Sentiment Analysis of tweets

Project By,

Team Name: Project Code

Members:

Varshini P.J

Dharineesh Karthikeyan

Abstract:

This project addresses the problem of sentiment analysis on twitter data. The goal of our project is to build a sentiment analysis model on the given data. Our main area of focus is on the Classical Machine Learning methods.

The dataset given was not about any specific topic, and was quite random. We performed cleaning on the given twitter data. Analyzed the given dataset and also the cleaned tweets. We studied 3 different word Embeddings- Bag-Of-Words, Tf-Idf vectorization, and Word2vec. Compared the results of these word embeddings for various models and then selected the best performing model. Performed hyper parameter tuning and Grid Search on the model, to improve its performance. Extracted additional features from the twitter data, which might help the model perform better. Performed Feature Selection on the additional features, and chose the most useful features and made a model. Our final model got a Kaggle score of 0.71051.

Keywords: sentiment analysis, Classical Machine Learning methods, Bag-of-words, Tf-idf, Word2vec, Hyper parameter tuning, Grid Search, Feature Selection

INDEX

1) Introduction

1.1) Problem Statement:

The aim of the project is to perform sentiment analysis to predict the sentiments of the given tweets using Classical Machine Learning methods. In this case, the sentiments are: positive, neutral and negative.

1.2) Datasets and Input:

Given two datasets:

1. train.txt - 21,465 entries.
2. test_samples.txt - 5,398 entries.

The datasets contain the following attributes:

Train dataset contains:

- Tweet_id
- Sentiment
- Tweet_text

Test dataset contains:

- Tweet_id
- Tweet_text

Here the "sentiment" is the target class. The target class classifies the user tweet as positive or neutral, or negative.

The first 5 entries of the train dataset look like:

	tweet_id	sentiment	tweet_text
0	264183816548130816	positive	Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :)
1	263405084770172928	negative	Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday.
2	262163168678248449	negative	its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february.
3	264249301910310912	negative	Iranian general says Israel\u2019s Iron Dome can\u2019t deal with their missiles (keep talking like that and we may end up finding out)
4	262682041215234048	neutral	Tehran\u002c Mon Amour: Obama Tried to Establish Ties with the Mullahs http://t.co/TZZzrrKa via @PJMedia_com No Barack Obama - Vote Mitt Romney

The first 5 entries of the test dataset look like:

	tweet_id	tweet_text
0	264238274963451904	@jjueellzz down in the Atlantic city, ventnor, margate, ocean city area. I'm just waiting for the coordinator to hopefully call me tomorrow
1	218775148495515649	Musical awareness: Great Big Beautiful Tomorrow has an ending, Now is the time does not
2	258965201766998017	On Radio786 100.4fm 7:10 Fri Oct 19 Labour analyst Shawn Hattingh: Cosatu's role in the context of unrest in the mining http://t.co/46pjzzl6
3	262926411352903682	Kapan sih lo ngebuktiin,jan ngomong doang Susah Susah.usaha Aja blm udh nyerah,inget.if you never try you'll never know.cowok kok gentle bgt
4	171874368908050432	Excuse the connectivity of this live stream, from Baba Amr, so many activists using only one Sat Modem. LIVE http://t.co/U283IhZ5 #Homs

2) Preprocessing:

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

The basic steps followed in Text Preprocessing are:

1. Cleaning
2. Text processing
3. Data Analysis
4. Word Embeddings

2.1) Cleaning:

2.1.1) Step 1:

In the first step, we had a close look at the given data and we could observe the existence of terms such as “ \u2019” , “\u002c” , “&”, ‘ \"" ‘ .

After analyzing the data properly, we found out the following:

- “\u2019” is used in place of '
- “\u002c” is used in place of ,
- “&” is used in place of &
- ‘ \"" ‘ is used in place of “

So, the wrong terms were replaced with the correct terms in its place.

2.1.2) Step 2:

Now we remove the parts of the tweet that don't contribute much to the sentiment of the tweet. So, we remove the following from the given tweets:

- **URL**
  - URL are links that usually follow the patterns of http:// or https:// or www.
- **Retweets**
  - Retweets are mentioning that the tweet is forwarded from someone else. It is denoted with RT. Usually followed with a user name, but we will remove it later on.
- **Users**
  - Users are people's twitter ID's. They are random and do not contribute much to the sentiment. Hence, we remove them. Users usually start with the '@' symbol.
- **Repeated Letters**
  - For emphasize purposes, we sometimes tend to prolong specific letters to make it more dramatic. For example, 'nooooo' in place of 'no'. In such cases, we try to limit the number of additional letters to a maximum of 2. Hence, in this case, it will be replaced with 'noo'.
- **Numbers and Digits**
  - Since, numbers and digits do not contribute much to the sentiment. We can remove them from the tweet.
- **Punctuations**
  - Similar to numbers and digits, punctuations do not contribute much to the sentiment and it can be a hindrance while processing the text. Hence, we can remove them from the tweet as well.

2.1.3) Step 3:

Now, we focus on replacing certain important parts of the tweet to a more suitable format for the purpose of data processing.

- **Emoticons**

  Emoticons or emojis play a critical role in determining the sentiment of the tweet (if present). In the given data, we have emojis represented as symbols and not as Unicode's.

  We created a dictionary of basic emojis mapped to their respective emotions. During cleaning process, we replaced the emojis with emotions. This has to be done before removal of punctuations.

```
emojis = {':-)':'happy',':)':'happy',':D':'happy',':o)':'happy',':]':'happy',':3':'happy',':c)':'happy', ':>':'happy',
          '=]':'happy','8)':'happy',';)':'happy',';-)':'happy',':d':'happy',';d':'happy','xx':'happy','.xx':'happy',
          '<33':'happy','<3':'happy','>.<':'happy','|;-)':'happy',

          '>:[':'sad',':-(':'sad',':(':'sad',':-c':'sad',':c':'sad',':-<':'sad',':<':'sad', ':-[':'sad',':[':'sad',
          ':{':'sad',':’-(':'sad',':’(':'sad','>:':'sad','>:/':'sad',':-/':'sad',':-.':'sad',':- :':'sad','=/=':'sad',
          ':L':'sad','=L':'sad',':S':'sad',

          ':-||':'angry',':@':'angry','>:(':'angry',

         'D:<':'shocked','D:':'shocked','D8':'shocked','D;':'shocked','D=':'shocked','DX':'shocked','v.v':'shocked',
          'D-’':'shocked',';o':'shocked',';O':'shocked','XDD':'shocked','XD':'shocked','>:O':'shocked',':-O':'shocked',
          ':O':'shocked',':-o':'shocked',':o':'shocked','8O':'shocked','O_O':'shocked','o-o':'shocked','O_o':'shocked',
          'o_O':'shocked','o_o':'shocked','O-O':'shocked',

          ':|':'neutral',':-|':'netural',

         }
```

- **Contradictions and Abbreviations**

  Words that could be split up into other words, and also some of the most common Abbreviations have also been replaced with their full forms.

  We created a dictionary with some of the common contradictions and some commonly found abbreviations mapped to their full forms.

```
replace = {"dont":"do not","don't":"do not",
               "aren't":"are not","arent":"are not","cant":"can not","can't":"can not",
               "couldn't":"could not",'couldn':"could not","couldnt":"could not",
               'didn':"did not","didn't":"did not","didnt":"did not",
               'doesn':"does not","doesn't":"does not","doesnt":"does not",
               'hadn':"had not","hadn't":"had not","hadnt":"had not",
               'hasn':"has not", "hasn't":"has not", 'haven':"have not",
               "haven't":"have not","haven":"have not","havent":"have not",
               'isn':"is not", "isn't":"is not","isnt":"is not",
               'mightn':"might not","mightn't":"might not","mightnt":"might not",
               'mustn':"must not", "mustn't":"must not","mustnt":"must not",
               'needn':"need not","needn't":"need not","neednt":"need not",
               'shan':"shall not", "shan't":"shall not","shant":"shall not",
               'shouldn':"should not", "shouldn't":"should not","shouldnt":"should not",
               'wasn':"was not","wasn't":"was not","wasnt":"was not",
               'weren':"were not", "weren't":"were not","werent":"were not",
               'won':'will not', "won't":"will not","wont":"will not",
               'wouldn':"would not","wouldn't":"would not","wouldnt":"would not",
               "aint":"am not","ain't":"am not",'im':'i am',"i'm":'i am',
               'yall':'you all',"youve":'you have',"you've":'you have',"i'll":'i will','ill':'i will','u':'you',
               'amp':'is not my problem','smh':'shaking my head','lol':'laughing out loud',
               'jan':'january','feb':'february','mar':'march','apr':'april','jun':'june','aug':'august',
               'nov':'november','sept':'september','oct':'october','dec':'december',
               'mon':'monday','tue':'tuesday','tues':'tuesday','wed':'wednesday','thur':'thursday','fri':'friday',
               '@':'at','tis':'this','ohlawd':'oh lord','yas':'yes','yass':'yes','hmm':'',
               'tho':'though','thov':'though','btw':'by the way',"cmon":'come on',"c'mon":'come on','cuz':'cause',
               'diff':'difference','sayin':'saying','drivin':'driving',
               'lmfao':'laughing my fucking ass off','lmao':'laughing my ass off','rofl':'rolling on the floor laughing',
               'lmaoo':'laughing my ass off'}
```

- **Hashtags**

We did not remove the hashtags, as sometimes it contains information which might contribute towards the sentiment. In most cases, hashtags are a bunch of words mixed together, for example “#iamhappynow” , which clearly needs to be split up as “I am happy now”. To perform this task, we used a python package called ‘wordninja’ which helped us perform this task.

Since spelling mistakes and combined words are a common issue in social media, we performed this task on every word in the tweet. To avoid, smaller words from getting split such as “ohlawd” which got split as “ohl awd”. To prevent this, we kept the length of the word at a cap of 6 letter to prevent this from happening.

```
Original Tweet:
My Saturday night has consisted of me watching The Grey with my puppy while my parents throw a rager #liamneesonisbosstho
--------------------------------------------------------------------------------------------------------------------------

Split Tweet:
My Saturday night has consisted of me watching The Grey with my puppy while my parents throw a rager liam neeson is boss tho
```

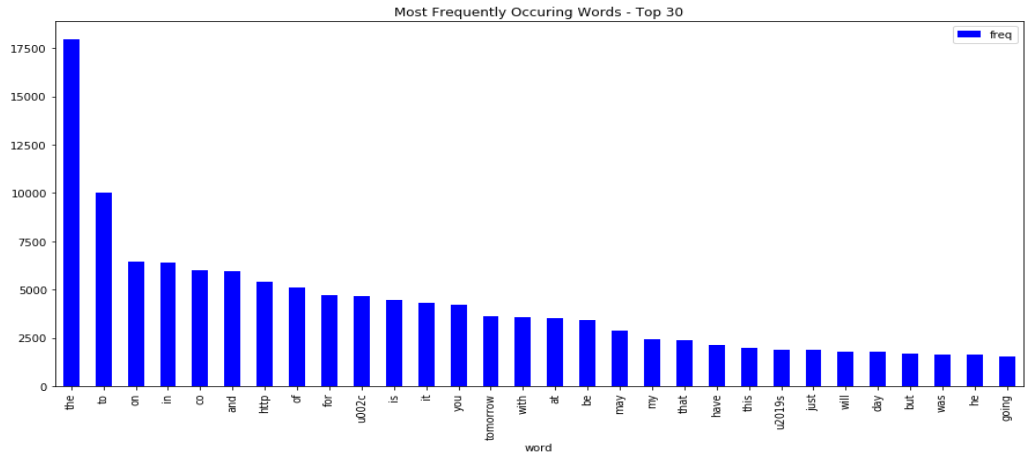
2.2) Text Processing:

2.2.1) Stopwords Removal:

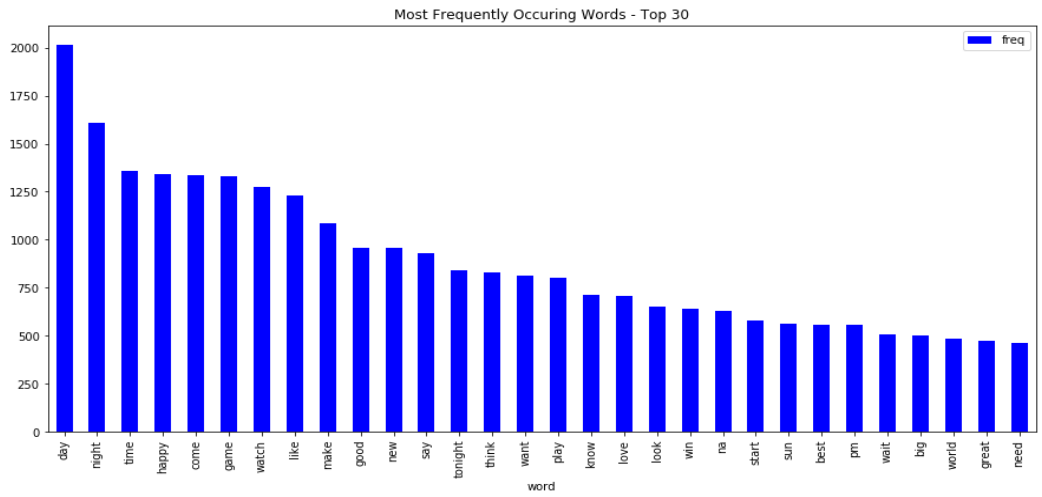
A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that has been programmed to ignored in an NLP task, since their contribution to the task is limited.

We would not want these words to take up space in our database, or taking up valuable processing time. Hence, it’s a common practice to remove these stop words during NLP tasks.Let's have a look at the frequency chart of these tweets before and after stopwords removal

Before removing stopwords:

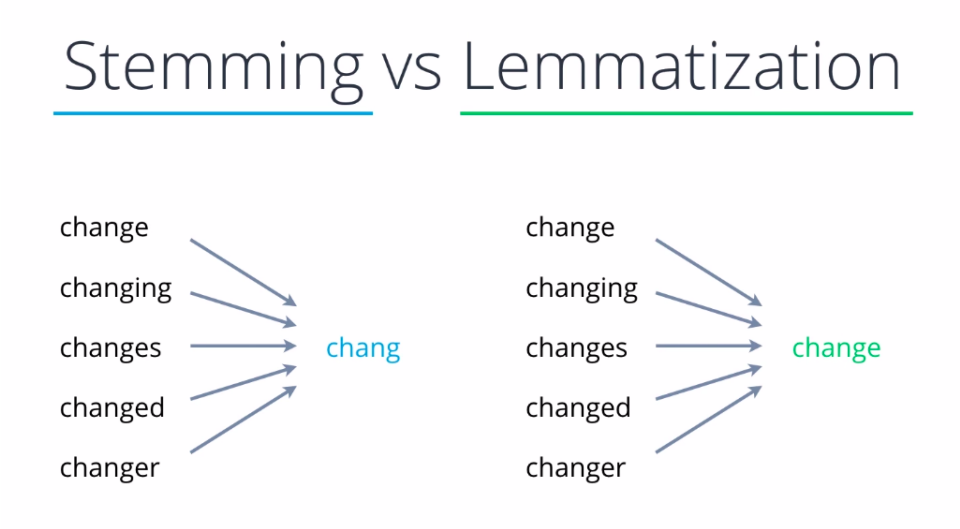


After removing stopwords:



Additionally, to the common English stopwords provided by NLTK package, we also added a few additional words such as the months and dates, and the words ‘tomorrow’ and ‘today’ as well. We also removed the words ‘no’ and ‘not’ from the existing stopwords list, as we felt that those words were important for the sentiment of the text

2.2.2) Stemming and Lemmatization:



As we can see from the given diagram, stemming gives the root words (sometimes the root word is not English). Whereas, lemmatization makes sure that the root word we get is an English word. Already, the data contains a lot of spelling mistakes, so lemmatization stands as the best choice.

Stemming doesn’t depend on the context in which the word is used, whereas lemmatization requires the context in order to lemmatize a given word. It can be demonstrated as an example, given below:

```
print(lemmatizer.lemmatize("loving"))
print(lemmatizer.lemmatize("loving", "v")) #Here v specifies that it is a verb
```

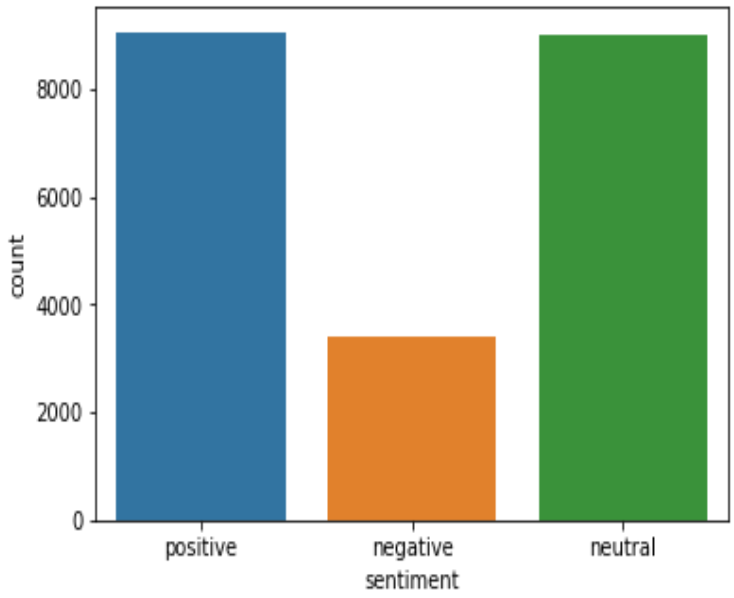
```
loving
love
```

Hence, in order for lemmatization to perform well, we need to pass the context along with the word as well. Therefore, we use Parts of speech tagging to find out the context of the word in the given tweet and pass it on along with the word for lemmatization to work on. We defined a function to perform this task for us.

2.3) Data Analysis

2.3.1) Data Distribution:

The below figure shows how the target class is distributed in the given training dataset

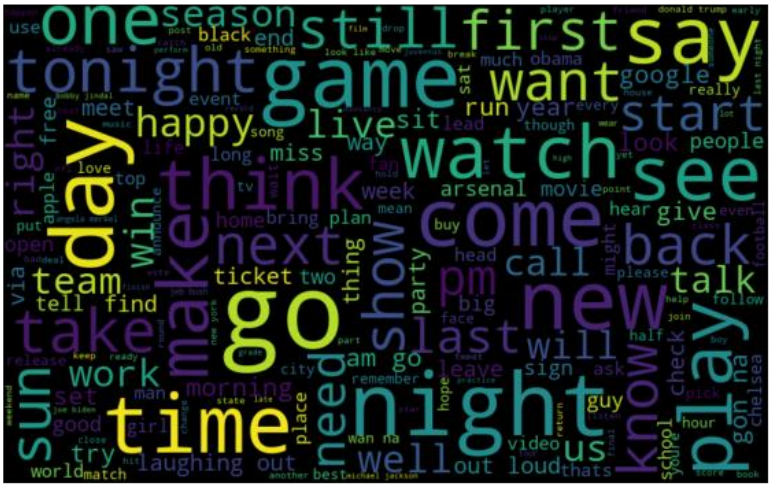


From the above diagram, it is clearly visible that negative samples are very less compared to the positive and neutral samples. Hence, we are dealing with an imbalanced class dataset. We need to try up sampling and down sampling methods to deal with this dataset.

2.3.2) Word Cloud

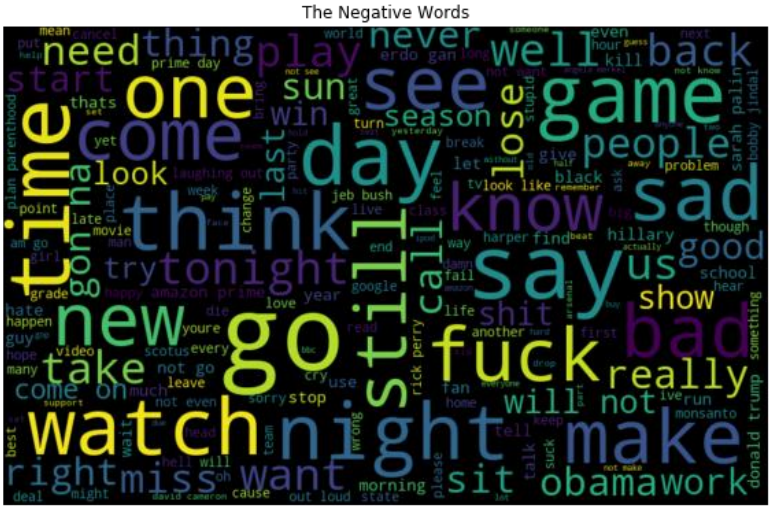
Verbal Representation of the words in the given tweets specific to their sentiment. Word Cloud helps us get a clearer picture of the words and their frequency in the given data.

The Neutral Words



The Positive Words

The Negative Words



2.4) Word Embeddings:

Word embeddings can simply be explained as the process of converting text into a more suitable format for the machine to work on. Machine learning algorithms cannot work with raw text directly. Hence, the text must be converted into numbers. Specifically, vectors of numbers.

Some of the commonly used Word Embedding's are the following:

- **Bag-Of-Words (BOW)**

  A bag-of-words is a representation of text that describes the occurrence of words within a document. It involves two things:

  i. A vocabulary of known words.
  ii. A measure of the presence of known words.

  BOW is built by simple count of occurrences of every unique word across all of the training dataset.

- **Tf-idf Vectorization**

  Term Frequency – Inverse Document Frequency, or TF-IDF for short, where:

  i. Term Frequency: is a scoring of the frequency of the word in the current document.
  ii. Inverse Document Frequency: is a scoring of how rare the word is across documents.

- **Word-2-Vector**

Word2vec is a two-layer neural net that processes text by "vectorizing" words. Its input is a text corpus and its output are a set of vectors: feature vectors that represent words in that corpus. While Word2vec is not a deep neural network, it turns text into a numerical form that deep neural networks can understand.

Word2vec helps to form connections between similar words which helps the model understand similarities between words much easier.

3) Model Selection:

Model selection is the process of experimenting different types of word embeddings and classification models for our data, and selecting the most promising model and word embeddings to work with.

This experimentation was done on the training data, by splitting it into a 75% training set and a 25% testing set. All the below shown results are based on this data.

Table Terminologies:

- LR – Logistic Regression
- SVM – Support Vector Machine Classification
- DT – Decision Tree Classification
- RF – Random Forest
- BNB – Bernoulli Naïve Bayes
- MNB – Multinomial Naïve Bayes
- SGD – Stochastic Gradient Descent Classification
- XGB – XG Boost

Experimentation Results:

- **Bag-Of-Words (BOW)**

Model	**Negative (F1-Score)**	**Neutral (F1-Score)**	**Positive (F1-Score)**	**Mean (F1-Score)**
LR	0.45340502	0.66204773	0.70528967	0.60691414
SVM	0.4730832	0.65224192	0.7014756	0.60893357
DT	0.32180209	0.62405554	0.43172527	0.45919425
RF	0.32	0.67305389	0.69438906	0.56248098
BNB	0.	0.62094434	0.68157511	0.43417315
MNB	0.367428	0.6105919	0.6916996	0.55657317
SGD	0.4461671	0.65331599	0.70167598	0.60038636
XGB	0.44026341	0.68325041	0.69351908	0.60567763

- **Tf-Idf Vectorization:**

Model	**Negative (F1-Score)**	**Neutral (F1-Score)**	**Positive (F1-Score)**	**Mean (F1-Score)**
LR	0.32079646	0.67004097	0.70648174	0.56577306
SVM	0.39324727	0.66618182	0.69484655	0.58475855
DT	0.32045089	0.62311762	0.42304527	0.45553793
RF	0.31116122	0.66951567	0.68769716	0.55612468
BNB	0.	0.62094434	0.68157511	0.43417315
MNB	0.	0.61347038	0.68344811	0.43230616
SGD	0.35791757	0.67081936	0.69226361	0.57366685
XGB	0.41570881	0.67750294	0.68789809	0.59370328

- **Word2Vec:**

Model	**Negative (F1-Score)**	**Neutral (F1-Score)**	**Positive (F1-Score)**	**Mean (F1-Score)**
LR	0.40687161	0.60505529	0.64258555	0.55150415
SVM	0.35568513	0.6086736	0.63586054	0.53340642
DT	0.22494888	0.52233316	0.54076802	0.42935002
RF	0.12516644	0.60774578	0.63829787	0.45707003
BNB	0.41035857	0.54828481	0.5959568	0.51820006
MNB	--	--	--	--
SGD	0.20754717	0.60731645	0.63985375	0.48490579
XGB	0.37981651	0.60294892	0.64250946	0.5417583

Note:

Word2Vec word embeddings couldn't be fitted into a Multinomial NB model. Since, multinomial NB doesn't accept negative values in its inputs, but Word2Vec model contains negative values in its input.

Now as we can see from our tabulated performance table, Logistic Regression Model seems to be performing quite well on our given data for both Bag of words model and also Tf-Idf model. As we can see, the model performs badly for 'negative' sentiment. From the analysis we performed on the data before, we can clearly see that this is a result of lack of 'negative' samples in our dataset. Hence, we need to perform Up Sampling on the data and compare how the Logistic Regression model performs.

After performing Up Sampling, on the Bag-of-words Tf-Idf vectorized and Word2vec Logistic Regression models. The results are as follows:

Logistic Regression				
	Negative (F1-Score)	**Neutral (F1-Score)**	**Positive (F1-Score)**	**Mean (F1-Score)**
Bag-Of-Words	0.44414536	0.61385042	0.69856734	0.58552104
TF-IDF	0.49888971	0.66510782	0.6987596	0.62091904
Word2Vec	0.47065217	0.54589372	0.62609202	0.54754597

From the above tabulation, we can see that Tf-Idf Vectorized Logistic Regression Model performs better than the Bag-Of-Words Logisitic Regression Model on a Up Sampled Dataset.

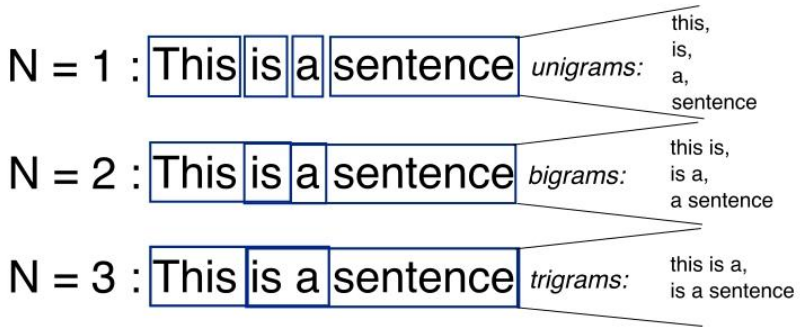
Hence, from this we select the Word Embeddings as Tf-Idf and the model as Logistic Regression Model to proceed with.

4) Hyper Parameter Tuning:

Going forward with the TF-IDF vectorized Logistic Model, we have to tune the hyper parameters to improve the performance.

- **N-grams: (TF-IDF Vectorizer parameter)**

  An n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application
  The n-grams typically are collected from a text or speech corpus.



So, experimenting with various n-gram features, the results are as follows:

N-grams	**Negative (F1-Score)**	**Neutral (F1-score)**	**Positive (F1-Score)**	**Mean (F1-Score)**
Uni-grams (1,1)	0.50053022	0.63028709	0.68051342	0.60377691
Uni and Bi grams (1,2)	0.52865256	0.62121562	0.69888143	0.61624987
Bi-grams (2,2)	0.39251208	0.56882291	0.59912904	0.60807228
Uni and Tri-grams (1,3)	0.51711804	0.60979087	0.69730794	0.60807228
Tri-grams (3,3)	0.20499109	0.59972106	0.35834125	0.38768447

From this we can see that that choosing uni and bi- grams (1,2) as the n_gram range for our TF-IDF vectorizer is a good choice.

- **Min_df (TF-IDF Vectorizer parameter)**

  This parameter helps to decide, the minimum frequency of a word in the final Tf-Idf vectorization.

Min_df	**Negative (F1-Score)**	**Neutral (F1-score)**	**Positive (F1-Score)**	**Mean (F1-Score)**
1 (Default value)	0.51948052	0.61922366	0.69805847	0.61225422
2	0.51527924	0.62901786	0.69559229	0.61329646
3	0.52378463	0.63284379	0.69100994	0.61587945
4	0.52269171	0.63111111	0.68936762	0.61439015
5	0.51422659	0.62933333	0.685887	0.60981564

- **Max_df (TF-IDF Vectorizer parameter)**

This parameter helps to decide, the probability of the maximum frequency of a word in the final Tf-Idf vectorization.

Max_df	**Negative (F1-Score)**	**Neutral (F1-score)**	**Positive (F1-Score)**	**Mean (F1-Score)**
1 (Default value)	0.52097359	0.6185853	0.69912888	0.61289592
0.95	0.52277433	0.61819021	0.69709172	0.61268541
0.90	0.51944012	0.61847575	0.69675978	0.61155855
0.85	0.52146922	0.61925652	0.69619687	0.61230753
0.80	0.52178423	0.62072155	0.69701026	0.61317201

From the above tabulation results, we can see that there's only a marginal difference between the f1-scores. Hence, we decide to stick with the default values of min_df=1, max_df=1. This is because, doing so will help us increase the vocabulary size and might be helpful for us.

- **Logistic Regression Grid Search**

Our main focus is on the parameter 'C' of Logistic Regression. Hence, we performed Grid search on it and the result is as follows:

```
from sklearn.model_selection import GridSearchCV
from imblearn.pipeline import Pipeline

grid_search_pipeline = Pipeline([
    ('vectorizer', TfidfVectorizer(ngram_range=(1,2))),
    ('sampling', SMOTE()),
    ('model', LogisticRegression(max_iter=1000)),
])

params = [
    {
        'model__C':[0.01,0.1,1,10,100],
    },
]
grid_search = GridSearchCV(grid_search_pipeline, params, cv=5, scoring='f1_weighted')
grid_search.fit(tweets, sentiment)
print(grid_search.best_params_)
```

```
{'model__C': 1}
```

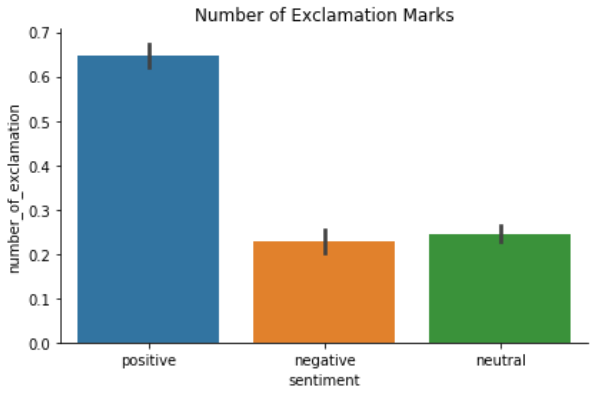
Hence, from this we find out that the default value of C=1 is best suited for our data.

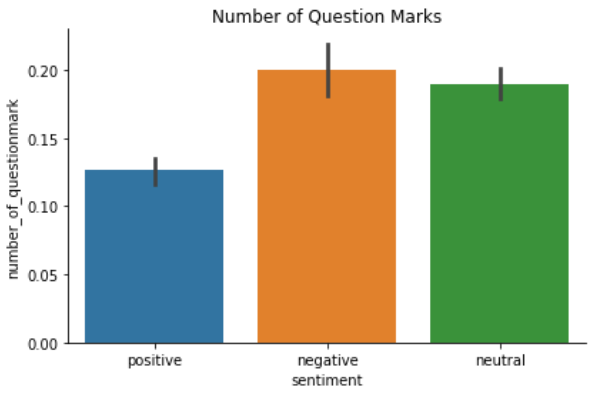
5) Additional Features:

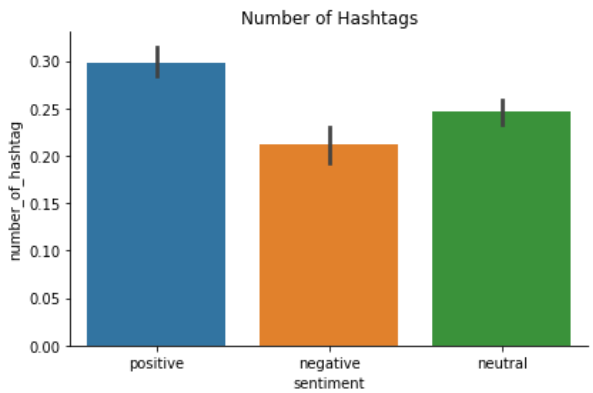
Let's try to add some features that might help the model to classify tweets. Some of the additional features that we choose to try out are the following:

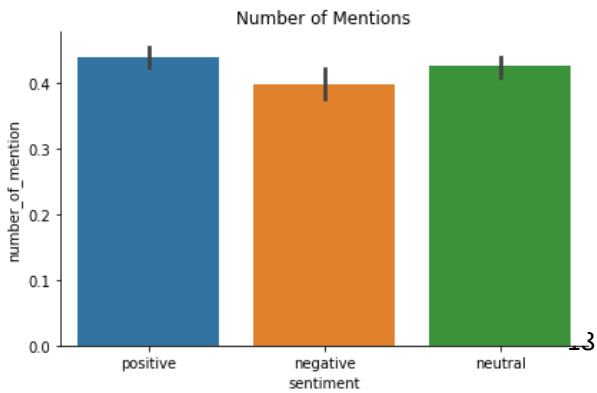
Feature	Reason
Exclamation marks (!)	Exclamation marks are mostly used to increase the strength of the emotion
Question marks (?)	Can be used to identify neutral tweets and questions
Hashtags (#)	Hashtags are mostly used to convey an emotion or action
Mention (@)	Mentioning another user, might mean some message or emotion is being conveyed to others.
Emoticons	Finding the number of positive and negative emoticons in a message, helps understand the sentiment of the tweet
Words	Finding the common positive and negative words used in a tweet, can help understand the sentiment of the tweet

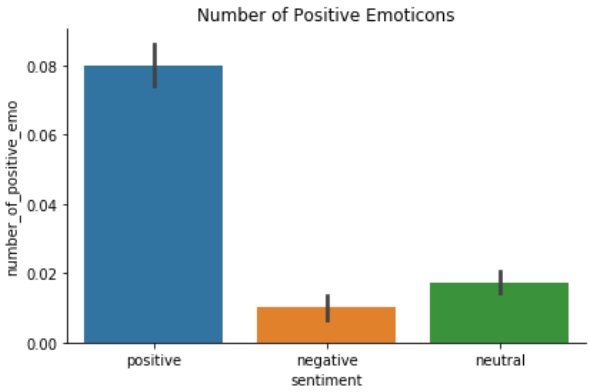
Let’s have a look at how these additional features plays a vital role in the sentiment analysis

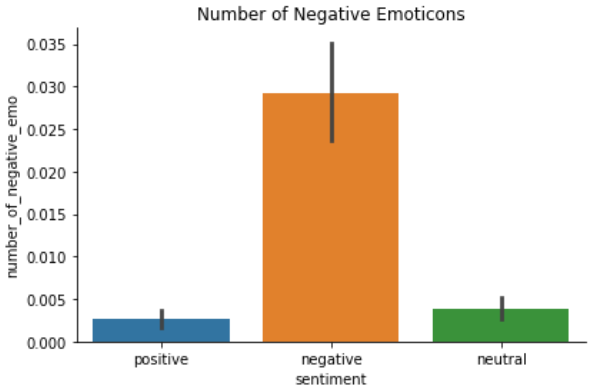


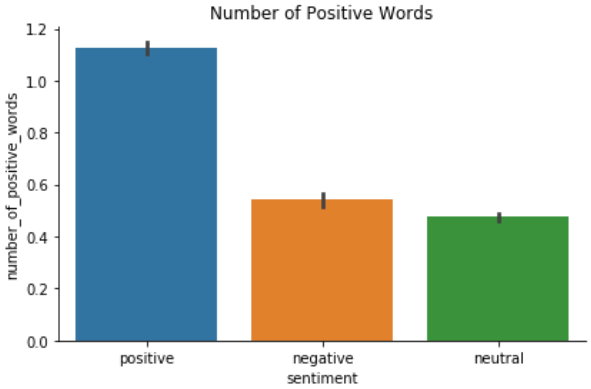


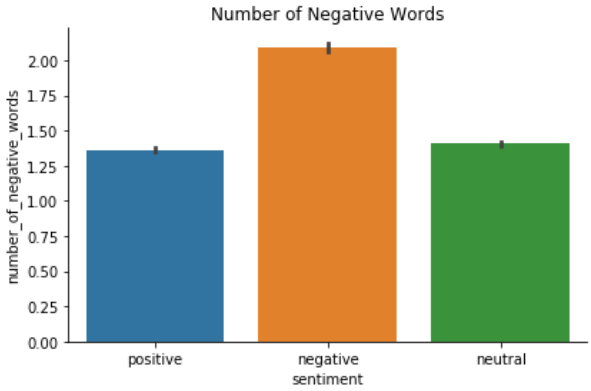












5.1) Features Selection:

Feature Selection is the process of selecting the best possible features for our model from the list of features available.

We tried using all of the features at the beginning, after that limited the number of features we choose based on the above graphs.

Studying from the graph, we can see that the occurrence of certain features is pretty close to each other for every sentiment class. For example, the number of question mark, mentions and hashtags are commonly found in all three sentiment classes.

Whereas, the number of exclamations is prominently found in the positive class. Even, the number of positive emoticons and words are also majorly present in the positive sentiment class. Similarly, for the negative emoticons and words are majorly present in the negative sentiment class.

The final feature selection for our final model, is the following:

- Number of exclamations
- Number of positive and negative emoticons
- Number of positive and negative words

6) Conclusion:

All the cleaning and processing steps explained till now, was for our final model. There were many changes to the data cleaning and processing steps on the way and will be explained below.

The following are the trials made prior to the final model:

- **Trial 1:**

  Data Cleaning:

  - Removal of unwanted phrases like '\u2019' and '\u002c'
  - Removal of users, punctuations, URLs and hashtags and also repeated letters

  Processing:

  - TF-IDF Vectorization

  Model:

  - Logisitic Regression Model

- **Trial 2:**

  Updates to Data Cleaning:

  - Replaced emoji's with emotions

- **Trial 3:**

  Updates to Data Cleaning:

  - Built a bigger emoji dictionary, and replaced emojis with emotions
  - Made changes to stop words dictionary

- Built the contradictions dictionary
- Did not remove hashtags
- Used wordninja to clean the text
- Removed words that were less than 3 characters long

- **Trial 4:**

  Updates to Vectorization:

  - Hyper parameter tuning. (As mentioned in our report)

  Updates to Model:

  - Performed Up sampling

- **Trial 5:**

  Updates to Data Cleaning:

  - Final Data Cleaning (as mentioned in our report)

  Updates to Model:

  - Performed SMOTE Up sampling on minority class

- **Trial 6:**

  Updates to Model:

  - Performed SMOTE Up sampling on all classes

- **Trial 7:**

  Updates to Model:

  - Additional Features was added (As mentioned in our report)

- **Trial 8:**

  Updates to Model:

  - Feature selection was performed on the Additional Features
  - Removed the features ‘number of mentions’

- **Trial 9:**

  Updates to Model:

  - Feature selection was performed on the Additional Feature
  - Removed number of hashtags and question marks as well

- **Trial 10**

  Updates to Vectorization:

  - Added additional parameters of min_df and max_features
    (Explanation of why is below)

Note:

In our final model, we had to make some changes to the hyper paramters of Tf-Idf Vectorization.

The changes were, we added min_df=2, and max_features=40000. This was done, as a preventive measure to reduce the vocab size of the vectorization. Since python had a **'Memory Error'**, while trying to work with the whole vocab size. Since, we didn't have enough computational power to handle this process, we had to make the vocab size smaller.

Model	Kaggle Score (Mean F1-Score)
Trial 1	0.65006
Trial 2	0.65655
Trial 3	0.66049
Trial 4	0.67786
Trial 5	0.69383
Trial 6	0.69522
Trial 7	0.70472
Trial 8	0.70704
Trial 9	0.70958
Trial 10	0.71051

7) Scope for Improvement:

Some of the possible improvements that can be made, that we found out are the following

- **Bigger Dataset**

In an NLP problem, it is important to choose the test and training dataset very carefully. Its best suited if the training and test dataset is from the same source and also based on a similar topic. In our case, the topics were not that similar. Hence, a much bigger training dataset might have helped increase the score

- **Cleaning the Data**

Improvements can be made to the text cleaning process, in such a way that the training and test corpus is more similar in features to each other. Cleaning is a huge part of NLP, since text data can be very noisy and we must pick and select the correct terms and features for the model to perform well

- **Additional Features**

We could try to add more additional features that help analyze the sentiment of the tweet.

- **Deep learning models**

Since, our main focus was on Classical Machine Learning methods, we could not focus on deep learning methods. Deep learning methods are well suited to work on text data. Features such as the LSTM, which are capable of learning long-term dependencies, will help the model perform better. Whereas, classical methods cannot do better beyond an extent.

- **Transfer Learning**

We could use pre-trained models and transfer learning and it would help increase the performance

8) References:

1) Prateek Joshi , (July 30, 2018), *Comprehensive Hands on Guide to Twitter Sentiment Analysis with dataset and code*, https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

2) Sharma Roshan, (Mar 31, 2019), https://github.com/sharmaroshan/Twitter-Sentiment-Analysis/blob/master/Twitter_Sentiment.ipynb

3) Gagandeep Singh (Oct 22, 2019), *Updated Text Preprocessing techniques for Sentiment Analysis*, https://towardsdatascience.com/updated-text-preprocessing-techniques-for-sentiment-analysis-549af7fe412a

4) Usman Malik, *Python for NLP: Tokenization, Stemming, and Lemmatization with SpaCy Library*, https://stackabuse.com/python-for-nlp-tokenization-stemming-and-lemmatization-with-spacy-library/

5) Saket Garodia, (Dec 24, 2019), *Twitter Sentiment Analysis*, https://medium.com/analytics-vidhya/twitter-sentiment-analysis-134553698978

6) Ajay Shewale, (June 29, 2019), https://github.com/ajayshewale/Sentiment-Analysis-of-Text-Data-Tweets-/blob/master/Sentiment%20Analysis%20of%20Text%20Data.ipynb

7) Mohamed Afham , (Sept 25, 2019), *Twitter Sentiment Analysis using NLTK, Python*, https://towardsdatascience.com/twitter-sentiment-analysis-classification-using-nltk-python-fa912578614c