# Sentiment Analysis of Tweets

Sneha Banerjee
Snehabanerjee221@gmail.com

**A Project report submitted in fulfilment of the requirement for Data Science and Machine Learning Internship at IIT Kanpur**

## Abstract

*This project addresses the problem of sentiment analysis in twitter; that is classifying tweets according to the sentiment expressed in them: positive, negative or neutral. Twitter is an online micro-blogging and social-networking platform which allows users to write short status updates of maximum length 140 characters. It is a rapidly expanding service with over 200 million registered users - out of which 100 million are active users and half of them log on twitter on a daily basis - generating nearly 250 million tweets per day . Due to this large amount of usage we hope to achieve a reflection of public sentiment by analyzing the sentiments expressed in the tweets. Analyzing the public sentiment is important for many applications such as firms trying to find out the response of their products in the market, predicting political elections and predicting socioeconomic phenomena like stock exchange. The aim of this project is to develop a functional classifier for accurate and automatic sentiment classification of an unknown tweet stream.*

## 1. Introduction

This project of analyzing sentiments of tweets comes under the domain of "Pattern Classification" and "Data Mining". Both of these terms are very closely related and intertwined, and they can be formally defined as the process of discovering "useful" patterns in large set of data, either automatically (unsupervised) or semi- automatically (supervised). The project would heavily rely on techniques of "Natural Language Processing" in extracting significant patterns and features from the large data set of tweets and on "Machine Learning" techniques for accurately classifying individual unlabeled data samples (tweets) according to whichever pattern model best describes them.

The features that can be used for modeling patterns and classification can be divided into two main groups: formal language based and informal blogging based. Language based features are those that deal with formal linguistics and include prior sentiment polarity of individual words and phrases, and parts of speech tagging of the sentence. Prior sentiment polarity means that some words and phrases have a natural innate tendency for expressing particular and specific sentiments in general. For example the word "excellent" has a strong positive connotation while the word "evil" possesses a strong negative connotation. So whenever a word with positive connotation is used in a sentence, chances are that the entire sentence would be expressing a positive sentiment. Parts of Speech tagging, on the other hand, is a syntactical approach to the problem. It means to automatically identify which part of speech each individual word of a sentence belongs to: noun, pronoun, adverb, adjective, verb, interjection, etc.

**What is Sentiment Analysis?**

Sentiment analysis (also known as opinion mining) is one of the many applications of Natural Language Processing. It is a set of methods and techniques used for extracting subjective information from text or speech, such as opinions or attitudes. In simple terms, it involves classifying a piece of text as positive, negative or neutral.

The objective of this task is to detect hate speech in tweets. For the sake of simplicity, we say a tweet contains hate speech if it has a racist or sexist sentiment associated with it. So, the task is to classify racist or sexist tweets from other tweets.

Formally, given a training sample of tweets and labels, where label '1' denotes the tweet is negative and label '0' denotes the tweet is positive/neutral, our objective is to predict the labels on the given test dataset.

```
# Let's load the libraries

import re     # for regular expressions
import nltk   # for text manipulation
import string
import warnings
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore", category=DeprecationWarning)

%matplotlib inline
```

```
In [229]:
```

```
# Let's read train and test datasets.

train = pd.read_csv('../input/sentiment-analysis-project/train.txt',delimiter=',')
test = pd.read_csv('../input/sentiment-analysis-project/test_samples.txt',delimiter=',')
train = train.replace({
    'neutral':0,
    'negative':1,
    'positive':0


})
train['label']=train['sentiment']
train = train.drop(columns=['sentiment'])
train = train[['tweet_id','label','tweet_text']]
print(train.head(5))
```

```
             tweet_id                                                              ...
tweet_text
0  264183816548130816                                                              ...
Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :)
1  263405084770172928                                                              ...
Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday.
2  262163168678248449                                                              ...
its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february.
3  264249301910310912                                                              ...
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                                                              ...
Tehran\u002c Mon Amour: Obama Tried to Establish Ties with the Mullahs http://t.co/TZZzrrKa via @P
JMedia_com No Barack Obama - Vote Mitt Romney

[5 rows x 3 columns]
```

In [230]:

```
train[train['label'] == 0].head(10)
```

Out[230]:

| | tweet_id | label | tweet_text |
|---|---|---|---|
| **0** | 264183816548130816 | 0 | Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :) |
| **4** | 262682041215234048 | 0 | 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 |
| **5** | 264229576773861376 | 0 | I sat through this whole movie just for Harry and Ron at christmas. ohlawd |
| **6** | 264105751826538497 | 0 | with J Davlar 11th. Main rivals are team Poland. Hopefully we an make it a successful end to a tough week of training tomorrow. |
| **8** | 212392538055778304 | 0 | Why is \""Happy Valentines Day\"" trending? It\u2019s on the 14th of February not 12th of June smh.. |
| **10** | 264169034155696130 | 0 | Im bringing the monster load of candy tomorrow\u002c I just hope it doesn\u2019t get all squiched |
| **11** | 263192091700654080 | 0 | Apple software\u002c retail chiefs out in overhaul: SAN FRANCISCO Apple Inc CEO Tim Cook on Monday replaced the heads... http://t.co/X49ZEOsG |
| **12** | 263398998675693568 | 0 | @oluoch @victor_otti @kunjand I just watched it! Sridevi\u2019s comeback.... U remember her from the 90s?? Sun mornings on NTA ;) |
| **13** | 263650552167157762 | 0 | One of my best 8th graders Kory was excited after his touchdown today!! He did the victor cruz!!lol http://t.co/tqORFrXB |
| **14** | 260200142420992000 | 0 | #Livewire Nadal confirmed for Mexican Open in February: Rafael Nadal is set to play at the Me... http://t.co/zgUXpcnC #LiveWireAthletics |

Text is a highly unstructured form of data, various types of noise are present in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as text preprocessing. We will divide it into 2 parts:

- Data Inspection
- Data Cleaning

**Data Inspection**

Let’s check out a few **non** racist/sexist tweets.

In [231]:

```
train[train['label'] == 0].head(10)
```

Out[231]:

| | tweet_id | label | tweet_text |
|---|---|---|---|

| | tweet_id | label | tweet_text |
|---|---|---|---|
| **0** | 264183816548130816 | 0 | Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :) |
| **4** | 262682041215234048 | 0 | 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 |
| **5** | 264229576773861376 | 0 | I sat through this whole movie just for Harry and Ron at christmas. ohlawd |
| **6** | 264105751826538497 | 0 | with J Davlar 11th. Main rivals are team Poland. Hopefully we an make it a successful end to a tough week of training tomorrow. |
| **8** | 212392538055778304 | 0 | Why is \""Happy Valentines Day\"" trending? It\u2019s on the 14th of February not 12th of June smh.. |
| **10** | 264169034155696130 | 0 | Im bringing the monster load of candy tomorrow\u002c I just hope it doesn\u2019t get all squiched |
| **11** | 263192091700654080 | 0 | Apple software\u002c retail chiefs out in overhaul: SAN FRANCISCO Apple Inc CEO Tim Cook on Monday replaced the heads... http://t.co/X49ZEOsG |
| **12** | 263398998675693568 | 0 | @oluoch @victor_otti @kunjand I just watched it! Sridevi\u2019s comeback.... U remember her from the 90s?? Sun mornings on NTA ;) |
| **13** | 263650552167157762 | 0 | One of my best 8th graders Kory was excited after his touchdown today!! He did the victor cruz!!lol http://t.co/tqORFrXB |
| **14** | 260200142420992000 | 0 | #Livewire Nadal confirmed for Mexican Open in February: Rafael Nadal is set to play at the Me... http://t.co/zgUXpcnC #LiveWireAthletics |

Now check out a few racist/sexist tweets.

In [232]:

```
train[train['label'] == 1].head(10)
```

Out[232]:

| | tweet_id | label | tweet_text |
|---|---|---|---|
| **1** | 263405084770172928 | 1 | Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday. |
| **2** | 262163168678248449 | 1 | its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february. |
| **3** | 264249301910310912 | 1 | Iranian general says Israel\u2019s Iron Dome can\u2019t deal with their missiles (keep talking like that and we may end up finding out) |
| **7** | 264094586689953794 | 1 | Talking about ACT\u2019s && SAT\u2019s\u002c deciding where I want to go to college\u002c applying to colleges and everything about college stresses me out. |
| **9** | 254941790757601280 | 1 | They may have a SuperBowl in Dallas\u002c but Dallas ain\u2019t winning a SuperBowl. Not with that quarterback and owner. @S4NYC @RasmussenPoll |
| **23** | 264125591337463808 | 1 | AFC away fans on Saturday. All this stuff about the \u2019she said no\u2019 chant. It\u2019s bollocks. When he has the ball\u002c just turn your back on him. |
| **24** | 262350309781823488 | 1 | My Saturday night has consisted of me watching The Grey with my puppy while my parents throw a rager #whaa #liamneesonisbosstho |
| **25** | 264259830590603264 | 1 | Why is it so hard to find the @TVGuideMagazine these days? Went to 3 stores for the Castle cover issue. NONE. Will search again tomorrow... |
| **36** | 257951107530252289 | 1 | @MelmurMel @PBandJenelley_1 @vl_delp_ham_ Jenelle lies\u002c1st she said she was alone &the hosp.now she\u2019s saying how weird it was for Keiffer\u2019s |
| **37** | 263502599699955712 | 1 | @MyBeautyisBrown LMFAO his big ass get on my nerves\u002c you going to class tomorrow? |

There are quite a many words and characters which are not really required. So, we will try to keep only those words which are important and add value.

Let's check dimensions of the train and test dataset.

In [233]:

```
train.shape, test.shape
```

Out[233]:

```
((21465, 3), (5398, 2))
```

Train set has 31,962 tweets and test set has 17,197 tweets.

Let’s have a glimpse at label-distribution in the train dataset.

In [234]:

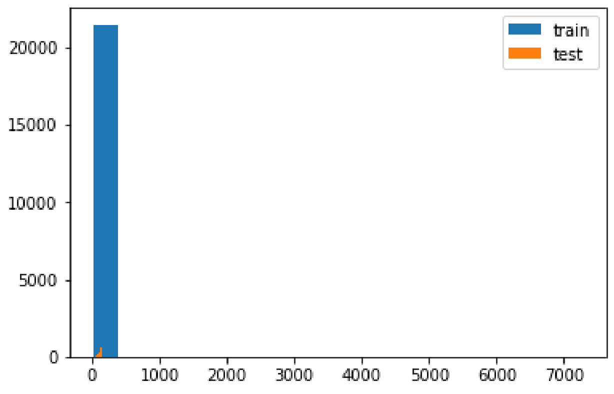
```
train["label"].value_counts()
```

Out[234]:

```
0    18078
1     3387
Name: label, dtype: int64
```

In [235]:

```
plt.hist(train.tweet_text.str.len(),bins=20, label='train')
plt.hist(test.tweet_text.str.len(), bins=20, label='test')
plt.legend()
plt.show()
```



In any natural language processing task, cleaning raw text data is an important step. It helps in getting rid of the unwanted words and characters which helps in obtaining better features. If we skip this step then there is a higher chance that you are working with noisy and inconsistent data. The objective of this step is to clean noise those are less relevant to find the sentiment of tweets such as punctuation, special characters, numbers, and terms which don’t carry much weightage in context to the text.

Before we begin cleaning, let’s first combine train and test datasets. Combining the datasets will make it convenient for us to preprocess the data. Later we will split it back into train and test data.

In [236]:

```
combi = train.append(test, ignore_index=True, sort=True)
combi.shape
```

Out[236]:

```
(26863, 3)
```

Given below is a user-defined function to remove unwanted text patterns from the tweets.

In [237]:

```
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, '', input_txt)
    return input_txt
```

We will be following the steps below to clean the raw tweets in out data.

1. We will remove the twitter handles as they are already masked as @user due to privacy concerns. These twitter handles hardly give any information about the nature of the tweet.
2. We will also get rid of the punctuations, numbers and even special characters since they wouldn’t help in differentiating different types of tweets.
3. Most of the smaller words do not add much value. For example, ‘pdx’, ‘his’, ‘all’. So, we will try to remove them as well from our data.
4. Lastly, we will normalize the text data. For example, reducing terms like loves, loving, and lovable to their base word, i.e., ‘love’.are often used in the same context. If we can reduce them to their root word, which is ‘love’. It will help in reducing the total number of unique words in our data without losing a significant amount of information.

**1. Removing Twitter Handles (@user)**

Let’s create a new column tidy_tweet, it will contain the cleaned and processed tweets. Note that we have passed “@[]*” as the pattern to the remove_pattern function. It is actually a regular expression which will pick any word starting with ‘@’.

In [238]:

```
combi['tidy_tweet'] = np.vectorize(remove_pattern)(combi['tweet_text'], "@[\w]*")
combi.head(10)
```

Out[238]:

| | label | tweet_id | tweet_text | tidy_tweet |
|---|---|---|---|---|
| **0** | 0.0 | 264183816548130816 | Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :) | Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :) |
| **1** | 1.0 | 263405084770172928 | Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday. | Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday. |
| **2** | 1.0 | 262163168678248449 | its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february. | its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february. |
| **3** | 1.0 | 264249301910310912 | Iranian general says Israel\u2019s Iron Dome can\u2019t deal with their missiles (keep talking like that and we may end up finding out) | 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** | 0.0 | 262682041215234048 | 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 | Tehran\u002c Mon Amour: Obama Tried to Establish Ties with the Mullahs http://t.co/TZZzrrKa via No Barack Obama - Vote Mitt Romney |
| **5** | 0.0 | 264229576773861376 | I sat through this whole movie just for Harry and Ron at christmas. ohlawd | I sat through this whole movie just for Harry and Ron at christmas. ohlawd |
| **6** | 0.0 | 264105751826538497 | with J Davlar 11th. Main rivals are team Poland. Hopefully we an make it a successful end to a tough week of training tomorrow. | with J Davlar 11th. Main rivals are team Poland. Hopefully we an make it a successful end to a tough week of training tomorrow. |
| **7** | 1.0 | 264094586689953794 | Talking about ACT\u2019s && SAT\u2019s\u002c deciding where I want to go to college\u002c applying to colleges and everything about college stresses me out. | Talking about ACT\u2019s && SAT\u2019s\u002c deciding where I want to go to college\u002c applying to colleges and everything about college stresses me out. |
| **8** | 0.0 | 212392538055778304 | Why is \""Happy Valentines Day\"" trending? It\u2019s on the 14th of February not 12th of June smh.. | Why is \""Happy Valentines Day\"" trending? It\u2019s on the 14th of February not 12th of June smh.. |
| **9** | 1.0 | 254941790757601280 | They may have a SuperBowl in Dallas\u002c but Dallas ain\u2019t winning a SuperBowl. Not with that quarterback and owner. @S4NYC @RasmussenPoll | They may have a SuperBowl in Dallas\u002c but Dallas ain\u2019t winning a SuperBowl. Not with that quarterback and owner. |

**2. Removing Punctuations, Numbers, and Special Characters**

Here we will replace everything except characters and hashtags with spaces. The regular expression “[^a-zA-Z#]” means anything except alphabets and ‘#’.

In [239]:

```python
combi.tidy_tweet = combi.tidy_tweet.str.replace("[^a-zA-Z#]", " ")
combi.head(10)
```

Out[239]:

| | label | tweet_id | tweet_text | tidy_tweet |
|---|---|---|---|---|
| **0** | 0.0 | 264183816548130816 | Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :) | Gas by my house hit I u m going to Chapel Hill on Sat |
| **1** | 1.0 | 263405084770172928 | Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday. | Theo Walcott is still shit u c watch Rafa and Johnny deal with him on Saturday |
| **2** | 1.0 | 262163168678248449 | its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february. | its not that I u m a GSP fan u c i just hate Nick Diaz can u t wait for february |
| **3** | 1.0 | 264249301910310912 | Iranian general says Israel\u2019s Iron Dome can\u2019t deal with their missiles (keep talking like that and we may end up finding out) | Iranian general says Israel u s Iron Dome can u t deal with their missiles keep talking like that and we may end up finding out |
| **4** | 0.0 | 262682041215234048 | 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 | Tehran u c Mon Amour Obama Tried to Establish Ties with the Mullahs http t co TZZzrrKa via No Barack Obama Vote Mitt Romney |
| **5** | 0.0 | 264229576773861376 | I sat through this whole movie just for Harry and Ron at christmas. ohlawd | I sat through this whole movie just for Harry and Ron at christmas ohlawd |
| **6** | 0.0 | 264105751826538497 | with J Davlar 11th. Main rivals are team Poland. Hopefully we an make it a successful end to a tough week of training tomorrow. | with J Davlar th Main rivals are team Poland Hopefully we an make it a successful end to a tough week of training tomorrow |
| **7** | 1.0 | 264094586689953794 | Talking about ACT\u2019s && SAT\u2019s\u002c deciding where I want to go to college\u002c applying to colleges and everything about college stresses me out. | Talking about ACT u s SAT u s u c deciding where I want to go to college u c applying to colleges and everything about college stresses me out |
| **8** | 0.0 | 212392538055778304 | Why is \""Happy Valentines Day\"" trending? It\u2019s on the 14th of February not 12th of June smh.. | Why is Happy Valentines Day trending It u s on the th of February not th of June smh |
| **9** | 1.0 | 254941790757601280 | They may have a SuperBowl in Dallas\u002c but Dallas ain\u2019t winning a SuperBowl. Not with that quarterback and owner. @S4NYC @RasmussenPoll | They may have a SuperBowl in Dallas u c but Dallas ain u t winning a SuperBowl Not with that quarterback and owner |

### 3. Removing Short Words

We have to be a little careful here in selecting the length of the words which we want to remove. So, I have decided to remove all the words having length 3 or less. For example, terms like “hmm”, “oh” are of very little use. It is better to get rid of them.

In [240]:

```python
combi.tidy_tweet = combi.tidy_tweet.apply(lambda x: ' '.join([w for w in x.split() if len(w) > 3]))
combi.head(10)
```

Out[240]:

| | label | tweet_id | tweet_text | tidy_tweet |
|---|---|---|---|---|
| **0** | 0.0 | 264183816548130816 | Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :) | house going Chapel Hill |
| **1** | 1.0 | 263405084770172928 | Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday. | Theo Walcott still shit watch Rafa Johnny deal with Saturday |
| **2** | 1.0 | 262163168678248449 | its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february. | that just hate Nick Diaz wait february |
| | | | Iranian general says Israel\u2019s Iron Dome can\u2019t | Iranian general says Israel Iron Dome |

|  | label | tweet_id | tweet_text | tidy_tweet |
|---|---|---|---|---|
| **3** | 1.0 | 264249301910310912 | deal with their missiles (keep talking like that and we may end up finding out) | deal with their missiles keep talking like that finding |
| **4** | 0.0 | 262682041215234048 | 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 | Tehran Amour Obama Tried Establish Ties with Mullahs http TZZzrrKa Barack Obama Vote Mitt Romney |
| **5** | 0.0 | 264229576773861376 | I sat through this whole movie just for Harry and Ron at christmas. ohlawd | through this whole movie just Harry christmas ohlawd |
| **6** | 0.0 | 264105751826538497 | with J Davlar 11th. Main rivals are team Poland. Hopefully we an make it a successful end to a tough week of training tomorrow. | with Davlar Main rivals team Poland Hopefully make successful tough week training tomorrow |
| **7** | 1.0 | 264094586689953794 | Talking about ACT\u2019s && SAT\u2019s\u002c deciding where I want to go to college\u002c applying to colleges and everything about college stresses me out. | Talking about deciding where want college applying colleges everything about college stresses |
| **8** | 0.0 | 212392538055778304 | Why is \""Happy Valentines Day\"" trending? It\u2019s on the 14th of February not 12th of June smh.. | Happy Valentines trending February June |
| **9** | 1.0 | 254941790757601280 | They may have a SuperBowl in Dallas\u002c but Dallas ain\u2019t winning a SuperBowl. Not with that quarterback and owner. @S4NYC @RasmussenPoll | They have SuperBowl Dallas Dallas winning SuperBowl with that quarterback owner |

You can see the difference between the raw tweets and the cleaned tweets (tidy_tweet) quite clearly. Only the important words in the tweets have been retained and the noise (numbers, punctuations, and special characters) has been removed.

**4. Text Normalization**

Here we will use nltk's PorterStemmer() function to normalize the tweets. But before that we will have to tokenize the tweets. Tokens are individual terms or words, and tokenization is the process of splitting a string of text into tokens.

In [241]:

```python
tokenized_tweet = combi.tidy_tweet.apply(lambda x: x.split())
tokenized_tweet.head()
```

Out[241]:

```
0                                                                                         [house, 
going, Chapel, Hill]
1                                              [Theo, Walcott, still, shit, watch, Rafa, Johnny, 
deal, with, Saturday]
2                                                                          [that, just, hate, Nick, Dia
wait, february]
3       [Iranian, general, says, Israel, Iron, Dome, deal, with, their, missiles, keep, talking, li
ke, that, finding]
4    [Tehran, Amour, Obama, Tried, Establish, Ties, with, Mullahs, http, TZZzrrKa, Barack, Obama, 
Vote, Mitt, Romney]
Name: tidy_tweet, dtype: object
```

In [242]:

```python
# Now we can normalize the tokenized tweets.

from nltk.stem.porter import *
stemmer = PorterStemmer()
tokenized_tweet = tokenized_tweet.apply(lambda x: [stemmer.stem(i) for i in x]) # stemming
tokenized_tweet.head()
```

Out[242]:

```
0                                                                                         [hous, go, 
apel, hill]
1                                          [theo, walcott, still, shit, watch, rafa, johnni, deal, w
th, saturday]
2                                                                      [that, just, hate, nick, diaz, 
wait, februari]
3              [iranian, gener, say, israel, iron, dome, deal, with, their, missil, keep, talk, lik
e, that, find]
```

```
4    [tehran, amour, obama, tri, establish, tie, with, mullah, http, tzzzrrka, barack, obama,
vote, mitt, romney]
Name: tidy_tweet, dtype: object
```

In [243]:

```
# Now let’s stitch these tokens back together. It can easily be done using nltk’s MosesDetokenizer
function.

for i in range(len(tokenized_tweet)):
    tokenized_tweet[i] = ' '.join(tokenized_tweet[i])
combi['tidy_tweet'] =tokenized_tweet
combi.head(10)
```

Out[243]:

| | label | tweet_id | tweet_text | tidy_tweet |
|---|---|---|---|---|
| 0 | 0.0 | 264183816548130816 | Gas by my house hit $3.39!!!! I\u2019m going to Chapel Hill on Sat. :) | hous go chapel hill |
| 1 | 1.0 | 263405084770172928 | Theo Walcott is still shit\u002c watch Rafa and Johnny deal with him on Saturday. | theo walcott still shit watch rafa johnni deal with saturday |
| 2 | 1.0 | 262163168678248449 | its not that I\u2019m a GSP fan\u002c i just hate Nick Diaz. can\u2019t wait for february. | that just hate nick diaz wait februari |
| 3 | 1.0 | 264249301910310912 | Iranian general says Israel\u2019s Iron Dome can\u2019t deal with their missiles (keep talking like that and we may end up finding out) | iranian gener say israel iron dome deal with their missil keep talk like that find |
| 4 | 0.0 | 262682041215234048 | 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 | tehran amour obama tri establish tie with mullah http tzzzrrka barack obama vote mitt romney |
| 5 | 0.0 | 264229576773861376 | I sat through this whole movie just for Harry and Ron at christmas. ohlawd | through thi whole movi just harri christma ohlawd |
| 6 | 0.0 | 264105751826538497 | with J Davlar 11th. Main rivals are team Poland. Hopefully we an make it a successful end to a tough week of training tomorrow. | with davlar main rival team poland hope make success tough week train tomorrow |
| 7 | 1.0 | 264094586689953794 | Talking about ACT\u2019s && SAT\u2019s\u002c deciding where I want to go to college\u002c applying to colleges and everything about college stresses me out. | talk about decid where want colleg appli colleg everyth about colleg stress |
| 8 | 0.0 | 212392538055778304 | Why is \""Happy Valentines Day\"" trending? It\u2019s on the 14th of February not 12th of June smh.. | happi valentin trend februari june |
| 9 | 1.0 | 254941790757601280 | They may have a SuperBowl in Dallas\u002c but Dallas ain\u2019t winning a SuperBowl. Not with that quarterback and owner. @S4NYC @RasmussenPoll | they have superbowl dalla dalla win superbowl with that quarterback owner |

In this section, we will explore the cleaned tweets. Exploring and visualizing data, no matter whether its text or any other data, is an essential step in gaining insights. Do not limit yourself to only these methods told in this course, feel free to explore the data as much as possible.

Before we begin exploration, we must think and ask questions related to the data in hand. A few probable questions are as follows:

- What are the most common words in the entire dataset?
- What are the most common words in the dataset for negative and positive tweets, respectively?
- How many hashtags are there in a tweet?
- Which trends are associated with my dataset?
- Which trends are associated with either of the sentiments? Are they compatible with the sentiments?

**A) Understanding the common words used in the tweets: WordCloud**

Now I want to see how well the given sentiments are distributed across the train dataset. One way to accomplish this task is by understanding the common words by plotting wordclouds.

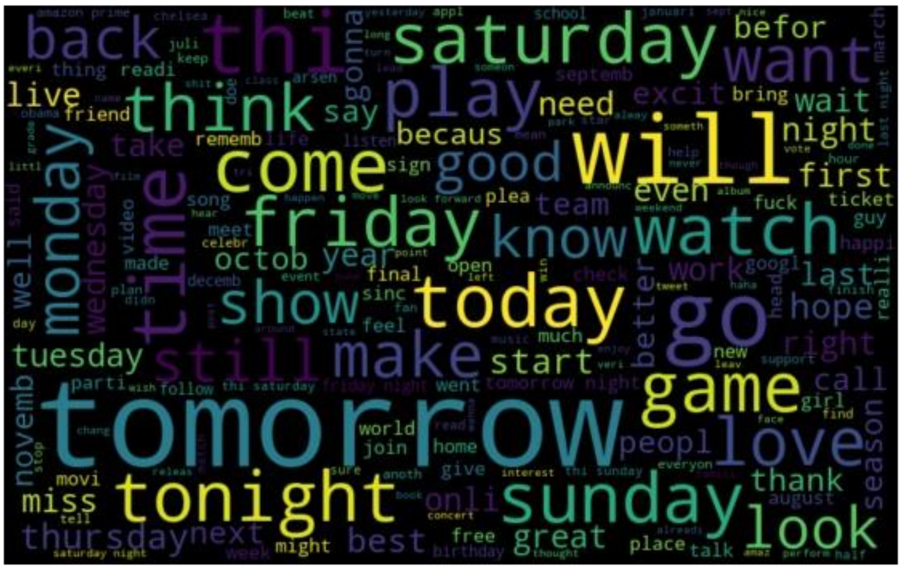
A wordcloud is a visualization wherein the most frequent words appear in large size and the less frequent words appear in smaller sizes.

Let’s visualize all the words our data using the wordcloud plot.

In [244]:

```
all_words = ' '.join([text for text in combi['tidy_tweet']])

from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(all_words
)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



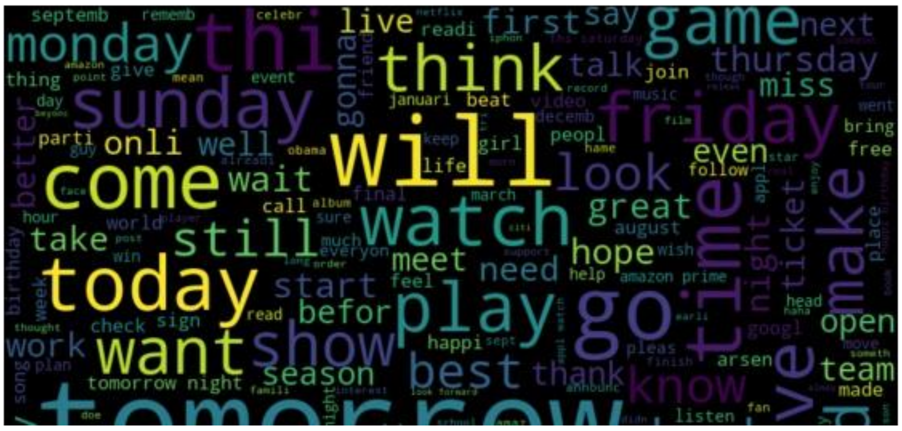
We can see most of the words are positive or neutral. Words like love, great, friend, life are the most frequent ones. It doesn’t give us any idea about the words associated with the racist/sexist tweets. Hence, we will plot separate wordclouds for both the classes (racist/sexist or not) in our train data.

**B) Words in non racist/sexist tweets**

In [245]:

```
normal_words =' '.join([text for text in combi['tidy_tweet'][combi['label'] == 0]])

wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(normal_wo
rds)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```



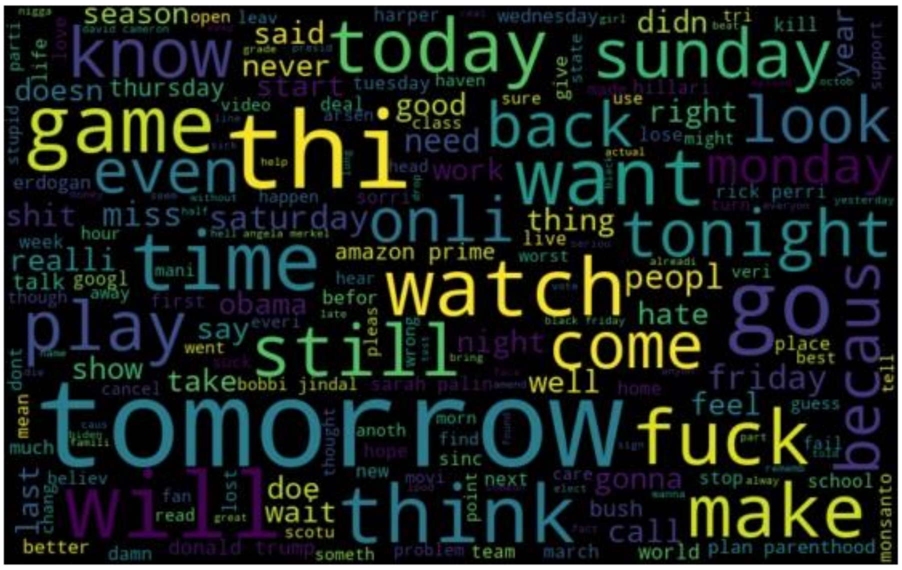
Most of the frequent words are compatible with the sentiment, i.e, non-racist/sexists tweets. Similarly, we will plot the word cloud for the other sentiment. Expect to see negative, racist, and sexist terms.

**C) Racist/Sexist Tweets**

In [246]:

```
negative_words = ' '.join([text for text in combi['tidy_tweet'][combi['label'] == 1]])

wordcloud = WordCloud(width=800, height=500, random_state=21, max_font_size=110).generate(negative_
words)
plt.figure(figsize=(10, 7))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis('off')
plt.show()
```

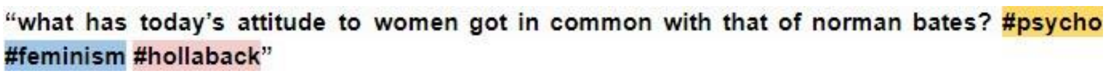


As we can clearly see, most of the words have negative connotations. So, it seems we have a pretty good text data to work on. Next we will the hashtags/trends in our twitter data.

**D) Understanding the impact of Hashtags on tweets sentiment**

Hashtags in twitter are synonymous with the ongoing trends on twitter at any particular point in time. We should try to check whether these hashtags add any value to our sentiment analysis task, i.e., they help in distinguishing tweets into the different sentiments.

For instance, given below is a tweet from our dataset:



"what has today's attitude to women got in common with that of norman bates? #psycho #feminism #hollaback"

The tweet seems sexist in nature and the hashtags in the tweet convey the same feeling.

We will store all the trend terms in two separate lists — one for non-racist/sexist tweets and the other for racist/sexist tweets.

In [247]:

```
# function to collect hashtags

def hashtag_extract(x):
    hashtags = []    # Loop over the words in the tweet
    for i in x:
        ht = re.findall(r"#(\w+)",i)
        hashtags.append(ht)
    return hashtags
```

In [248]:

```
# extracting hashtags from non racist/sexist tweets

HT_regular = hashtag_extract(combi['tidy_tweet'][combi['label'] == 0])
```

In [249]:

```
# extracting hashtags from racist/sexist tweets

HT_negative = hashtag_extract(combi['tidy_tweet'][combi['label'] ==1])
```

In [250]:

```
# unnesting list

HT_regular = sum(HT_regular,[])
HT_negative = sum(HT_negative,[])
```

Now that we have prepared our lists of hashtags for both the sentiments, we can plot the top ‘n’ hashtags. So, first let’s check the hashtags in the non-racist/sexist tweets.

**Non-Racist/Sexist Tweets**

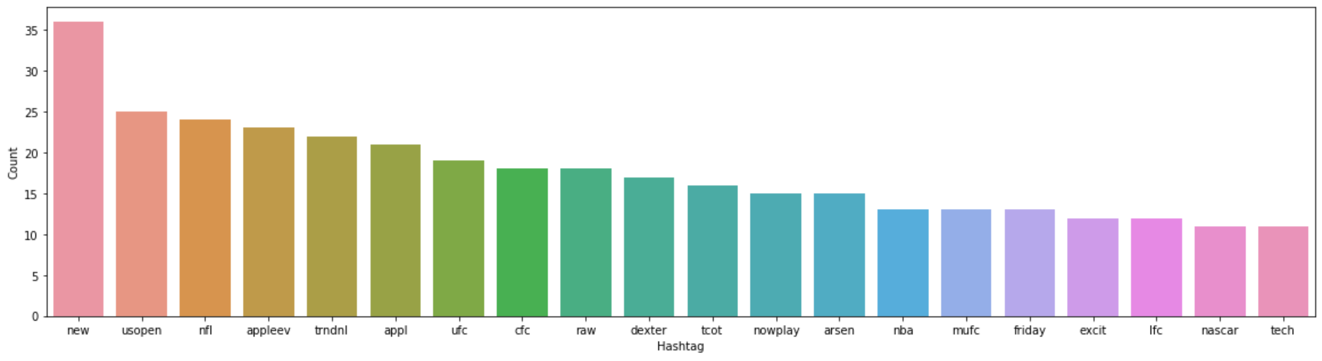
In [251]:

```
a =nltk.FreqDist(HT_regular)
d = pd.DataFrame(
    {
    'Hashtag': list(a.keys()),
    'Count': list(a.values())
    }
)
```

In [252]:

```
# selecting top 20 most frequent hashtags

d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(20,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
# plt.xticks(rotation=90)
plt.show()
```



All these hashtags are positive and it makes sense. I am expecting negative terms in the plot of the second list. Let’s check the most frequent hashtags appearing in the racist/sexist tweets.

**Racist/Sexist Tweets**

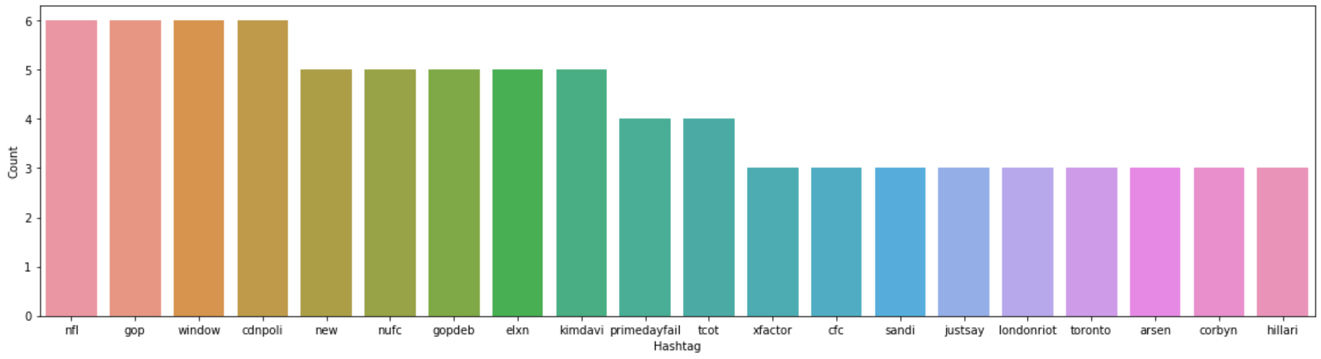
In [253]:

```
a =nltk.FreqDist(HT_negative)
d = pd.DataFrame(
    {
    'Hashtag': list(a.keys()),
    'Count': list(a.values())
    }
)
```

In [254]:

```
# selecting top 20 most frequent hashtags

d = d.nlargest(columns="Count", n = 20)
plt.figure(figsize=(20,5))
ax = sns.barplot(data=d, x= "Hashtag", y = "Count")
ax.set(ylabel = 'Count')
# plt.xticks(rotation=90)
plt.show()
```



As expected, most of the terms are negative with a few neutral terms as well. So, it’s not a bad idea to keep these hashtags in our data as they contain useful information. Next, we will try to extract features from the tokenized tweets.

**Bag-of-Words Features**

To analyse a preprocessed data, it needs to be converted into features. Depending upon the usage, text features can be constructed using assorted techniques – Bag of Words, TF-IDF, and Word Embeddings. Read on to understand these techniques in detail.

In [255]:

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import gensim
```

Let’s start with the **Bag-of-Words** Features.

Consider a Corpus C of D documents {d1,d2…..dD} and N unique tokens extracted out of the corpus C. The N tokens (words) will form a dictionary and the size of the bag-of-words matrix M will be given by D X N. Each row in the matrix M contains the frequency of tokens in document D(i).

Let us understand this using a simple example.

D1: He is a lazy boy. She is also lazy.

D2: Smith is a lazy person.

The dictionary created would be a list of unique tokens in the corpus =[‘He’,’She’,’lazy’,’boy’,’Smith’,’person’]

Here, D=2, N=6

Now the columns in the above matrix can be used as features to build a classification model.

In [256]:

```
bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')
bow = bow_vectorizer.fit_transform(combi['tidy_tweet'])
bow.shape
```

Out[256]:

```
(26863, 1000)
```

**TF-IDF Features**

This is another method which is based on the frequency method but it is different to the bag-of-words approach in the sense that it takes into account not just the occurrence of a word in a single document (or tweet) but in the entire corpus.

TF-IDF works by penalising the common words by assigning them lower weights while giving importance to words which are rare in the entire corpus but appear in good numbers in few documents.

Let's have a look at the important terms related to TF-IDF:

- TF = (Number of times term t appears in a document)/(Number of terms in the document)
- IDF = log(N/n), where, N is the number of documents and n is the number of documents a term t has appeared in.
- TF-IDF = TF*IDF

In [257]:

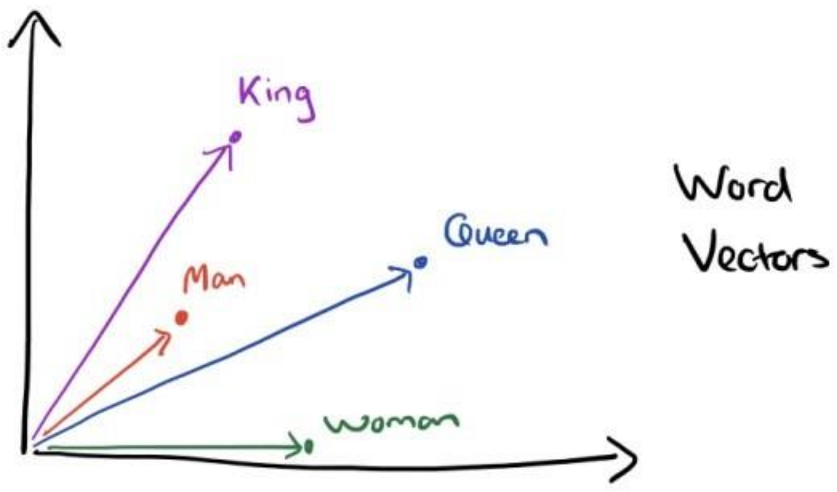
```
tfidf_vectorizer = TfidfVectorizer(max_df=0.90, min_df=2, max_features=1000, stop_words='english')
tfidf = tfidf_vectorizer.fit_transform(combi['tidy_tweet'])
tfidf.shape
```

Out[257]:

```
(26863, 1000)
```

**Word2Vec Features**

Word embeddings are the modern way of representing words as vectors. The objective of word embeddings is to redefine the high dimensional word features into low dimensional feature vectors by preserving the contextual similarity in the corpus. They are able to achieve tasks like **King -man +woman = Queen**, which is mind-blowing.



The advantages of using word embeddings over BOW or TF-IDF are:

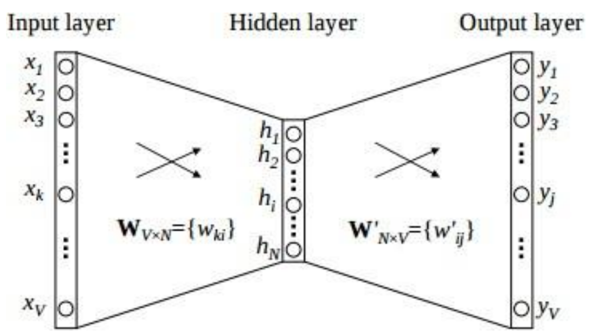
1. Dimensionality reduction - significant reduction in the no. of features required to build a model.
2. It capture meanings of the words, semantic relationships and the different types of contexts they are used in.

## 1. Word2Vec Embeddings

Word2Vec is not a single algorithm but a combination of two techniques – **CBOW (Continuous bag of words)** and **Skip-gram** model. Both of these are shallow neural networks which map word(s) to the target variable which is also a word(s). Both of these techniques learn weights which act as word vector representations.

CBOW tends to predict the probability of a word given a context. A context may be a single adjacent word or a group of surrounding words. The Skip-gram model works in the reverse manner, it tries to predict the context for a given word.

Below is a diagrammatic representation of a 1-word context window Word2Vec model.



There are three laters: - an input layer, - a hidden layer, and - an output layer.

The input layer and the output, both are one- hot encoded of size [1 X V], where V is the size of the vocabulary (no. of unique words in the corpus). The output layer is a softmax layer which is used to sum the probabilities obtained in the output layer to 1. The weights learned by the model are then used as the word-vectors.

We will go ahead with the Skip-gram model as it has the following advantages:

- It can capture two semantics for a single word. i.e it will have two vector representations of ‘apple’. One for the company Apple and the other for the fruit.
- Skip-gram with negative sub-sampling outperforms CBOW generally.

We will train a Word2Vec model on our data to obtain vector representations for all the unique words present in our corpus. There is one more option of using **pre-trained word vectors** instead of training our own model. Some of the freely available pre-trained vectors are:

1. Google News Word Vectors
2. Freebase names
3. DBPedia vectors (wiki2vec)

However, for now, we will train our own word vectors since size of the pre-trained word vectors is generally huge.

Let’s train a Word2Vec model on our corpus.

In [258]:

```
%%time

tokenized_tweet = combi['tidy_tweet'].apply(lambda x: x.split()) # tokenizing

model_w2v = gensim.models.Word2Vec(
            tokenized_tweet,
            size=200, # desired no. of features/independent variables
            window=5, # context window size
            min_count=2, # Ignores all words with total frequency lower than 2.

            sg = 1, # 1 for skip-gram model
            hs = 0,
            negative = 10, # for negative sampling
            workers= 32, # no.of cores
            seed = 34
)

model_w2v.train(tokenized_tweet, total_examples= len(combi['tidy_tweet']), epochs=20)
```

```
CPU times: user 2min 36s, sys: 453 ms, total: 2min 36s
Wall time: 41.7 s
```

Let’s play a bit with our Word2Vec model and see how does it perform. We will specify a word and the model will pull out the most similar words from the corpus.

```
In [259]:
```

```
model_w2v.wv.most_similar(positive="dinner")
```

```
Out[259]:
```

```
[('mvokmm', 0.6059072017669678),
 ('#bsbp', 0.6016099452972412),
 ('pork', 0.583012580871582),
 ('spaghetti', 0.5568623542785645),
 ('kiwani', 0.5542941093444824),
 ('#joke', 0.5426076650619507),
 ('milford', 0.5401712656021118),
 ('burrito', 0.526952862739563),
 ('steak', 0.5190435647964478),
 ('oliv', 0.498757004737854)]
```

```
In [260]:
```

```
model_w2v.most_similar(positive="trump")
```

```
Out[260]:
```

```
[('donald', 0.8883587121963501),
 ('#makeamericagreatagain', 0.6452406644821167),
 ('#trump', 0.6335049271583557),
 ('avant', 0.6087232232093811),
 ('clerk', 0.5711929798126221),
 ('#foxnew', 0.5607380270957947),
 ('#whitehous', 0.5602566003799438),
 ('bush', 0.5561950206756592),
 ('tapper', 0.5504021644592285),
 ('loyalti', 0.5499828457832336)]
```

From the above two examples, we can see that our word2vec model does a good job of finding the most similar words for a given word. But how is it able to do so? That’s because it has learned vectors for every unique word in our data and it uses cosine similarity to find out the most similar vectors (words).

Let’s check the vector representation of any word from our corpus.

```
In [261]:
```

```
model_w2v['food']
```

```
Out[261]:
```

```
array([-2.75135726e-01,  7.29500413e-01,  5.53565085e-01,  6.30067766e-01,
        2.63798863e-01, -5.14783002e-02,  4.39918160e-01, -4.11224738e-02,
        2.08707243e-01,  4.14173789e-02, -1.96380228e-01, -8.27241421e-01,
       -7.42748260e-01,  1.63105592e-01, -2.35061675e-01,  8.69490504e-01,
        3.78545463e-01, -2.64046580e-01,  6.16639331e-02, -1.29403666e-01,
        5.29682577e-01, -6.11955166e-01,  4.70178723e-01, -7.67194510e-01,
       -4.10802215e-01,  6.74093485e-01,  2.21116826e-01,  5.94996333e-01,
        1.01842575e-01,  7.11124614e-02, -4.82095540e-01, -4.77448523e-01,
       -2.10588425e-01,  5.58098435e-01, -1.86256438e-01,  2.76366681e-01,
       -3.29610765e-01,  3.26365709e-01,  3.69474977e-01, -7.11092353e-03,
        7.58812800e-02,  2.79507160e-01, -2.08109453e-01, -5.34557514e-02,
        4.66189921e-01, -6.73084855e-02,  6.41994411e-03,  3.30113441e-01,
       -1.04644024e+00,  3.29550594e-01, -4.53089401e-02,  1.96164668e-01,
       -7.07565472e-02, -1.28673285e-01, -4.47059330e-03,  2.15506274e-02,
       -3.39567661e-01,  6.17222041e-02,  5.40916502e-01,  4.86749887e-01,
        2.07796410e-01,  2.15915039e-01, -9.44240570e-01, -8.25343549e-01,
        1.09003760e-01,  6.07652366e-02,  5.82588613e-02,  2.54519224e-01,
       -7.45790720e-01, -7.20981956e-02, -7.41860121e-02,  2.90430963e-01,
        7.47254640e-02, -5.60394883e-01,  7.68075287e-02, -7.74848536e-02,
        2.20913246e-01, -5.76112211e-01, -1.26915261e-01, -2.49269262e-01,
        3.44393641e-01,  2.16096818e-01,  1.93583816e-01, -2.33861580e-01,
       -4.25969571e-01, -5.63998163e-01, -1.58494145e-01, -3.28091770e-01,
       -1.51447400e-01,  5.80517352e-01, -2.72001803e-01,  3.00407428e-02,
        3.84961993e-01, -1.71144292e-01, -2.05210060e-01, -1.72541425e-01,
        4.15758878e-01, -1.94796696e-02,  2.35569894e-01, -2.35062152e-01,
        1.41019180e-01, -1.88333973e-01, -1.97659120e-01, -3.49065453e-01,
        2.13424265e-02,  5.14120869e-02,  7.01919273e-02, -7.08526492e-01,
        4.48336780e-01, -4.87753004e-01,  9.51199234e-02, -2.15754747e-01,
       -1.57368302e-01,  7.23001659e-01, -3.53600860e-01, -7.52473027e-02,
        3.85268867e-01,  1.24274038e-01,  4.20480102e-01, -2.12307483e-01,
        1.98084906e-01, -1.75924244e-04, -7.74487853e-02,  2.95731008e-01,
        5.67436099e-01, -3.55068713e-01, -1.64094493e-01, -1.44013196e-01,
       -4.32488412e-01, -5.92827750e-03,  1.44039514e-04,  4.94389594e-01,
       -4.38278526e-01,  3.71083796e-01, -2.31631085e-01,  2.82049924e-01,
       -2.62208879e-01,  4.10405487e-01, -2.90941417e-01,  7.04328120e-02,
       -3.11613590e-01,  5.50844483e-02, -2.57199347e-01,  1.38768211e-01,
       -3.53526175e-01, -9.45541710e-02, -5.04442811e-01, -2.91295171e-01,
        2.93858707e-01,  1.20663851e-01, -8.17555636e-02,  1.14743091e-01,
       -4.25082773e-01,  1.29466563e-01,  1.98860332e-01,  4.10533011e-01,
       -2.68504322e-01,  1.96056709e-01,  4.14735347e-01, -4.54331517e-01,
       -6.89176977e-01, -5.13894081e-01,  1.12234034e-01, -3.31306428e-01,
        1.32217407e-01, -6.20090187e-01, -5.42754605e-02, -5.00587642e-01,
       -3.33158821e-01,  2.47321953e-03, -5.63275218e-01,  9.83463153e-02,
       -1.88992172e-01, -1.70784265e-01,  8.22570398e-02, -2.05007002e-01,
       -9.95834097e-02,  1.46905944e-01,  1.96689278e-01, -3.29159796e-02,
        4.71769452e-01, -1.95950553e-01,  1.60453543e-01, -7.48171031e-01,
       -1.08744889e-01, -1.21637993e-01, -1.94232464e-01,  5.32555282e-01,
        5.23986518e-01, -3.37206960e-01, -2.32007578e-02,  3.48962814e-01,
        1.07573342e+00, -1.43700331e-01, -2.96675086e-01,  1.22826889e-01,
        8.64122063e-02, -4.01468039e-01,  2.49852970e-01,  2.49267444e-01],
      dtype=float32)
```

In [262]:

```python
len(model_w2v['food']) #The length of the vector is 200
```

Out[262]:

```
200
```

**Preparing Vectors for Tweets**

Since our data contains tweets and not just words, we'll have to figure out a way to use the word vectors from word2vec model to create vector representation for an entire tweet. There is a simple solution to this problem, we can simply take mean of all the word vectors present in the tweet. The length of the resultant vector will be the same, i.e. 200. We will repeat the same process for all the tweets in our data and obtain their vectors. Now we have 200 word2vec features for our data.

We will use the below function to create a vector for each tweet by taking the average of the vectors of the words present in the tweet.

In [263]:

```python
def word_vector(tokens, size):
    vec = np.zeros(size).reshape((1, size))
    count = 0
    for word in tokens:
        try:
            vec += model_w2v[word].reshape((1, size))
            count += 1.
        except KeyError: # handling the case where the token is not in vocabulary
            continue
    if count != 0:
        vec /= count
    return vec
```

Preparing word2vec feature set…

In [264]:

```python
wordvec_arrays = np.zeros((len(tokenized_tweet), 200))
for i in range(len(tokenized_tweet)):
    wordvec_arrays[i,:] = word_vector(tokenized_tweet[i], 200)
wordvec_df = pd.DataFrame(wordvec_arrays)
wordvec_df.shape
```

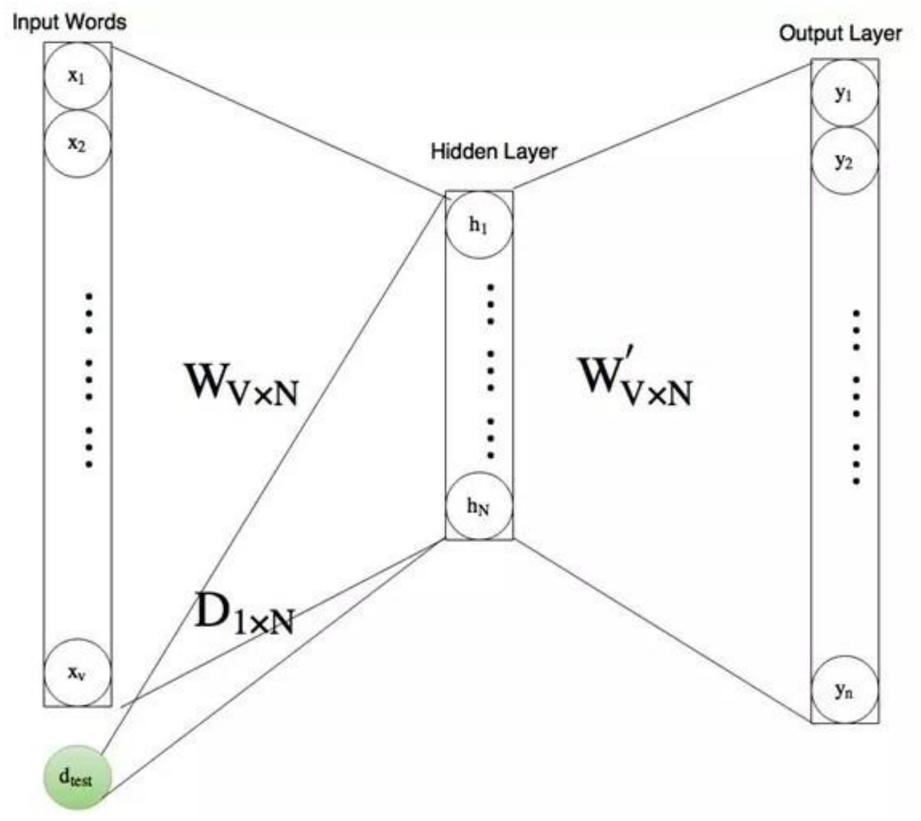
Out[264]:

```
(26863, 200)
```

Now we have 200 new features, whereas in Bag of Words and TF-IDF we had 1000 features.

**2. Doc2Vec Embedding**

Doc2Vec model is an unsupervised algorithm to generate vectors for sentence/paragraphs/documents. This approach is an extension of the word2vec. The major difference between the two is that doc2vec provides an additional context which is unique for every document in the corpus. This additional context is nothing but another feature vector for the whole document. This document vector is trained along with the word vectors.



Let’s load the required libraries.

In [265]:

```python
from tqdm import tqdm
tqdm.pandas(desc="progress-bar")
from gensim.models.doc2vec import LabeledSentence
```

To implement doc2vec, we have to **labelise** or **tag** each tokenised tweet with unique IDs. We can do so by using Gensim’s *LabeledSentence()* function.

In [266]:

```python
def add_label(twt):
    output = []
    for i, s in zip(twt.index, twt):
        output.append(LabeledSentence(s, ["tweet_" + str(i)]))
    return output

labeled_tweets = add_label(tokenized_tweet) # label all the tweets
```

Let’s have a look at the result.

In [267]:

```python
labeled_tweets[:6]
```

Out[267]:

```
[LabeledSentence(words=['hous', 'go', 'chapel', 'hill'], tags=['tweet_0']),
 LabeledSentence(words=['theo', 'walcott', 'still', 'shit', 'watch', 'rafa', 'johnni', 'deal', 'wi
th', 'saturday'], tags=['tweet_1']),
 LabeledSentence(words=['that', 'just', 'hate', 'nick', 'diaz', 'wait', 'februari'], tags=
['tweet_2']),
 LabeledSentence(words=['iranian', 'gener', 'say', 'israel', 'iron', 'dome', 'deal', 'with',
'their', 'missil', 'keep', 'talk', 'like', 'that', 'find'], tags=['tweet_3']),
 LabeledSentence(words=['tehran', 'amour', 'obama', 'tri', 'establish', 'tie', 'with', 'mullah', '
http', 'tzzzrrka', 'barack', 'obama', 'vote', 'mitt', 'romney'], tags=['tweet_4']),
 LabeledSentence(words=['through', 'thi', 'whole', 'movi', 'just', 'harri', 'christma', 'ohlawd'],
tags=['tweet_5'])]
```

Now let's train a **doc2vec** model.

In [268]:

```
%%time
model_d2v = gensim.models.Doc2Vec(dm=1, # dm = 1 for 'distributed memory' model
                                  dm_mean=1, # dm_mean = 1 for using mean of the context word
vectors
                                  vector_size=200, # no. of desired features
                                  window=5, # width of the context window

                                  negative=7, # if > 0 then negative sampling will be used
                                  min_count=5, # Ignores all words with total frequency lower than
.
                                  workers=32, # no. of cores
                                  alpha=0.1, # learning rate
                                  seed = 23, # for reproducibility
                                 )

model_d2v.build_vocab([i for i in tqdm(labeled_tweets)])

model_d2v.train(labeled_tweets, total_examples= len(combi['tidy_tweet']), epochs=15)
```

```
100%|██████████| 26863/26863 [00:00<00:00, 1150238.26it/s]
```

```
CPU times: user 1min 44s, sys: 34.2 s, total: 2min 18s
Wall time: 1min 17s
```

**Preparing doc2vec Feature Set**

In [269]:

```
docvec_arrays = np.zeros((len(tokenized_tweet), 200))
for i in range(len(combi)):
    docvec_arrays[i,:] = model_d2v.docvecs[i].reshape((1,200))

docvec_df =pd.DataFrame(docvec_arrays)
docvec_df.shape
```

Out[269]:

```
(26863, 200)
```

We are now done with all the pre-modeling stages required to get the data in the proper form and shape. We will be building models on the datasets with different feature sets prepared in the earlier sections — Bag-of-Words, TF-IDF, word2vec vectors, and doc2vec vectors. We will use the following algorithms to build models:

1. Logistic Regression
2. Support Vector Machine
3. RandomForest
4. XGBoost

**Evaluation Metric**

**F1 score** is being used as the evaluation metric. It is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It is suitable for uneven class distribution problems.

The important components of F1 score are:

1. True Positives (TP) - These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes.
2. True Negatives (TN) - These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no.
3. False Positives (FP) – When actual class is no and predicted class is yes.
4. False Negatives (FN) – When actual class is yes but predicted class in no.

**Precision** = TP/TP+FP

**Recall** = TP/TP+FN

**F1 Score** = 2(Recall * Precision) / (Recall + Precision)

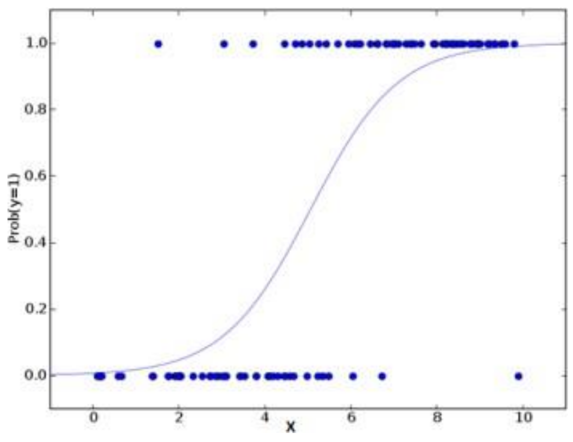
### Logistic Regression

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as the dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

The following equation is used in Logistic Regression:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta(\text{Age})$$

A typical logistic model plot is shown below. You can see probability never goes below 0 and above 1.



Read this article to know more about Logistic Regression.

In [270]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score
```

### Bag-of-Words Features

We will first try to fit the logistic regression model on the Bag-of-Words (BoW) features.

In [271]:

```
# Extracting train and test BoW features
train_bow = bow[:21465,:]
test_bow = bow[21465:,:]

# splitting data into training and validation set
xtrain_bow, xvalid_bow, ytrain, yvalid = train_test_split(train_bow, train['label'], random_state=4
2, test_size=0.3)
```

```
lreg = LogisticRegression(solver='lbfgs')

# training the model
lreg.fit(xtrain_bow, ytrain)
prediction = lreg.predict_proba(xvalid_bow) # predicting on the validation set
prediction_int = prediction[:,1] >= 0.3 # if prediction is greater than or equal to 0.3 than 1 else
0
prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int) # calculating f1 score for the validation set
```

Out[271]:

```
0.4590163934426229
```

Now let's make predictions for the test dataset and create a submission file.

In [272]:

```
test_pred = lreg.predict_proba(test_bow)
test_pred_int = test_pred[:,1] >= 0.3
test_pred_int = test_pred_int.astype(np.int)
test['label'] = test_pred_int
submission = test[['tweet_id','label']]
submission.to_csv('sub_lreg_bow.csv', index=False) # writing data to a CSV file
```

### TF-IDF Features

We'll follow the same steps as above, but now for the TF-IDF feature set.

In [273]:

```
train_tfidf = tfidf[:31962,:]
test_tfidf = tfidf[31962:,:]

xtrain_tfidf = train_tfidf[ytrain.index]
xvalid_tfidf = train_tfidf[yvalid.index]

lreg.fit(xtrain_tfidf, ytrain)

prediction = lreg.predict_proba(xvalid_tfidf)

prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int)

f1_score(yvalid, prediction_int) # calculating f1 score for the validation set
```

Out[273]:

```
0.45405982905982906
```

### Word2Vec Features

In [274]:

```
train_w2v = wordvec_df.iloc[:31962,:]
test_w2v = wordvec_df.iloc[31962:,:]

xtrain_w2v = train_w2v.iloc[ytrain.index,:]
xvalid_w2v = train_w2v.iloc[yvalid.index,:]

lreg.fit(xtrain_w2v, ytrain)

prediction = lreg.predict_proba(xvalid_w2v)

prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int)

f1_score(yvalid, prediction_int)
```

Out[274]:

```
0.43486874690440813
```

**Doc2Vec Features**

In [275]:

```
train_d2v = docvec_df.iloc[:31962,:]
test_d2v = docvec_df.iloc[31962:,:]

xtrain_d2v = train_d2v.iloc[ytrain.index,:]
xvalid_d2v = train_d2v.iloc[yvalid.index,:]

lreg.fit(xtrain_d2v, ytrain)

prediction = lreg.predict_proba(xvalid_d2v)

prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int)

f1_score(yvalid, prediction_int)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:758: ConvergenceWarning: l
bfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

Out[275]:

```
0.37286324786324787
```

Doc2Vec features do not seem to be capturing the right signals as the F1-score on validation set is quite low.

**Support Vector Machine (SVM)**

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiate the two classes as shown in the plot below:

In [276]:

```
from sklearn import svm
```

**Bag-of-Words Features**

In [277]:

```
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_bow, ytrain)
prediction = svc.predict_proba(xvalid_bow)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

Out[277]:

```
0.3782091421415153
```

Again let’s make predictions for the test dataset and create another submission file.

In [278]:

```
test_pred = svc.predict_proba(test_bow)
test_pred_int = test_pred[:,1] >= 0.3
test_pred_int = test_pred_int.astype(np.int)
test['label'] = test_pred_int
submission = test[['tweet_id','label']]
submission.to_csv('sub_svm_bow.csv',  index=False)
```

Here validation score is slightly lesser than the Logistic Regression score for bag-of-words features.

**TF-IDF Features**

In [279]:

```
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_tfidf, ytrain)
prediction = svc.predict_proba(xvalid_tfidf)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

Out[279]:

```
0.2916053019145803
```

**Word2Vec Features**

In [280]:

```
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_w2v, ytrain)
prediction = svc.predict_proba(xvalid_w2v)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

Out[280]:

```
0.18618181818181817
```

**Doc2Vec Features**

In [281]:

```
svc = svm.SVC(kernel='linear', C=1, probability=True).fit(xtrain_d2v, ytrain)
prediction = svc.predict_proba(xvalid_d2v)
prediction_int = prediction[:,1] >= 0.3
prediction_int = prediction_int.astype(np.int)
f1_score(yvalid, prediction_int)
```

Out[281]:

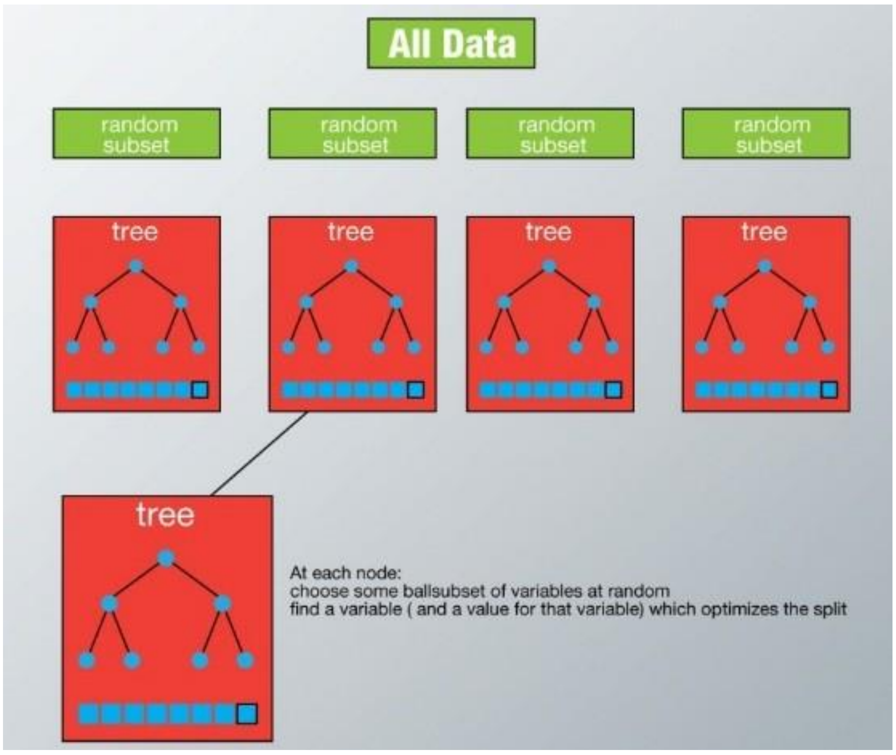
```
0.15007656967840738
```

**RandomForest**

Random Forest is a versatile machine learning algorithm capable of performing both regression and classification tasks. It is a kind of ensemble learning method, where a few weak models combine to form a powerful model. In Random Forest, we grow multiple trees as opposed to a decision single tree. To classify a new object based on attributes, each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

It works in the following manner. Each tree is planted & grown as follows:

1. Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m (m<M) is specified such that at each node, m variables are selected at random out of the M. The best split on these m variables is used to split the node. The value of m is held constant while we grow the forest.
3. Each tree is grown to the largest extent possible and there is no pruning.
4. Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some ballsubset of variables at random
find a variable ( and a value for that variable) which optimizes the split

In [282]:

```
from sklearn.ensemble import RandomForestClassifier
```

**Bag-of-Words Features**

First we will train our RandomForest model on the Bag-of-Words features and check its performance on validation set.

In [283]:

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_bow, ytrain)
prediction = rf.predict(xvalid_bow)
f1_score(yvalid, prediction) # validation score
```

Out[283]:

```
0.28077455048409405
```

Let’s make predictions for the test dataset and create another submission file.

In [284]:

```
test_pred =rf.predict(test_bow)
test['label'] = test_pred
submission = test[['tweet_id','label']]
submission.to_csv('sub_rf_bow.csv',  index=False)
```

**TF-IDF Features**

In [285]:

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_tfidf, ytrain)
```

```
prediction =rf.predict(xvalid_tfidf)
f1_score(yvalid, prediction)
```

Out[285]:

```
0.280058651026393
```

### Word2Vec Features

In [286]:

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_w2v, ytrain)
prediction = rf.predict(xvalid_w2v)
f1_score(yvalid, prediction)
```

Out[286]:

```
0.034482758620689655
```

### Doc2Vec Features

In [287]:

```
rf = RandomForestClassifier(n_estimators=400, random_state=11).fit(xtrain_d2v, ytrain)
prediction = rf.predict(xvalid_d2v)
f1_score(yvalid, prediction)
```

Out[287]:

```
0.0
```

### XGBoost

Extreme Gradient Boosting (xgboost) is an advanced implementation of gradient boosting algorithm. It has both linear model solver and tree learning algorithms. Its ability to do parallel computation on a single machine makes it extremely fast. It also has additional features for doing cross validation and finding important variables. There are many parameters which need to be controlled to optimize the model.

Some key benefits of XGBoost are:

1. **Regularization** - helps in reducing overfitting
2. **Parallel Processing** - XGBoost implements parallel processing and is blazingly faster as compared to GBM.
3. **Handling Missing Values** - It has an in-built routine to handle missing values.
4. **Built-in Cross-Validation** - allows user to run a cross-validation at each iteration of the boosting process

Check out this wonderful guide on XGBoost parameter tuning.

In [288]:

```
from xgboost import XGBClassifier
```

### Bag-of-Words Features

In [289]:

```
xgb_model = XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_bow, ytrain)
prediction = xgb_model.predict(xvalid_bow)
f1_score(yvalid, prediction)
```

Out[289]:

```
0.3316796598157335
```

In [290]:

```
test_pred =xgb_model.predict(test_bow)
test['label'] = test_pred
```

```
submission = test[['tweet_id','label']]
submission.to_csv('sub_xgb_bow.csv',  index=False)
```

**TF-IDF Features**

In [291]:

```
xgb = XGBClassifier(max_depth=6, n_estimators=1000).fit(xtrain_tfidf, ytrain)
prediction = xgb.predict(xvalid_tfidf)
f1_score(yvalid, prediction)
```

Out[291]:

```
0.3015873015873016
```

**Word2Vec Features**

In [292]:

```
xgb = XGBClassifier(max_depth=6, n_estimators=1000, nthread= 3).fit(xtrain_w2v, ytrain)
prediction = xgb.predict(xvalid_w2v)
f1_score(yvalid, prediction)
```

Out[292]:

```
0.23208722741433022
```

XGBoost model on word2vec features has outperformed all the previous models

**Doc2Vec Features**

In [293]:

```
xgb = XGBClassifier(max_depth=6, n_estimators=1000, nthread= 3).fit(xtrain_d2v, ytrain)
prediction = xgb.predict(xvalid_d2v)
f1_score(yvalid, prediction)
```

Out[293]:

```
0.12241379310344826
```

**FineTuning XGBoost + Word2Vec**

XGBoost with Word2Vec model has given us the best performance so far. Let’s try to tune it further to extract as much from it as we can. XGBoost has quite a many tuning parameters and sometimes it becomes tricky to properly tune them. This is what we are going to do in the following steps. You can refer this guide to learn more about parameter tuning in XGBoost.

In [294]:

```
import xgboost as xgb
```

Here we will use DMatrices. A DMatrix can contain both the features and the target.

In [295]:

```
dtrain = xgb.DMatrix(xtrain_w2v, label=ytrain)
dvalid = xgb.DMatrix(xvalid_w2v, label=yvalid)
dtest = xgb.DMatrix(test_w2v)
# Parameters that we are going to tune
params = {
    'objective':'binary:logistic',
    'max_depth':6,
    'min_child_weight': 1,
    'eta':.3,
    'subsample': 1,
    'colsample_bytree': 1
 }
```

```
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarning: Series.base is
deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

We will prepare a custom evaluation metric to calculate F1 score.

In [296]:

```
def custom_eval(preds, dtrain):
    labels = dtrain.get_label().astype(np.int)
    preds = (preds >= 0.3).astype(np.int)
    return [('f1_score', f1_score(labels, preds))]
```

**General Approach for Parameter Tuning**

We will follow the steps below to tune the parameters.

1. Choose a relatively high learning rate. Usually a learning rate of 0.3 is used at this stage.
2. Tune tree-specific parameters such as max_depth, min_child_weight, subsample, colsample_bytree keeping the learning rate fixed.
3. Tune the learning rate.
4. Finally tune gamma to avoid overfitting.

*Tuning max_depth and min_child_weight*

In [297]:

```
gridsearch_params = [
    (max_depth, min_child_weight)
    for max_depth in range(6,10)
    for min_child_weight in range(5,8)
    ]

max_f1 = 0. # initializing with 0

best_params = None

for max_depth, min_child_weight in gridsearch_params:
    print("CV with max_depth={}, min_child_weight={}".format(max_depth,min_child_weight))

     # Update our parameters
    params['max_depth'] = max_depth
    params['min_child_weight'] = min_child_weight

     # Cross-validation
    cv_results = xgb.cv(
        params,
        dtrain,
        feval= custom_eval,
        num_boost_round=200,
        maximize=True,
        seed=16,
        nfold=5,
        early_stopping_rounds=10
        )

# Finding best F1 Score
mean_f1 = cv_results['test-f1_score-mean'].max()
boost_rounds = cv_results['test-f1_score-mean'].idxmax()
print("\tF1 Score {} for {} rounds".format(mean_f1, boost_rounds))

if mean_f1 > max_f1:
        max_f1 = mean_f1
        best_params = (max_depth,min_child_weight)

print("Best params: {}, {}, F1 Score: {}".format(best_params[0], best_params[1], max_f1))
```

```
CV with max_depth=6, min_child_weight=5
CV with max_depth=6, min_child_weight=6
CV with max_depth=6, min_child_weight=7
CV with max_depth=7, min_child_weight=5
CV with max_depth=7, min_child_weight=6
CV with max_depth=7, min_child_weight=7
CV with max_depth=8, min_child_weight=5
CV with max_depth=8, min_child_weight=6
CV with max_depth=8, min_child_weight=7
CV with max_depth=9, min_child_weight=5
CV with max_depth=9, min_child_weight=6
CV with max_depth=9, min_child_weight=7
 F1 Score 0.385422 for 23 rounds
Best params: 9, 7, F1 Score: 0.385422
```

Updating max_depth and min_child_weight parameters.

In [298]:

```
params['max_depth'] = 9
params['min_child_weight'] = 7
```

Tuning *subsample* and *colsample*

In [299]:

```
gridsearch_params = [
    (subsample, colsample)
    for subsample in [i/10. for i in range(5,10)]
    for colsample in [i/10. for i in range(5,10)]
]

max_f1 = 0.
best_params = None

for subsample, colsample in gridsearch_params:
    print("CV with subsample={}, colsample={}".format(subsample,colsample))

    # Update our parameters
    params['colsample'] = colsample
    params['subsample'] = subsample

    cv_results = xgb.cv(
        params,
        dtrain,
        feval= custom_eval,
        num_boost_round=200,
        maximize=True,
        seed=16,
        nfold=5,
        early_stopping_rounds=10
        )

    # Finding best F1 Score
    mean_f1 = cv_results['test-f1_score-mean'].max()
    boost_rounds = cv_results['test-f1_score-mean'].idxmax()
    print("\tF1 Score {} for {} rounds".format(mean_f1, boost_rounds))

    if mean_f1 > max_f1:
        max_f1 = mean_f1
        best_params = (subsample, colsample)

print("Best params: {}, {}, F1 Score: {}".format(best_params[0], best_params[1], max_f1))
```

```
CV with subsample=0.5, colsample=0.5
 F1 Score 0.3745262 for 17 rounds
CV with subsample=0.5, colsample=0.6
 F1 Score 0.3745262 for 17 rounds
CV with subsample=0.5, colsample=0.7
 F1 Score 0.3745262 for 17 rounds
CV with subsample=0.5, colsample=0.8
 F1 Score 0.3745262 for 17 rounds
CV with subsample=0.5, colsample=0.9
 F1 Score 0.3745262 for 17 rounds
CV with subsample=0.6, colsample=0.5
 F1 Score 0.3633178 for 3 rounds
```

```
CV with subsample=0.6,colsample=0.6
 F1 Score 0.3633178 for 3 rounds
CV with subsample=0.6,colsample=0.7
 F1 Score 0.3633178 for 3 rounds
CV with subsample=0.6,colsample=0.8
 F1 Score 0.3633178 for 3 rounds
CV with subsample=0.6,colsample=0.9
 F1 Score 0.3633178 for 3 rounds
CV with subsample=0.7,colsample=0.5
 F1 Score 0.37210479999999996 for 28 rounds
CV with subsample=0.7, colsample=0.6
 F1 Score 0.37210479999999996 for 28 rounds
CV with subsample=0.7, colsample=0.7
 F1 Score 0.37210479999999996 for 28 rounds
CV with subsample=0.7, colsample=0.8
 F1 Score 0.37210479999999996 for 28 rounds
CV with subsample=0.7, colsample=0.9
 F1 Score 0.37210479999999996 for 28 rounds
CV with subsample=0.8, colsample=0.5
 F1 Score 0.38260500000000003 for 31 rounds
CV with subsample=0.8, colsample=0.6
 F1 Score 0.38260500000000003 for 31 rounds
CV with subsample=0.8, colsample=0.7
 F1 Score 0.38260500000000003 for 31 rounds
CV with subsample=0.8, colsample=0.8
 F1 Score 0.38260500000000003 for 31 rounds
CV with subsample=0.8, colsample=0.9
 F1 Score 0.38260500000000003 for 31 rounds
CV with subsample=0.9, colsample=0.5
 F1 Score 0.3659434 for 6 rounds
CV with subsample=0.9,colsample=0.6
 F1 Score 0.3659434 for 6 rounds
CV with subsample=0.9,colsample=0.7
 F1 Score 0.3659434 for 6 rounds
CV with subsample=0.9,colsample=0.8
 F1 Score 0.3659434 for 6 rounds
CV with subsample=0.9,colsample=0.9
 F1 Score 0.3659434 for 6 rounds
Best params: 0.8, 0.5, F1 Score: 0.38260500000000003
```

Updating *subsample* and *colsample_bytree*

In [ ]:

```
params['subsample'] = 0.8
params['colsample_bytree'] = 0.5
```

Now let’s tune the *learning rate*.

In [301]:

```
max_f1 = 0.
best_params = None
for eta in [.3, .2, .1, .05, .01, .005]:
    print("CV with eta={}".format(eta))
     # Update ETA
    params['eta'] = eta

     # Run CV
    cv_results = xgb.cv(
        params,
        dtrain,
        feval= custom_eval,
        num_boost_round=1000,
        maximize=True,
        seed=16,
        nfold=5,
        early_stopping_rounds=20
    )

    # Finding best F1 Score
    mean_f1 = cv_results['test-f1_score-mean'].max()
    boost_rounds = cv_results['test-f1_score-mean'].idxmax()
    print("\tF1 Score {} for {} rounds".format(mean_f1, boost_rounds))
    if mean_f1 > max_f1:
        max_f1 = mean_f1
        best_params = eta

print("Best params: {}, F1 Score: {}".format(best_params, max_f1))
```

```
CV with eta=0.3
 F1 Score 0.3805438 for 92 rounds
CV with eta=0.2
 F1 Score 0.3743656 for 29 rounds
CV with eta=0.1
 F1 Score 0.38754459999999996 for 11 rounds
CV with eta=0.05
 F1 Score 0.40683240000000004 for 26 rounds
CV with eta=0.01
 F1 Score 0.2723294 for 0 rounds
CV with eta=0.005
 F1 Score 0.2723294 for 0 rounds
Best params: 0.05, F1 Score: 0.40683240000000004
```

Let’s have a look at the final list of tuned parameters.

In [306]:

```
params = {
    'colsample': 0.5,
    'colsample_bytree': 0.5,
    'eta': 0.05,
    'max_depth': 9,
    'min_child_weight': 7,
    'objective':'binary:logistic',
    'subsample': 0.8
}
```

Finally we can now use these tuned parameters in our xgboost model. We have used early stopping of 10 which means if the model’s performance doesn’t improve under 10 rounds, then the model training will be stopped.

In [307]:

```
xgb_model = xgb.train(
    params,
    dtrain,
    feval= custom_eval,
    num_boost_round= 1000,
    maximize=True,
    evals=[(dvalid, "Validation")],
    early_stopping_rounds=10
 )
```

```
[0] Validation-error:0.178416 Validation-f1_score:0.272995
Multiple eval metrics have been passed: 'Validation-f1_score' will be used for early stopping.

Will train until Validation-f1_score hasn't improved in 10 rounds.
[1] Validation-error:0.161491   Validation-f1_score:0.272995
[2] Validation-error:0.158075   Validation-f1_score:0.272995
[3] Validation-error:0.159161   Validation-f1_score:0.272995
[4] Validation-error:0.156988   Validation-f1_score:0.272995
[5] Validation-error:0.159006   Validation-f1_score:0.272995
[6] Validation-error:0.157919   Validation-f1_score:0.272995
[7] Validation-error:0.156988   Validation-f1_score:0.272995
[8] Validation-error:0.158075   Validation-f1_score:0.272995
[9] Validation-error:0.157919   Validation-f1_score:0.272995
[10] Validation-error:0.158696   Validation-f1_score:0.278249
[11] Validation-error:0.158385   Validation-f1_score:0.296318
[12] Validation-error:0.159006   Validation-f1_score:0.319339
[13] Validation-error:0.159006   Validation-f1_score:0.339292
[14] Validation-error:0.15823 Validation-f1_score:0.364322
[15] Validation-error:0.158075   Validation-f1_score:0.382353
[16] Validation-error:0.157919   Validation-f1_score:0.393162
[17] Validation-error:0.158075   Validation-f1_score:0.400567
[18] Validation-error:0.157919   Validation-f1_score:0.405117
```

```
[19] Validation-error:0.157609 Validation-f1_score:0.41561
[20] Validation-error:0.157609   Validation-f1_score:0.415917
[21] Validation-error:0.158075   Validation-f1_score:0.412364
[22] Validation-error:0.158385   Validation-f1_score:0.423502
[23] Validation-error:0.158075   Validation-f1_score:0.422468
[24] Validation-error:0.15823 Validation-f1_score:0.416873
[25] Validation-error:0.157764   Validation-f1_score:0.415529
[26] Validation-error:0.157609   Validation-f1_score:0.414666
[27] Validation-error:0.157453   Validation-f1_score:0.411475
[28] Validation-error:0.157764   Validation-f1_score:0.408425
[29] Validation-error:0.157453 Validation-f1_score:0.40822
[30] Validation-error:0.157609   Validation-f1_score:0.403417
[31] Validation-error:0.157453   Validation-f1_score:0.400585
[32] Validation-error:0.157764 Validation-f1_score:0.395222
Stopping. Best iteration:
[22] Validation-error:0.158385 Validation-f1_score:0.423502
```

In [308]:

```
test_pred = xgb_model.predict(dtest)
# test['label'] = (test_pred >= 0.3).astype(np.int)
test = test.replace({
    0:'positive',
    1:'negative'
})
test['sentiment']=test['label']
test =test.drop(columns=['label'])
test = test[['tweet_id','sentiment','tweet_text']]
submission = test[['tweet_id','sentiment']]

print(submission)
submission.to_csv('sub_xgb_w2v_finetuned.csv', index=False)
```

Word2Vec features turned out to be most useful. Whereas **XGBoost with Word2Vec features** was the best model for this problem. This clearly shows the power of word embeddings in dealing with NLPproblems.

## CONCLUSION:

We have covered a lot in this Sentiment Analysis, but still there is plenty of room for other things to try out. Given below is a list of tasks that you can try with this data.

1. We have built so many models, we can definitely try model ensembling. A simple ensemble of all the submission files (maximum voting) yielded an F1 score of **0.47313**
2. Use Parts-of-Speech tagging to create new features.
3. Use stemming and/or lemmatization. It might help in getting rid of unnecessary words.
4. Use bi-grams or tri-grams (tokens of 2 or 3 words respectively) for Bag-of-Words and TF-IDF.
5. We can give pretrained word-embeddings models a try.