**Analysis using NLTK Vader SentimentAnalyser**

NLTK comes with an inbuilt sentiment analyser module – *nltk.sentiment.vader*—that can analyse a **piece of text and classify the sentences under positive, negative and neutral polarity** of sentiments. A code snippet of how this could be done is shown below:

|  |
| --- |
| import nltk  from nltk.sentiment.vader import SentimentIntensityAnalyzer  reviews = [‘this is good learning center’,’python is good programing language’,’python is not too secure’]  hotel\_rev = [“Great place to be when you are in Bangalore.”,  “The place was being renovated when I visited so the seating was limited.”,  “Loved the ambience, loved the food”,  “The food is delicious but not over the top.”,  “Service - Little slow, probably because too many people.”,  “The place is not easy to locate”,  “Mushroom fried rice was tasty”]    sid = SentimentIntensityAnalyzer()  for sentence in reviews:       print(sentence)       ss = sid.polarity\_scores(sentence)       for k in ss:           print(‘{0}: {1}, ‘.format(k, ss[k]), end=’’)       print() |

When running the above Python script, the different sentiment proportions for individual sentences are obtained as shown below:

|  |
| --- |
| Great place to be when you are in Bangalore.  neg: 0.0, neu: 0.661, compound: 0.6249, pos: 0.339,    The place was being renovated when I visited so the seating was limited.  neg: 0.147, neu: 0.853, compound: -0.2263, pos: 0.0,    Loved the ambience, loved the food  neg: 0.0, neu: 0.339, compound: 0.8316, pos: 0.661,    The food is delicious but not over the top.  neg: 0.168, neu: 0.623, compound: 0.1184, pos: 0.209,    Service - Little slow, probably because too many people.  neg: 0.0, neu: 1.0, compound: 0.0, pos: 0.0,    The place is not easy to locate  neg: 0.286, neu: 0.714, compound: -0.3412, pos: 0.0,    Mushroom fried rice was tasty  neg: 0.0, neu: 1.0, compound: 0.0, pos: 0.0, |

The compound value here conveys the overall positive or negative user experience.

**Analysis using Naïve’s Bayes Classifier**  
Apart from Vader, one can create one’s own classification model using Naïve’s Bayes Classifier. In the machine learning context, Naïve’s Bayes Classifier is a probabilistic classifier based on Bayes’ theorem that constructs a classification model out of training data. This classifier learns to classify the reviews to positive or negative using the supervised learning mechanism. The learning process starts by feeding in sample data that aids the classifier to construct a model to classify these reviews.

|  |
| --- |
| import nltk  from nltk.tokenize import word\_tokenize    # Step 1 – Training data  train = [("Great place to be when you are in Bangalore.", "pos"),    ("The place was being renovated when I visited so the seating was limited.", "neg"),    ("Loved the ambience, loved the food", "pos"),    ("The food is delicious but not over the top.", "neg"),    ("Service - Little slow, probably because too many people.", "neg"),    ("The place is not easy to locate", "neg"),    ("Mushroom fried rice was spicy", "pos"),  ]    # Step 2  dictionary = set(word.lower() for passage in train for word in word\_tokenize(passage[0]))    # Step 3  t = [({word: (word in word\_tokenize(x[0])) for word in dictionary}, x[1]) for x in train]    # Step 4 – the classifier is trained with sample data  classifier = nltk.NaiveBayesClassifier.train(t)    test\_data = "Manchurian was hot and spicy"  test\_data\_features = {word.lower(): (word in word\_tokenize(test\_data.lower())) for word in dictionary}    print (classifier.classify(test\_data\_features)) |

The output for the above code can be ‘pos’, denoting positive. The training data here is an array of sentences with corresponding class types – positive (pos) or negative (neg) to train the classifier. The dictionary formed in Step 2 consists of all the words obtained by tokenising or breaking this list of sentences. Step 3 starts constructing the data to be fed to the Naïve Bayes Classifier and Step 4 feeds the data to the classifier. With these steps, you might try out testing the classifier with different sentences.

*# This Python 3 environment comes with many helpful analytics libraries installed*

*# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python*

*# For example, here's several helpful packages to load in*

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

from sklearn.model\_selection import train\_test\_split *# function for splitting data to train and test sets*

import nltk

from nltk.corpus import stopwords

from nltk.classify import SklearnClassifier

from wordcloud import WordCloud,STOPWORDS

import matplotlib.pyplot as plt

%matplotlib inline

*# Input data files are available in the "../input/" directory.*

*# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory*

from subprocess import check\_output

I decided to only do sentiment analysis on this dataset, therfore I dropped the unnecessary colunns, keeping only *sentiment*and *text*.

In [2]:

data = pd.read\_csv('../input/Sentiment.csv')

*# Keeping only the neccessary columns*

data = data[['text','sentiment']]

First of all, splitting the dataset into a training and a testing set. The test set is the 10% of the original dataset. For this particular analysis I dropped the neutral tweets, as my goal was to only differentiate positive and negative tweets.

In [3]:

*# Splitting the dataset into train and test set*

train, test = train\_test\_split(data,test\_size = 0.1)

*# Removing neutral sentiments*

train = train[train.sentiment != "Neutral"]

As a next step I separated the Positive and Negative tweets of the training set in order to easily visualize their contained words. After that I cleaned the text from hashtags, mentions and links. Now they were ready for a WordCloud visualization which shows only the most emphatic words of the Positive and Negative tweets.

In [4]:

train\_pos = train[ train['sentiment'] == 'Positive']

train\_pos = train\_pos['text']

train\_neg = train[ train['sentiment'] == 'Negative']

train\_neg = train\_neg['text']

def wordcloud\_draw(data, color = 'black'):

words = ' '.join(data)

cleaned\_word = " ".join([word for word **in** words.split()

if 'http' **not** **in** word

**and** **not** word.startswith('@')

**and** **not** word.startswith('#')

**and** word != 'RT'

])

wordcloud = WordCloud(stopwords=STOPWORDS,

background\_color=color,

width=2500,

height=2000

).generate(cleaned\_word)

plt.figure(1,figsize=(13, 13))

plt.imshow(wordcloud)

plt.axis('off')

plt.show()

print("Positive words")

wordcloud\_draw(train\_pos,'white')

print("Negative words")

wordcloud\_draw(train\_neg)

Positive words

Negative words

Interesting to notice the following words and expressions in the positive word set: **truth**, **strong**, **legitimate**, **together**, **love**, **job**

In my interpretation, people tend to believe that their ideal candidate is truthful, legitimate, above good and bad.

At the same time, negative tweets contains words like: **influence**, **news**, **elevator music**, **disappointing**, **softball**, **makeup**, **cherry picking**, **trying**

In my understanding people missed the decisively acting and considered the scolded candidates too soft and cherry picking.

After the vizualization, I removed the hashtags, mentions, links and stopwords from the training set.

**Stop Word:** Stop Words are words which do not contain important significance to be used in Search Queries. Usually these words are filtered out from search queries because they return vast amount of unnecessary information. ( the, for, this etc. )

In [5]:

tweets = []

stopwords\_set = set(stopwords.words("english"))

for index, row **in** train.iterrows():

words\_filtered = [e.lower() for e **in** row.text.split() if len(e) >= 3]

words\_cleaned = [word for word **in** words\_filtered

if 'http' **not** **in** word

**and** **not** word.startswith('@')

**and** **not** word.startswith('#')

**and** word != 'RT']

words\_without\_stopwords = [word for word **in** words\_cleaned if **not** word **in** stopwords\_set]

tweets.append((words\_cleaned,row.sentiment))

test\_pos = test[ test['sentiment'] == 'Positive']

test\_pos = test\_pos['text']

test\_neg = test[ test['sentiment'] == 'Negative']

test\_neg = test\_neg['text']

As a next step I extracted the so called features with nltk lib, first by measuring a frequent distribution and by selecting the resulting keys.

In [6]:

*# Extracting word features*

def get\_words\_in\_tweets(tweets):

all = []

for (words, sentiment) **in** tweets:

all.extend(words)

return all

def get\_word\_features(wordlist):

wordlist = nltk.FreqDist(wordlist)

features = wordlist.keys()

return features

w\_features = get\_word\_features(get\_words\_in\_tweets(tweets))

def extract\_features(document):

document\_words = set(document)

features = {}

for word **in** w\_features:

features['containts(**%s**)' % word] = (word **in** document\_words)

return features

Hereby I plotted the most frequently distributed words. The most words are centered around debate nights.

In [7]:

wordcloud\_draw(w\_features)

Using the nltk NaiveBayes Classifier I classified the extracted tweet word features.

In [8]:

*# Training the Naive Bayes classifier*

training\_set = nltk.classify.apply\_features(extract\_features,tweets)

classifier = nltk.NaiveBayesClassifier.train(training\_set)

Finally, with not-so-intelligent metrics, I tried to measure how the classifier algorithm scored.

In [9]:

neg\_cnt = 0

pos\_cnt = 0

for obj **in** test\_neg:

res = classifier.classify(extract\_features(obj.split()))

if(res == 'Negative'):

neg\_cnt = neg\_cnt + 1

for obj **in** test\_pos:

res = classifier.classify(extract\_features(obj.split()))

if(res == 'Positive'):

pos\_cnt = pos\_cnt + 1

print('[Negative]: **%s**/**%s** ' % (len(test\_neg),neg\_cnt))

print('[Positive]: **%s**/**%s** ' % (len(test\_pos),pos\_cnt))

[Negative]: 849/790

[Positive]: 230/91

Epilog

In this project I was curious how well nltk and the NaiveBayes Machine Learning algorithm performs for Sentiment Analysis. In my experience, it works rather well for negative comments. The problems arise when the tweets are ironic, sarcastic has reference or own difficult context.

Consider the following tweet: *"Muhaha, how sad that the Liberals couldn't destroy Trump. Marching forward."* As you may already thought, the words **sad** and **destroy** highly influences the evaluation, although this tweet should be positive when observing its meaning and context.

To improve the evalutation accuracy, we need something to take the context and references into consideration. As my project 2.0, I will try to build an LSTM network, and benchmark its results compared to this nltk Machine Learning implementation. Stay tuned.