Sentiment\_analyzer.py:

import nltk,yaml

#emodict = {':)': ['positive'], ':\'(': ['negative'], ':(': ['negative'], ':\*': ['positive'], ':D': ['positive']}

class Splitter(object):

def \_\_init\_\_(self):

self.nltk\_splitter = nltk.data.load('tokenizers/punkt/english.pickle')

self.nltk\_tokenizer = nltk.tokenize.TreebankWordTokenizer()

def split(self, text):

"""

input format: a paragraph of text

output format: a list of lists of words.

e.g.: [['this', 'is', 'a', 'sentence'], ['this', 'is', 'another', 'one']]

"""

sentences = self.nltk\_splitter.tokenize(text)

tokenized\_sentences = [self.nltk\_tokenizer.tokenize(sent) for sent in sentences]

return tokenized\_sentences

class POSTagger(object):

def \_\_init\_\_(self):

pass

def pos\_tag(self, sentences):

"""

input format: list of lists of words

e.g.: [['this', 'is', 'a', 'sentence'], ['this', 'is', 'another', 'one']]

output format: list of lists of tagged tokens. Each tagged tokens has a

form, a lemma, and a list of tags

e.g: [[('this', 'this', ['DT']), ('is', 'be', ['VB']), ('a', 'a', ['DT']), ('sentence', 'sentence', ['NN'])],

[('this', 'this', ['DT']), ('is', 'be', ['VB']), ('another', 'another', ['DT']), ('one', 'one', ['CARD'])]]

"""

pos = [nltk.pos\_tag(sentence) for sentence in sentences]

#adapt format

pos = [[(word, word, [postag]) for (word, postag) in sentence] for sentence in pos]

return pos

class DictionaryTagger(object):

def \_\_init\_\_(self, dictionary\_paths):

files = [open(path, 'r') for path in dictionary\_paths]

dictionaries = [yaml.load(dict\_file) for dict\_file in files]

map(lambda x: x.close(), files)

self.dictionary = {}

self.max\_key\_size = 0

for curr\_dict in dictionaries:

for key in curr\_dict:

if key in self.dictionary:

self.dictionary[key].extend(curr\_dict[key])

else:

self.dictionary[key] = curr\_dict[key]

self.max\_key\_size = max(self.max\_key\_size, len(key))

def tag(self, postagged\_sentences):

return [self.tag\_sentence(sentence) for sentence in postagged\_sentences]

def tag\_sentence(self, sentence, tag\_with\_lemmas=False):

"""

the result is only one tagging of all the possible ones.

The resulting tagging is determined by these two priority rules:

- longest matches have higher priority

- search is made from left to right

"""

tag\_sentence = []

N = len(sentence)

if self.max\_key\_size == 0:

self.max\_key\_size = N

i = 0

while (i < N):

j = min(i + self.max\_key\_size, N) #avoid overflow

tagged = False

while (j > i):

expression\_form = ' '.join([word[0] for word in sentence[i:j]]).lower()

expression\_lemma = ' '.join([word[1] for word in sentence[i:j]]).lower()

if tag\_with\_lemmas:

literal = expression\_lemma

else:

literal = expression\_form

if literal in self.dictionary:

is\_single\_token = j - i == 1

original\_position = i

i = j

taggings = [tag for tag in self.dictionary[literal]]

tagged\_expression = (expression\_form, expression\_lemma, taggings)

if is\_single\_token:

original\_token\_tagging = sentence[original\_position][2]

tagged\_expression[2].extend(original\_token\_tagging)

tag\_sentence.append(tagged\_expression)

tagged = True

else:

j = j - 1

if not tagged:

tag\_sentence.append(sentence[i])

i += 1

return tag\_sentence

def value\_of(sentiment):

if sentiment == 'positive': return 1

if sentiment == 'negative': return -1

return 0

def sentence\_score(sentence\_tokens, previous\_token, acum\_score):

if not sentence\_tokens:

return acum\_score

else:

current\_token = sentence\_tokens[0]

tags = current\_token[2]

token\_score = sum([value\_of(tag) for tag in tags])

if previous\_token is not None:

previous\_tags = previous\_token[2]

if 'inc' in previous\_tags:

token\_score \*= 2.0

elif 'dec' in previous\_tags:

token\_score /= 2.0

elif 'inv' in previous\_tags:

token\_score \*= -1.0

return sentence\_score(sentence\_tokens[1:], current\_token, acum\_score + token\_score)

def sentiment\_score(review):

return sum([sentence\_score(sentence, None, 0.0) for sentence in review])

##text = """What can I say about this place. The staff of the restaurant is nice and the eggplant is not bad. Apart from that, very uninspired food, lack of atmosphere and too expensive. :( I am a staunch vegetarian and was sorely dissapointed with the veggie options on the menu. Will be the last time I visit, I recommend others to avoid."""

##text = """This is bullshit!!"""

##splitter = Splitter()

##postagger = POSTagger()

##

##splitted\_sentences = splitter.split(text)

##

##print(splitted\_sentences)

##print('\n\n')

##pos\_tagged\_sentences = postagger.pos\_tag(splitted\_sentences)

##

##print(pos\_tagged\_sentences)

##

##

##dicttagger = DictionaryTagger([ 'dicts/pos\_dict.yml', 'dicts/newpos.yml', 'dicts/pos.yml', 'dicts/pos2.yml', 'dicts/neg\_dict.yml', 'dicts/newneg.yml', 'dicts/inc.yml', 'dicts/dec.yml', 'dicts/inv.yml'])

##

##dict\_tagged\_sentences = dicttagger.tag(pos\_tagged\_sentences)

##

##print(dict\_tagged\_sentences)

##

##print(sentiment\_score(dict\_tagged\_sentences))

Twitter.py:

from tweepy import Stream, OAuthHandler

from tweepy.streaming import StreamListener

import time, urllib, re

from textblob import TextBlob

from sentiment\_analyzer import Splitter, POSTagger, DictionaryTagger, value\_of

from sentiment\_analyzer import sentence\_score, sentiment\_score

ckey = 'McfEpmGoArTag1y0YEgLmAw9B'

csecret = 'nhh8YrANBIS1CcjM6T1XZr9dDuBOFBK69fyR3vIISlNKQUzmTF'

atoken = '580508392-TGvre1m0EqXppjxZoiNoB7B4bfDBxG61m4rOwIdo'

asecret = 'ZWLk0U1KilShIZHeJnYGshMhxI11JLHL9WurC9IsNck0m'

def get\_sentiment(tweet):

analysis = TextBlob(tweet)

return analysis.sentiment.polarity

class listener(StreamListener):

def \_\_init\_\_(self):

self.count = 0

self.positive\_score = 0.0

self.p\_count = 0.0

self.n\_count = 0.0

self.neutral = 0.0

self.negative\_score = 0.0

def on\_data(self, data):

self.count+=1

if self.count>5:

print('\n\n==================')

print('ANALYSIS COMPLETE: ')

print('==================\n')

print('Average sentiment score on the Modi: '+str((self.positive\_score+self.negative\_score)/5))

print('Percentage of people who gave +ve tweets: '+str((100\*self.p\_count/5)))

print('Percentage of people who gave -ve tweets: '+str((100\*self.n\_count/5)))

print('Percentage of people who gave neutral tweets: '+str((100\*self.neutral/5)))

exit(0)

tweet = data.split(',"text":')[1].split(',"source":')[0]

text = str(tweet)

splitter = Splitter()

postagger = POSTagger()

splitted\_sentences = splitter.split(text)

pos\_tagged\_sentences = postagger.pos\_tag(splitted\_sentences)

dicttagger = DictionaryTagger([ 'dicts/pos\_dict.yml', 'dicts/newpos.yml', 'dicts/pos.yml', 'dicts/pos2.yml', 'dicts/neg\_dict.yml', 'dicts/newneg.yml', 'dicts/inc.yml', 'dicts/dec.yml', 'dicts/inv.yml'])

dict\_tagged\_sentences = dicttagger.tag(pos\_tagged\_sentences)

senti = sentiment\_score(dict\_tagged\_sentences)

if senti>0:

self.positive\_score+=senti

self.p\_count+=1

elif senti<0:

self.negative\_score+=senti

self.n\_count+=1

else:

self.neutral+=1

date = data.split('{"created\_at":"')[1].split('","id":')[0]

favorites = int(data.split(',"favorite\_count":')[1].split(',"entities":')[0])

followers = int(data.split(',"followers\_count":')[1].split(',"friends\_count":')[0])

if followers!=0:

interest\_level = 100\*favorites/followers

else:

interest\_level = 0

retweets = data.split(',"retweet\_count":')[1].split(',"favorite\_count":')[0]

try:

file = open('tDB.csv','a')

file.write('Feed: \n')

file.write(data)

file.write('\n')

file.close()

print(len(data))

print('Tweet: ', text, '\nDate: ', date, '\nFavorites: ', favorites, '\nRetweets: ', retweets, '\nSentiment Rating: ', senti, '\nInterest Level: ', "%.2f"%interest\_level,'%')

except BaseException:

print('Failed')

time.sleep(5)

return False

return True

def on\_error(self, status):

print(status)

auth = OAuthHandler(ckey, csecret)

auth.set\_access\_token(atoken, asecret)

l = listener()

twitterStream = Stream(auth, l)

twitterStream.filter(track=['Modi'])