KnowledgeBase.py:

import nltk

import re

import time

try:

import urllib.request as urllib2

except ImportError:

import urllib2

try:

from urllib.request import urlopen

except ImportError:

from urllib2 import urlopen

import http.cookiejar as cookielib

from http.cookiejar import CookieJar

import datetime

import sqlite3

import yaml

cj = CookieJar()

opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(cj))

opener.addheaders = [('User-agent', 'Chrome/61.0')]

conn = sqlite3.connect('knowledgeBase.db')

c = conn.cursor()

site = 'http://www.imdb.com'

contentArray = ['Hey! Have a very nice day.',

'I am going outside to play football in this weather.',

'Let\'s get some content from IMDB reviews first shall we?'

]

ratingArray = ['1', '2', '3']

def loadDicts(dictionary\_paths):

i=0

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)

dictionary = {}

for curr\_dict in dictionaries:

for key in curr\_dict:

if key not in dictionary:

print(key)

print(curr\_dict[key])

if curr\_dict[key][0] == 'positive':

dictionary[key] = 10

elif curr\_dict[key][0] == 'negative':

dictionary[key] = -10

elif curr\_dict[key][0] == 'dec':

dictionary[key] = 0.5

elif curr\_dict[key][0] == 'inc':

dictionary[key] = 2.0

elif curr\_dict[key][0] == 'inv':

dictionary[key] = -1.0

return dictionary

def IMDBcontent():

try:

page = 'http://www.imdb.com/?ref\_=nv\_home'

sourceCode = opener.open(page).read().decode('utf-8')

try:

#links = re.findall(r'<a.\*href=\"(.\*?)\"',sourceCode)

links = []

try:

for i in range(1,100): links.append(sourceCode.split('<div class="title"> <a href="')[i].split('">')[0])

except :

pass

print(links)

reviewLinks = []

for link in links:

try:

subpage = site+link

linkCode = opener.open(subpage).read().decode('utf-8')

tID = link[7:16]

print(tID)

for start in {0,10,20,30}:

reviewLink = 'http://www.imdb.com/title/'+tID+'/reviews?start='+str(start)

print(reviewLink)

reviewLinks.append(reviewLink)

except Exception as e:

print('Failed in the 3rd loop')

print(str(e))

print(reviewLinks)

for link in reviewLinks:

try:

for i in range(1,10):

fail = False

try:

linkCode = opener.open(link).read().decode('utf-8')

reviewStart = '<img width="102" height="12" alt="'

reviewEnd = '</p>\n\n<div class="yn"'

review = linkCode.split(reviewStart)[i].split(reviewEnd)[0]

review = '::rate::'+review+'::content::'

rating = review.split('::rate::')[1].split('/10')[0]

content = review.split('</div>\n<p>')[1].split('::content::')[0]

print(rating)

print(content)

except Exception as e:

print('Failed in 4th loop, ', str(e))

fail = True

if not fail:

ratingArray.append(rating)

contentArray.append(content)

except Exception as e:

print('Failed in 3rd b loop, ', str(e))

except Exception as e:

print('Failed in the 2nd loop')

print (str(e))

except Exception as e:

print ('Failed in the 1st loop')

print (str(e))

i = 1

for content in contentArray:

print('Rating: '+str(i))

print(ratingArray[i-1])

if len(content)>0:

print('Content: '+str(i),', Size: '+str(len(content)))

else:

print('Content: '+str(i),'None')

print(content)

i+=1

def buildWordBase(dictionary):

for key in dictionary:

currentTime = time.time()

value = dictionary[key]

dateStamp = datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M: %S')

c.execute("INSERT INTO wordBase (dateStamp, word, polarity, isNeutral) VALUES (?, ?, ?, ?);",

(dateStamp, key, value, 0))

conn.commit()

def buildMultiplierBase(dictionary):

for key in dictionary:

currentTime = time.time()

value = dictionary[key]

dateStamp = datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M: %S')

c.execute("INSERT INTO multiplierBase (dateStamp, adverb, factor, isNeutral) VALUES (?, ?, ?, ?);",

(dateStamp, key, value, 0))

conn.commit()

dict\_paths = [ 'dicts/pos\_dict.yml', 'dicts/newpos.yml', 'dicts/pos.yml', 'dicts/pos2.yml',

'dicts/neg\_dict.yml', 'dicts/newneg.yml']

multiplier\_dict = ['dicts/inc.yml', 'dicts/dec.yml', 'dicts/inv.yml']

#dictionary = loadDicts(multiplier\_dict)

#buildMultiplierBase(dictionary)

IMDBcontent()

site = 'https://www.amazon.in'

aSearch = 'https://www.amazon.in/s/ref=nb\_sb\_noss\_2?url=search-alias%3Daps&field-keywords='

keyword = 'phones'

def amazonContent():

try:

page = aSearch+keyword

sourceCode = opener.open(page).read().decode('utf-8')

try:

#links = re.findall(r'<a.\*href=\"(.\*?)\"',sourceCode)

links = []

opening = '<div class="a-row a-spacing-none"><a class="a-link-normal a-text-normal" href="'

closing = '"><span class="a-size-small a-color-secondary"></span><span class="a-size-base a-color-price s-price a-text-bold"><span class="currencyINR">'

try:

for i in range(1,100): links.append(sourceCode.split(opening)[i].split(closing)[0])

except :

pass

print(links)

reviewLinks = []

for link in links:

try:

linkCode = opener.open(link).read().decode('utf-8')

opening = '<a id="acrCustomerReviewLink" class="a-link-normal" href="'

closing = '">'

reviewLink = site+linkCode.split(opening)[1].split(closing)[0]

print(reviewLink)

reviewLinks.append(reviewLink)

except Exception as e:

print('Failed in the 3rd loop')

print(str(e))

print(reviewLinks)

for link in reviewLinks:

try:

for i in range(1,5):

linkCode = opener.open(link).read().decode('utf-8')

opening = 'class="a-size-base a-link-normal review-title a-color-base a-text-bold" href="'

closing = '</a>'

rClosing = ' out of 5 stars'

rating = linkCode.split('review-rating"><span class="a-icon-alt">')[i].split(closing)[0]

rating = '::start::'+rating

testR = rating.split(opening)[1].split(closing)[0]

testR+='::end::'

testR = testR.split('">')[1].split('::end::')[0]

rating = rating.split('::start::')[1].split(rClosing)[0]

print(testR)

print(rating)

## ratingArray.append(rating)

## contentArray.append(testR)

except Exception as e:

print('Failed in 3rd b loop: ', str(e))

except Exception as e:

print('Failed in the 2nd loop')

print (str(e))

except Exception as e:

print ('Failed in the 1st loop')

print (str(e))

#amazonContent()

def buildBase(polarity, content):

tokenized = nltk.word\_tokenize(content)

postagged = nltk.pos\_tag(tokenized)

for word in postagged:

polarity2 = polarity

if word[1][:2] == 'JJ':

row = c.execute("SELECT \* FROM wordBase;")

exists = False

currentTime = time.time()

dateStamp = datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M: %S')

for eachRow in row:

if word[0].lower() == eachRow[1]:

print(eachRow)

print(word[0])

exists = True

pol = c.execute("SELECT polarity FROM wordBase WHERE word=?", (word[0],))

for eachPol in pol:

print(eachPol)

polarity2+=eachPol[0]

c.execute("UPDATE wordBase SET dateStamp=?, word=?, polarity=?, isNeutral=? WHERE word=?",

(dateStamp, eachRow[1], polarity2, 0, word[0].lower()));

if not exists:

c.execute("INSERT INTO wordBase VALUES (dateStamp, word, polarity, isNeutral) VALUES (?, ?, ?, ?);",

(dateStamp, word[0], polarity2, 0))

print(polarity2)

conn.commit()

elif word[1] == 'RB':

row = c.execute("SELECT \* FROM multiplierBase;")

if polarity>0:

polarity2 = 2.0

elif polarity<0:

polarity2 = 0.5

exists = False

currentTime = time.time()

dateStamp = datetime.datetime.fromtimestamp(currentTime).strftime('%Y-%m-%d %H: %M: %S')

for eachRow in row:

if word[0].lower() == eachRow[1]:

print(eachRow)

print(word[0])

exists = True

pol = c.execute("SELECT factor FROM multiplierBase WHERE adverb=?", (word[0],))

for eachPol in pol:

print(eachPol)

polarity2\*=eachPol[0]

c.execute("UPDATE multiplierBase SET dateStamp=?, adverb=?, factor=?, isNeutral=? WHERE adverb=?",

(dateStamp, eachRow[1], polarity2, 0, word[0].lower()));

if not exists:

c.execute("INSERT INTO wordBase VALUES (dateStamp, adverb, factor, isNeutral) VALUES (?, ?, ?, ?);",

(dateStamp, word[0], polarity2, 0))

print(polarity2)

conn.commit()

def processContent():

try:

i = 1

for content in contentArray:

if len(content)>0 and len(content)<2000:

if int(ratingArray[i-1])>6:

buildBase(1, content)

elif int(ratingArray[i-1])<5:

buildBase(-1, content)

i+=1

except Exception as e:

print ('Error: ',e)

processContent()

features.py:

import re

import json

class Emoticons:

def analyse(self, string):

self.string = re.sub(

r'\W+:\)\(\'\{\}\-\@\>\<\=\;\[\]\!',

' ',

string)

self.string = self.string.replace('.', '')

self.string = self.string.replace('?', '')

self.words = self.string.split(" ")

if self.words[-1] == '':

del self.words[-1]

positiveEmoz = [

':)',

':-)',

':D',

':-D',

':P',

':p',

':-P',

';)',

';-)',

';D',

';-D',

':o)',

':]',

':3',

':c)',

':>',

'=]',

'8)',

'B)',

'BD',

'<3',

'=)',

':}',

'8D',

'xD',

'XD',

'X-D',

'=D',

'=3',

':-))',

':\')',

'lol',

'lol!']

negativeEmoz = [

':(',

':-(',

':(',

':-(',

':-<',

':-[',

':[',

':{',

':-||',

':@',

':\'-(',

':\'(',

'QQ',

'D:',

'D:<',

'D8',

'D;',

'DX',

'</3',

'<\\3',

'v.v',

'>.<',

'D=']

positiveCount = 0

negativeCount = 0

for i in self.words:

if i in positiveEmoz:

positiveCount += 1

if i in negativeEmoz:

negativeCount += 1

positiveEmoz, negativeEmoz = 0, 0

if positiveCount + negativeCount == 0:

return {'positive': 0, 'negative': 0}

return {'positive': positiveCount, 'negative': negativeCount}

def repairString(string):

data = {

'm': 'am',

'u': 'you',

'ua': 'your',

'yrs': 'years',

'ur': 'your',

'urs': 'yours',

'tc': 'take care',

'gn': 'good night',

'gm': 'good morning',

'ryt': 'right',

'nite': 'night',

'wat': 'what',

'abt': 'about',

'k': 'okay',

'knw': 'know',

'nt': 'not',

'w8': 'wait',

'f9': 'fine',

'wbu': 'what about you',

'kk': 'okay',

'ok': 'okay',

'na': 'no',

'don\'t': 'do not',

'won\'t': 'will not',

'gonna': 'going to',

'juz': 'just',

'jus': 'just',

'fk': 'fuck',

'wtf': 'what the fuck',

'shud': 'should',

'coz': 'because',

'cos': 'because',

'ttyl': 'talk to you later',

'ty': 'thank you',

'hlo': 'hello',

'helo': 'hello',

'hola': 'hello',

'&#x27;': '\'',

'wut': 'what',

'gtfo': 'get the fuck out',

'whr': 'where',

'y': 'why',

'ohk': 'okay'}

string = string.split(" ")

for i in string:

if i.lower() in data:

index = string.index(i)

string[index] = data[i.lower()]

return " ".join(string)

sentiment\_analyzer.py:

import nltk,yaml

from emo import split\_emo

import datetime

import sqlite3

import features

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):

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

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

tokenized\_sentences = split\_emo(tokenized\_sentences)

return tokenized\_sentences

class POSTagger(object):

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

pass

def pos\_tag(self, sentences):

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):

self.dictionary = {}

self.max\_key\_size = 0

conn = sqlite3.connect('knowledgeBase.db')

c = conn.cursor()

row = c.execute("SELECT \* FROM wordBase;")

for eachRow in row:

if eachRow[1] in self.dictionary:

pass

else:

self.max\_key\_size = max(self.max\_key\_size, len(eachRow[1]))

if int(eachRow[3])==1:

self.dictionary[eachRow[1]] = ['neutral']

elif int(eachRow[2])>0:

self.dictionary[eachRow[1]] = ['positive']

elif int(eachRow[2])<0:

self.dictionary[eachRow[1]] = ['negative']

row = c.execute("SELECT \* FROM multiplierBase;")

for eachRow in row:

if eachRow[1] in self.dictionary:

pass

else:

self.max\_key\_size = max(self.max\_key\_size, len(eachRow[1]))

if int(eachRow[3])==1:

self.dictionary[eachRow[1]] = ['neutral']

elif int(eachRow[2])>1:

self.dictionary[eachRow[1]] = ['inc']

elif int(eachRow[2])<1 and int(eachRow[2])>0:

self.dictionary[eachRow[1]] = ['dec']

elif int(eachRow[2])<0:

self.dictionary[eachRow[1]] = ['inv']

def tag(self, postagged\_sentences):

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

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

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]

prev\_token\_score = sum([value\_of(tag) for tag in 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 = """He killed him! Killed him. Great! :'( Wtf are you doing?"""

##splitter = Splitter()

##postagger = POSTagger()

##emoji = features.Emoticons()

##emos = emoji.analyse(text)

##text = features.repairString(text)

##print(emos)

##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)

##sentiments = sentiment\_score(dict\_tagged\_sentences) + emos['positive'] - emos['negative']

##print(sentiments)

Twitter.py:

from tweepy import Stream, OAuthHandler

from tweepy.streaming import StreamListener

import time, urllib, re

from textblob import TextBlob

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

from sentiment\_analyser import sentence\_score, sentiment\_score

import features

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.limit = 20

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>self.limit:

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

print('ANALYSIS COMPLETE: ')

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

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

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

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

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

exit(0)

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

text = str(tweet)

splitter = Splitter()

postagger = POSTagger()

emoji = features.Emoticons()

emos = emoji.analyse(text)

text = features.repairString(text)

splitted\_sentences = splitter.split(text)

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

dicttagger = DictionaryTagger()

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

senti = sentiment\_score(dict\_tagged\_sentences) + emos['positive'] - emos['negative']

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('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'])