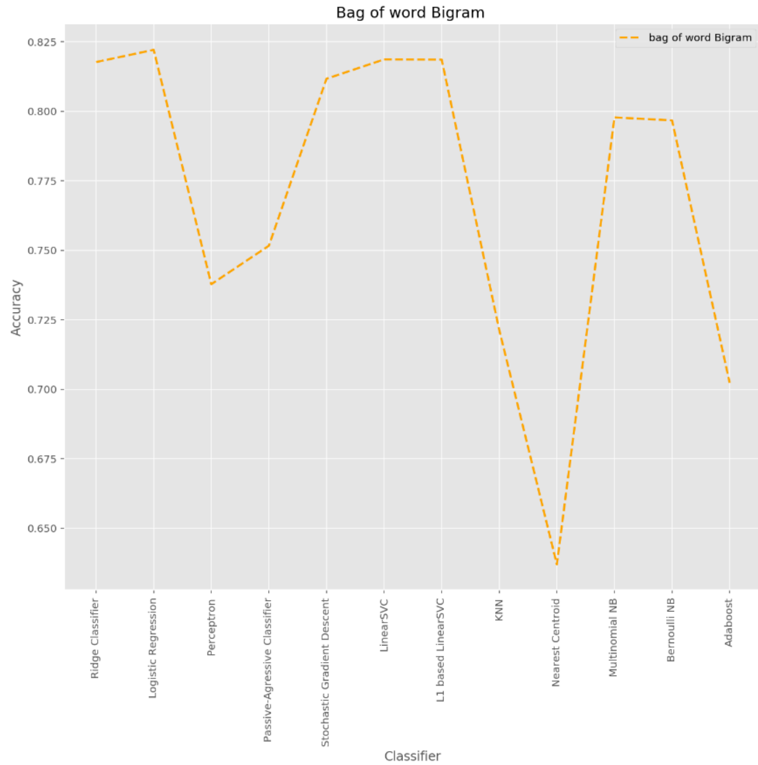

Figure 24:line plot of accuracy of classifers

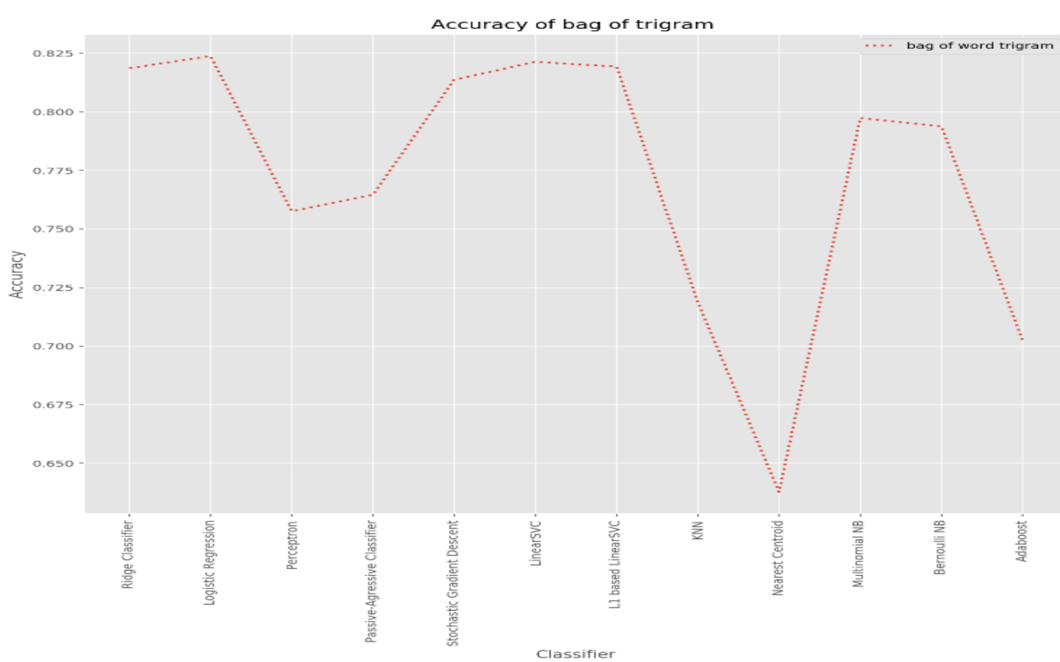

Figure 25:line plot of accuracy of bgw trigram of classifiers

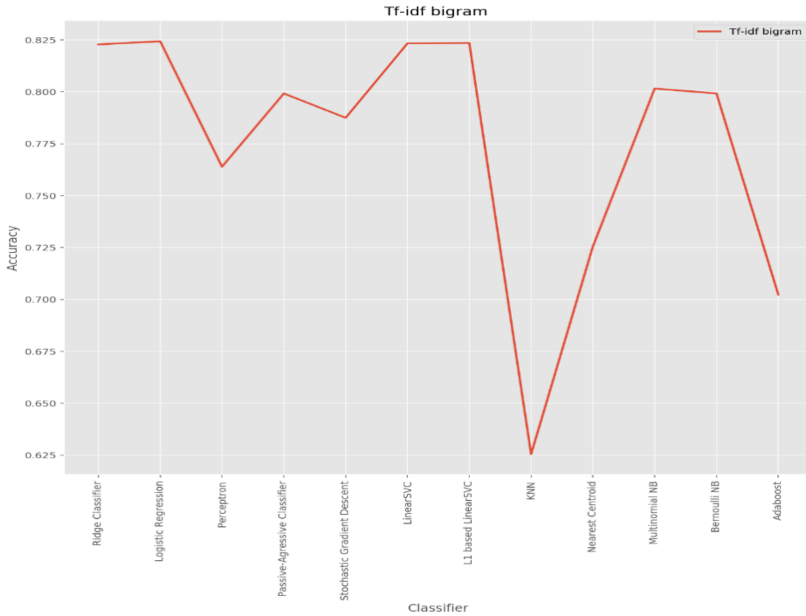

Figure 26:Accuracy of tf-idf bigram

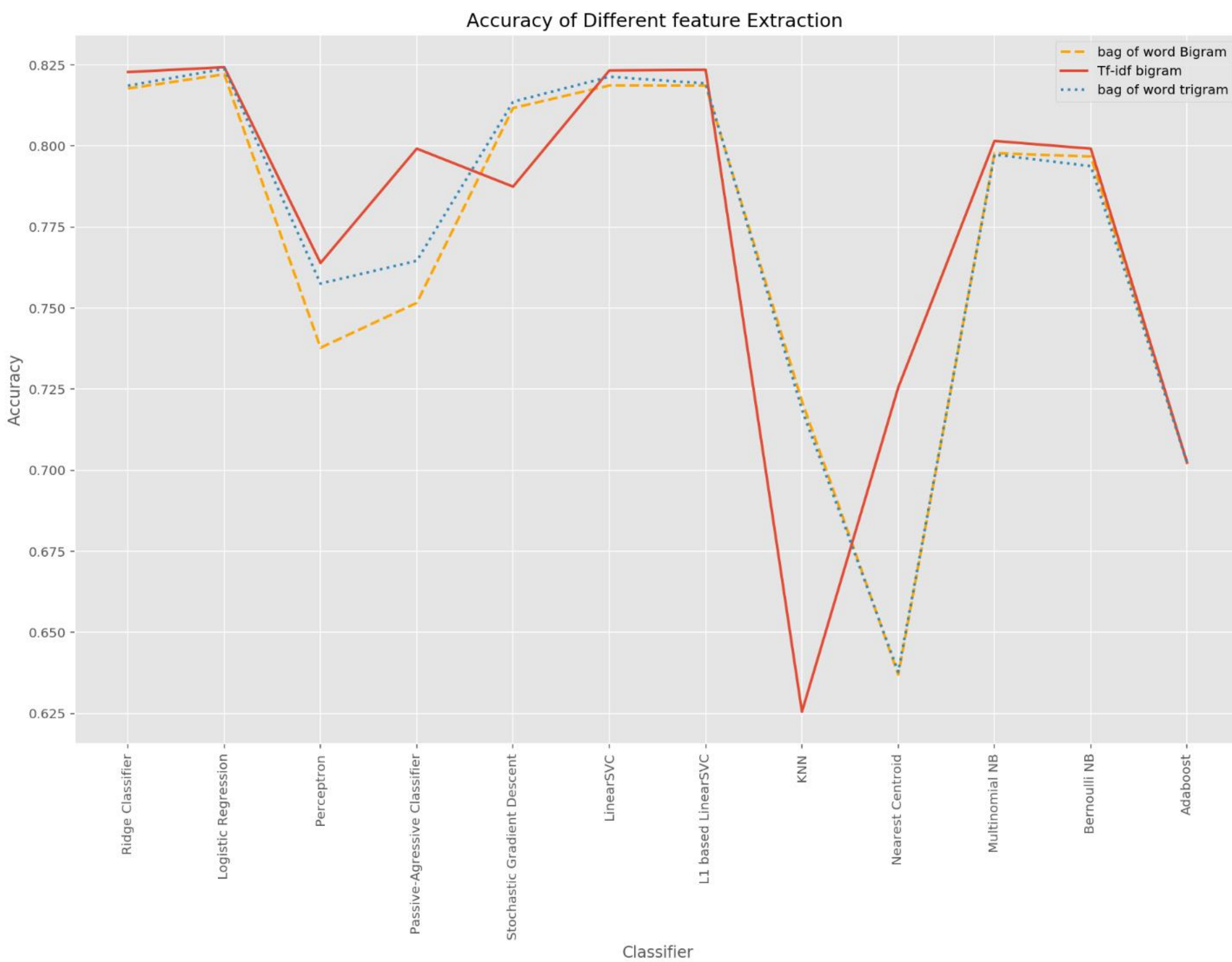

Figure 27: Accuracy of different Feature Extraction

6.Conclusion:

This paper addresses the task of sentimental analysis by developing using machine learning algorithm our system analyses the tweets or comments based on several features to determine the features to select feature extraction and classification my further work will be dependent feature combination to find the more accurate result of performance

7.Future Scopes:

my further work will be dependent feature combination to find the more accurate result of performance

8.References:

[1]. Efthymios Koulompis, Theresa Wilson, Johanna Moore (2011), “ Twitter Sentiment Analysis: The Good the Bad and the OMG!,” in: The fifth International AAAI Conference on Weblogs and Social Media.

[2]. G. A. Miller, R. Beckwith, C. D. Fell Baum, D. Gross, K.Miller. 1990. WordNet: An online lexical database. Int. J. Lexicographic. 3, 4, pp. 235–244

[3]. Saif, Hassan; He, Yulan and Alani, Harith (2012), “Semantic sentiment analysis of twitter,” in: The 11th International Semantic Web Conference (ISWC 2012), 11-15 November 2012, Boston, MA, USA.

[4]. Alec Go, Richa Bhayani and Lei Huaug, Stanford university(2009), “Twitter Sentiment Classification using Distant Supervision ,” in: The Third International Conference on Data Analytics.

[5]. A. Kumar and T.M. Sebastian, "Machine Learning assisted Sentiment Analysis". Proceedings of International Conference on computer science and engineering (ICCSE'2012), 2012.

[6]. Bifet and E. Frank, "Sentiment Knowledge Discovery In Twitter Streaming Data", In proceedings of 13th International Conference of Discovery Science, Berlin, Germany : Springer, 2010

[7]. Bo Pang and Lillian Lee, Opinion mining and sentiment Analysis

[8]. Tom M. Mitchell, generative and discriminative classifiers: Naive Bayes and Logistic Regression

[9]. Christopher M. Bishop, Pattern Recognition and Machine Learning

[10] Rosenthal, Sara, Noura Farra, and Preslav Nakov. "SemEval-2017 task 4: Sentiment analysis in Twitter." Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017). 2017.

[11] Pontiki, Maria, et al. "SemEval-2016 task 5: Aspect based sentiment analysis." Preworkshop on Semantic Evaluation (SemEval-2016). Association for Computational Linguistics, 2016.

[12] Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques." Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10. Association for Computational Linguistics, 2002.

[13] Go, Alec, Richa Bhayani, and Lei Huang. "Twitter sentiment classification using distant supervision." CS224N Project Report, Stanford 1.2009 (2009)

[14] Mohammad, Saif M., Svetlana Kiritchenko, and Xiao Dan Zhu. "NRC- Canada: Building the state-of-the-art in sentiment analysis of tweets." arXiv preprint arXiv:1308.6242 (2013). [15] Yang, Ang, et al. "Enhanced Twitter Sentiment Analysis by Using Feature Selection and Combination." Security and Privacy in Social Networks and Big Data (SocialSec), 2015 International Symposium on. IEEE, 2015.

[16] Fang, Xing, and Justin Zhan. "Sentiment analysis using product review data." Journal of Big Data 2.1 (2015): 5.

[17] Baccianella, Stefano, Andrea Esuli, and Fabrizio Sebastiani. "SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining." In LREC, vol. 10, pp. 2200-2204. 2010.

[18] Hutto, Clayton J., and Eric Gilbert. "Vader: A parsimonious rule-based model for sentiment analysis of social media text." In Eighth international AAAI conference on weblogs and social media. 2014.

[19] G. H. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in Proceedings of the Eleventh conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[20] Zhu, Shenghuo, et al. "Multi-labelled classification using maximum entropy method. "Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, 2005.

[21]. Mark Lutz , Programming python 4th Edition Trainingdata.zip Available at : http://help. sentiment140. com/for-students

[22] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," CS224N Project Report, Stanford, pp. 1–12, 2009