Abstract

In this report, we address the problem of sentiment classification on twitter dataset. We use a number of text preprocessing and Naive Bayes classifier methods to perform sentiment analysis. In the end, we use a accuracy classifying method of different text preprocessing to achieve the classification accuracy of 76.68% on Kaggle public leaderboard.

Keywords — Twitter Sentiment Analysis; Naive Bayes classifier, contraction, emoji classification, tokenization, stemming, lemmatization, stop words, count vectorizer, tf-idf vectorizer, unigram, bigram, trigram, ROC curve, precision, recall, accuracy, python 3.

Problem Statement

Twitter is a popular social networking website where members create and interact with messages known as “tweets”. This serves as a mean for individuals to express their thoughts or feelings about different subjects. Various different parties such as consumers and marketers have done sentiment analysis on such tweets to gather insights into products or to conduct market analysis. Furthermore, with the recent advancements in machine learning algorithms, we are able improve the accuracy of our sentiment analysis predictions.

In this report, we will attempt to conduct sentiment analysis on “tweets” using various different text preprocessing and Naïve Bayes Classifier algorithm. We attempt to classify the polarity of the tweet where it is either positive or negative. If the tweet has both positive and negative elements, the more dominant sentiment should be picked as the final label.

We use the dataset from Kaggle which was crawled and labeled positive/negative. The data provided comes with emoticons, usernames and hashtags which are required to be processed and converted into a standard form. We also need to extract useful features from the text such unigrams bigrams and trigrams which is a form of representation of the “tweet”. We use Naïve-Bayes Classifier to conduct sentiment analysis using the extracted features. However, just relying on an individual model did not give a high accuracy.

Dataset

The data given is in the form of a comma-separated values files with tweets and their corresponding sentiments. The training dataset is a csv file of type tweet_id, sentiment_tweet where the tweet_id is a unique integer identifying the tweet, sentiment is either 1 (positive) or 0 (negative), and tweet is the tweet enclosed in "". We will use this single dataset to cross validate our model.

The dataset is a mixture of words, emoticons, symbols, URLs, hashtags and references to people. Words and emoticons contribute to predicting the sentiment, but URLs and references to people don’t. Therefore, URLs and references can be ignored. The words are also a mixture of misspelled words, extra punctuations, and words with many repeated letters. The tweets, therefore, have to be preprocessed to standardize the dataset.

Range Index: 99989 entries, 0 to 99988

Pre-processing

Pre-processing Raw tweets scraped from twitter generally result in a noisy dataset. This is due to the casual nature of people's usage of social media. Tweets have certain special characteristics such as retweets, hashtags, emoticons, user mentions, etc. which have to be suitably extracted. Therefore, raw twitter data has to be normalized to create a dataset which can be easily learned by various classifiers. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size.

We first do some general pre-processing on tweets which is as follows.

- Convert the tweet to lower case.
- Replace 2 or more dots (.) with space.
- Strip spaces and quotes (" and ') from the ends of tweet.

We handle special twitter features as follows.

User names and mentions:

Every twitter user has a handle associated with them. Users often mention other users in their tweets by @handle. We delete all user mentions as they are unique and doesn't contribute anything to sentiment.

URL:

Users often share hyperlinks to other webpages in their tweets. Any particular URL is not important for text classification as it would lead to very sparse features. Therefore, we delete all the URLs in tweets. Thus calculation becomes less resource hungry.

Emoticon:

Users often use a number of different emoticons in their tweet to convey different emotions. It is impossible to exhaustively match all the different emoticons used on social media as the number is ever increasing. However, we match some common emoticons which are used very frequently. We replace the matched emoticons with either 'positiveemoji' or 'negetiveemoji' depending on whether it is conveying a positive or a negative emotion. A list of all emoticons matched by our method is given below.

Emoticons	Type	Replacement
:) , :) , :-) , (: , (: , (-: , :') , :O	Smile	positiveemoji
:D , : D , :-D , xD , x-D , XD , X-D	Laugh	positiveemoji
;-) , ;) , ;-D , ;D , (; , (-; , @-)	Wink	positiveemoji
<3 , :*	Love	positiveemoji
:-(, : (, :(,): ,)-: , :-/ , :-|	Sad	negetiveemoji
:,(, :'(, :"(	Cry	negetiveemoji

Hashtag:

Hashtags are un-spaced phrases prefixed by the hash symbol (#) which is frequently used by users to mention a trending topic on twitter. We replace all the hashtags with the words with the hash symbol. For example, #hello is replaced by hello.

After applying tweet level pre-processing, we processed individual words of tweets as follows.

- Strip any punctuation ['"?!,.():;] from the word.
- Convert 2 or more letter repetitions to 2 letters. Some people send tweets like 'I am sooooo happpppy' adding multiple characters to emphasize on certain words. This is done to handle such tweets by converting them to 'I am soo happy'.
- Removes any single characters.
- Remove - and '. This is done to handle words like t-shirt and their's by converting them to the more general form tshirt and theirs. Here we use contractions. We have created a separate dictionary file to handle contractions. Like 'can't' will be converted in 'cannot'.
- Replace 2 or more spaces with a single space.

Normalization

Stemming and Lemmatization are Text Normalization (or sometimes called Word Normalization) techniques in the field of Natural Language Processing that are used to prepare text, words, and documents for further processing.

Tokenization:

Tokenization refers to dividing the text into a sequence of words or sentences. We could use NLTK library, but we used string.split() which is faster.

Stemming:

Stemming refers to the removal of suffices, like "ing", "ly", "s", etc. by a simple rule-based approach. Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word. This indiscriminate cutting can be successful in some occasions, but not always, and that is why we affirm that this approach presents some limitations. For this purpose, we will use PorterStemmer from the NLTK library.

Form	Suffix	Stem
studies	-es	studi
studying	-ing	stufy

Lemmatization:

Lemmatization is a more effective option than stemming because it converts the word into its root word, rather than just stripping the suffices. It makes use of the vocabulary and does a morphological analysis to obtain the root word. Lemmatization takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries which the algorithm can look through to link the form back to its lemma. Therefore, we usually prefer using lemmatization over stemming.

Form	Morphological information	Lemma
studies	Third person, singular number, present tense of verb study	study
studying	Gerund of the verb study	study

Stop words:

Not all words determine sentiments. Some of them (mainly prepositions and articles) don't contribute anything to the sentiment. So we can remove these commonly occurring words from our text data. But NLTK library contains stop words like 'no', 'not', 'nor' etc., which determines positive or negative sentiments. So we built custom set of stop words, which doesn't contain any words that can affect sentiment.

NLTK stop words	"i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not", "only", "own", "same", "so", "than", "too", "very", "s", "t", "can", "will", "just", "don", "should", "now"
Custom stop words	"i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your", "yours", "yourself", "yourselves", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "they", "them", "their", "theirs", "themselves", "what", "which", "who", "whom", "this", "that", "these", "those", "am", "is", "are", "was", "were", "be", "been", "being", "have", "has", "had", "having", "do", "does", "did", "doing", "a", "an", "the", "and", "but", "if", "or", "because", "as", "until", "while", "of", "at", "by", "for", "with", "about", "against", "between", "into", "through", "during", "before", "after", "above", "below", "to", "from", "up", "down", "in", "out", "on", "off", "over", "under", "again", "further", "then", "once", "here", "there", "when", "where", "why", "how", "all", "any", "both", "each", "few", "more", "most", "other", "some", "such", "only", "own", "same", "so", "than", "too", "very", "can", "will", "just", "should", "now"

Feature Extraction

N-Grams:

We extract three types of features from our dataset, namely unigrams bigrams and trigrams. N-grams are the combination of multiple words used together. Ngrams with N=1 are called unigrams. Similarly, bigrams (N=2), trigrams (N=3) and so on can also be used.

Unigrams do not usually contain as much information as compared to bigrams and trigrams. The basic principle behind n-grams is that they capture the language structure, like what letter or word is likely to follow the given one. The longer the n-gram (the higher the n), the more context we have to work with. Optimum length really depends on the application – if our n-grams are too short, we may fail to capture important differences. On the other hand, if they are too long, we may fail to capture the "general knowledge" and only stick to particular cases.

Bag of Words:

Bag of Words (BoW) refers to the representation of text which describes the presence of words within the text data. The intuition behind this is that two similar text fields will contain similar kind of words, and will therefore have a similar bag of words. Further, that from the text alone we can learn something about the meaning of the document. We use CountVectorizer from sklearn for this purpose.

Term Frequency – Inverse Document Frequency:

Term frequency is simply the ratio of the count of a word present in a sentence, to the length of the sentence. Therefore, we can generalize term frequency as:

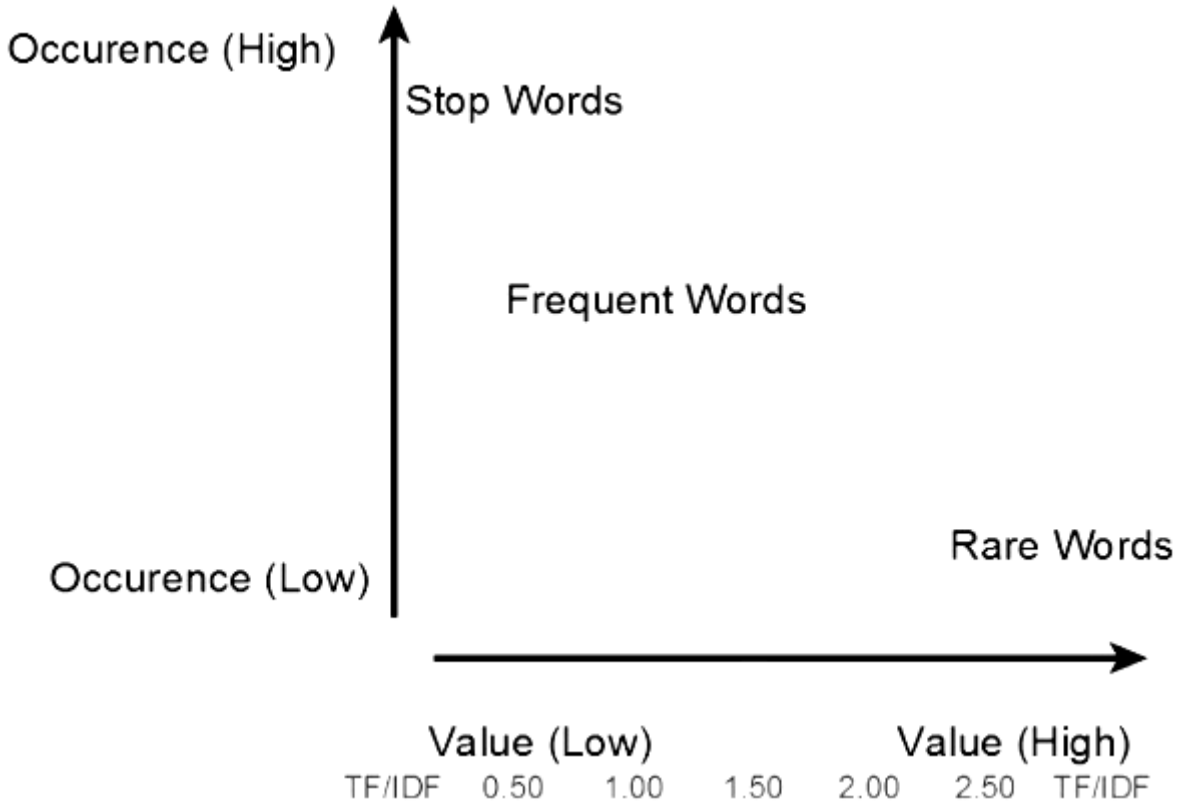
TF = (Number of times term T appears in the particular row) / (number of terms in that row)

The intuition behind inverse document frequency (IDF) is that a word is not of much use to us if it's appearing in all the documents. Therefore, the IDF of each word is the log of the ratio of the total number of rows to the number of rows in which that word is present.

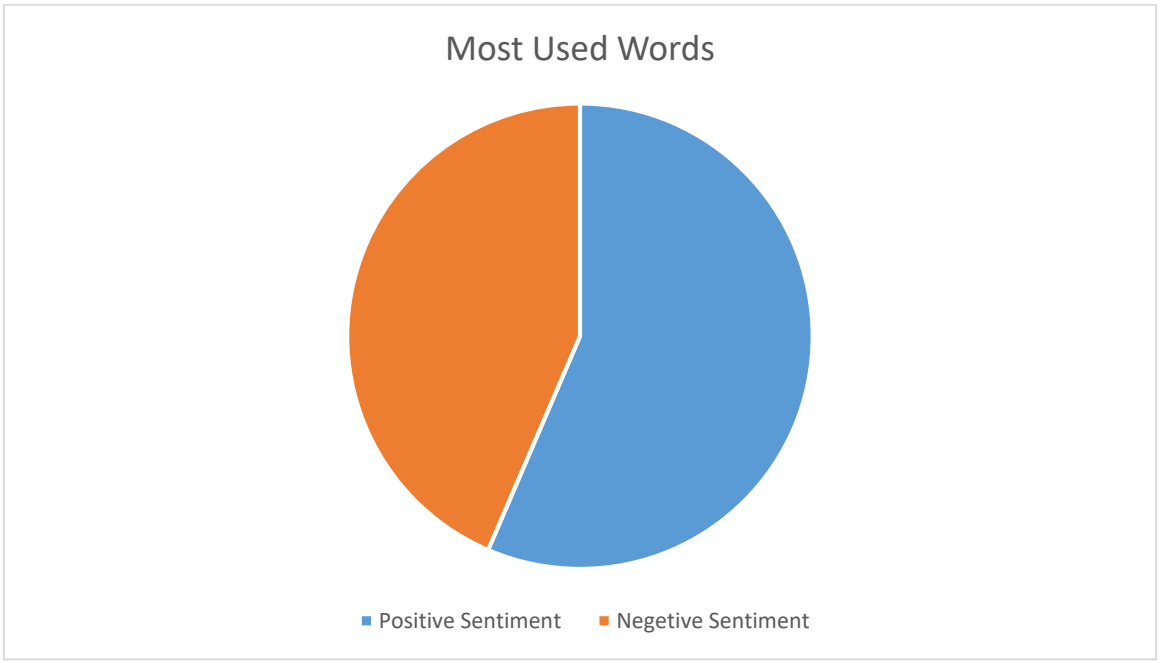
IDF = log (N/n), where, N is the total number of rows and n is the number of rows in which the word was present.

TF-IDF is the multiplication of the TF and IDF which we calculated above.

TF-IDF has penalized words like 'don't', 'can't', and 'use' because they are commonly occurring words. However, it has given a high weight to "disappointed" since that will be very useful in determining the sentiment of the tweet.



Most Used Words



See the Word ‘lol’ it is used in both positive and negative (sarcastic) sentiments.

Splitting dataset

Spitting the dataset into train and test data for cross validation. We have split the dataset into 8:2 ratio of (train : test) data randomly.

X_train_shape : (79991, 1195309) X_test_shape : (19998, 1195309)

y_train_shape : (79991,) y_test_shape : (19998,)

Naive Bayes Classifier

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes Theorem for events *A* and *B*, provided that $P(B) \neq 0$,

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

- P (A|B) is the posterior probability of class (c, target) given predictor (x, attributes).
- P (A) is the prior probability of class.
- P (B|A) is the likelihood which is the probability of predictor given class.
- P (B) is the prior probability of predictor.

Pros:

- It is easy and fast to predict class of test data set. It also perform well in multi class prediction
- When assumption of independence holds, a Naive Bayes classifier performs better compare to other models like logistic regression and you need less training data.
- It perform well in case of categorical input variables compared to numerical variable(s). For numerical variable, normal distribution is assumed (bell curve, which is a strong assumption).

Cons:

- If categorical variable has a category (in test data set), which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as "Zero Frequency". To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
- On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict_proba are not to be taken too seriously.
- Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.

Naive Bayes models:

- Gaussian: It is used in classification and it assumes that features follow a normal distribution.
- Multinomial: It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".

- Bernoulli: The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

Here we will be using Multinomial Classification.

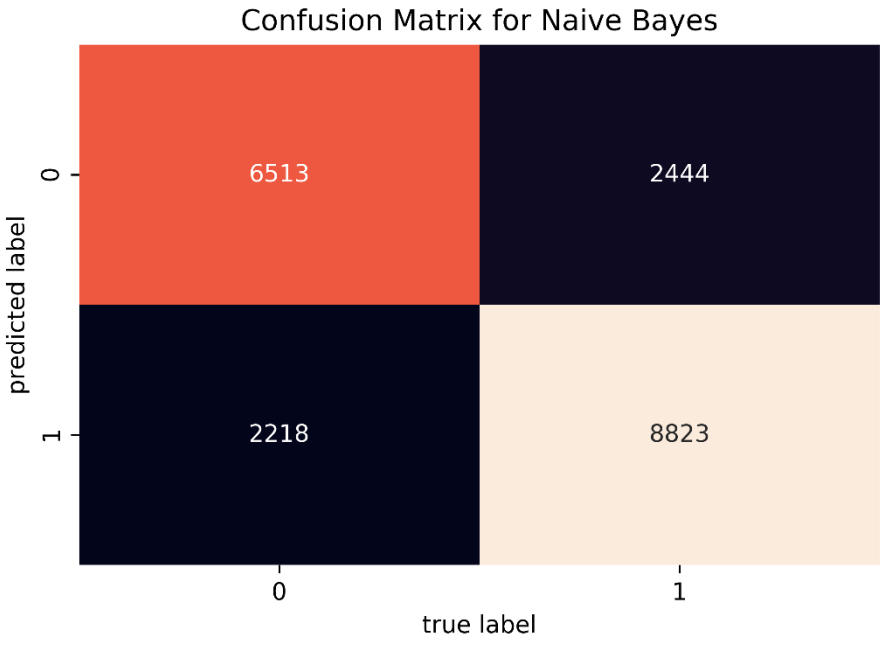
Precision Recall and Accuracy

Precision, recall, and accuracy are standard metrics used to evaluate the performance of a classifier.

- Precision measures how many texts were predicted correctly as belonging to a given category out of all of the texts that were predicted (correctly and incorrectly) as belonging to the category.
- Recall measures how many texts were predicted correctly as belonging to a given category out of all the texts that should have been predicted as belonging to the category. We also know that the more data we feed our classifiers with, the better recall will be.
- Accuracy measures how many texts were predicted correctly (both as belonging to a category and not belonging to the category) out of all of the texts in the corpus.

Most frequently, precision and recall are used to measure performance since accuracy alone does not say much about how good or bad a classifier is.

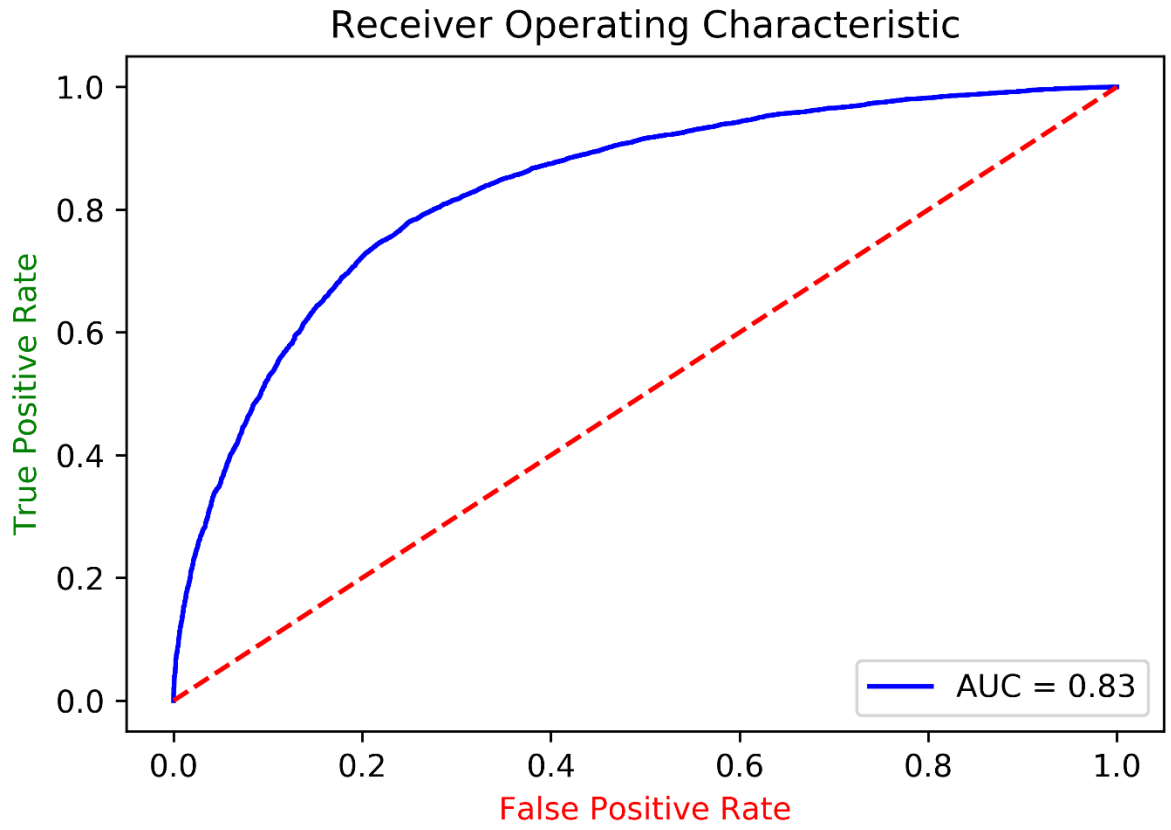
Confusion Matrix



Classification Report

	Precision	Recall	F1-score	Support
0	0.73	0.75	0.74	8731
1	0.80	0.78	0.79	11267
Accuracy			0.77	19998
Micro avg	0.76	0.76	0.76	19998
Weighted avg	0.77	0.77	0.77	19998

ROC Curve



Area Under the Curve of anything greater than 0.80 with real life data is not bad. We achieved 0.83.

Comparison (unigram only)

Normalization	Stop Words	Vectorizer	Accuracy
None	No	Count	0.7601260126012601
Lemmatizing	No	Count	0.757025702570257
Stemming	No	Count	0.755025502550255
Lemmatizing	Yes	Count	0.7483748374837483
None	No	Tf-Idf	0.7480748074807481
Stemming	Yes	Count	0.7477747774777478
Lemmatizing	Yes	Tf-Idf	0.7456245624562456
Stemming	No	Tf-Idf	0.7435743574357436
Lemmatizing	No	Tf-Idf	0.7403740374037404
Stemming	Yes	Tf-Idf	0.7383738373837384

So here we see the best combination for sentiment analysis with this dataset is using

- Without normalization
- Without stop words
- Count Vectorizer

For document analysis the best combination will include lemmatization, stop words, but in sentiment analysis, they aren't providing edge over the raw datas.

Accuracy (N-grams)

N-gram Methods	Accuracy
Unigram, Bigram and Trigram	0.76688
Unigram and Bigram	0.76788
Bigram and Trigram	0.72532

Conclusion

Achievements

The provided tweets were a mixture of words, emoticons, URLs, hashtags, user mentions, and symbols. Before training the dataset we pre-processed the tweets to make it suitable for feeding into models. We implemented machine learning algorithms Naive Bayes Classifier to classify the polarity of the tweet. We used three types of features namely unigrams, bigrams and trigrams for classification and observed that augmenting the feature vector with bigrams improved the accuracy and trigrams degraded the accuracy. Once the feature has been extracted it was represented as a sparse vector. It has been observed that presence in the sparse vector representation recorded a better performance than frequency. We finally achieved an accuracy of 76.78%

With unigram and bigram, without any normalization, stop words or tf-idf

Our model accuracy : 0.76788

Leaderboard highest : 0.79249

Future Directions

- Handling emotion ranges: We can improve and train our models to handle a range of sentiments. Tweets don't always have positive or negative sentiment. At times they may have no sentiment i.e. neutral. Sentiment can also have gradations like the sentence, 'This is good', is positive but the sentence, 'This is extraordinary', is somewhat more positive than the first. We can therefore classify the sentiment in ranges, say from -2 to +2.
- Using hashtags and other symbols like '?' and '!': During our pre-processing, we discard most of the symbols like hashtags, question marks and exclamation marks. These symbols may be helpful in assigning sentiment to a sentence.

Acknowledgements

- Towards Data Science
- Analytics Vidya
- Project report on Github
- Medium
- Stackabuse
- Geeksforgeeks
- Kaggle
- Stack Overflow
- Stack Exchange
- Monkey Learn

Full project on GitHub