```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 14 17:19:55 2018

@author: intern
"""
import re
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
from nltk.tokenize import WordPunctTokenizer
tok = WordPunctTokenizer()
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

#Import Data
data = pd.read_csv("/Users/intern/Results/
twitter_20180809.csv",encoding='utf-8')
data = data.dropna(subset=['id','author','text'])
data = data.reset_index(drop=True)
data['ID'] = data['id'].astype(np.int64)
data = data.drop('id', axis=1) #we need to change the column name "id"
to "ID" since "id" conflicts with one parameter in class defination

#Class Sentiment Defination
class Sentiment:
    """Clean and create sentiment class for each tweet in the
dataset"""
#
======================================================================
=======
# initialize 10 variables based on 10 columns in tweet dataset
#
======================================================================
=======
    def
__init__(self,date,ID,text,tags,retweet_count,favorite_count,user_scre
en_name,user_name,user_followers_count,author):
        self.date = date
        self.ID = ID
        self.text = text
        self.tags = tags
        self.retweet_count = retweet_count
        self.favorite_count = favorite_count
        self.user_screen_name = user_screen_name
        self.user_name = user_name
        self.user_followers_count = user_followers_count
        self.author = author
#
```

```
======================================================================
=======
# create tweet data cleaning and tokenize function
#
======================================================================
=======
    def clean_tweet(self):
        pat1 = r'@[A-Za-z0-9_]+' #mention pattern: @suedy
        pat2 = r'https?://[^ ]+'#URL pattern with http
        pat3 = r'RT'#retweet status
        combined_pat = r'|'.join((pat1, pat2,pat3))
        www_pat = r'www.[^ ]+'#url pattern without http
        negations_dic = {"isn't":"is not", "aren't":"are not",
"wasn't":"was not", "weren't":"were not",
                "haven't":"have not","hasn't":"has not","hadn't":"had
not","won't":"will not",
                "wouldn't":"would not", "don't":"do not",
"doesn't":"does not","didn't":"did not",
                "can't":"can not","couldn't":"could
not","shouldn't":"should not","mightn't":"might not",
                "mustn't":"must not"} #negation dictionary is used to
convert negation abbreviation
        emoji_dic = {"🤷‍♀️":" shrug", "❤️":" love", "😂":" joy", "😍":"
smiling face with heart-shaped eyes",
                "🤔":" thinking","🔥":" fire","😊":" smile","👍":"
thumbs up",
                "😀":" grin", "🤣":" laugh", "😅":" awkward","😉":"
wink",
                "😘":" kiss","☺️":" smile","🤗":" hug","🙄":" roll
eyes",
                "😏":" smirk","😣":" helpless","😫":"
distraught","🤤":" drool",
                "😒":" dissatisfied","🙃":" silly","😔":" sad","😲":"
shock",
                "☹️":" bad","😭":" cry","😨":" scared","😱":"
scream","😳":" shame",
                "😠":" angry","👻":" ghost","👿":" angry
devil","😈":" happy devil",
                "💪":" fight","👌":" ok","👏":"
congratulations","💕":" love heart",
                "✨":" sparkle"} #emoji dictionary is used to convert
emoji to word explanation
        neg_pattern = re.compile('|'.join(negations_dic.keys()))
        emo_pattern = re.compile('|'.join(emoji_dic.keys()))
        self.cleaned_tweet=[]
        for t in self.text:
            t = re.sub('\xe2\x80\x99',"'", t)
            t = emo_pattern.sub(lambda x: emoji_dic[x.group()], t)
            soup = BeautifulSoup(t, 'lxml')
            souped = soup.get_text()
            try:
                bom_removed = souped.decode("utf-8-
sig").replace(u"\ufffd", "?")
            except:
                bom_removed = souped
            stripped = re.sub(combined_pat, '', bom_removed)
            stripped = re.sub(www_pat, '', stripped)
            lower_case = stripped.lower()
            neg_handled = neg_pattern.sub(lambda x:
negations_dic[x.group()], lower_case)
            letters_only = re.sub("[^a-zA-Z]", " ", neg_handled)
            words = [x for x  in tok.tokenize(letters_only) if len(x)
> 1]
            cleaned_words = (" ".join(words)).strip()
            self.cleaned_tweet.append(cleaned_words)
        return self.cleaned_tweet
#
======================================================================
=======
#  quotation function is used to differentiate quotation tweets and
other tweets
#
======================================================================
=======
    def quotation(self):
        self.quotation_index=[]
        try:
            for i in range(len(self.cleaned_tweet)):
                if
self.cleaned_tweet[i].endswith(self.author[i].lower()):
                    self.quotation_index.append(i)
        except:
            pass
        return self.quotation_index
#
======================================================================
=======
# senti_analysis function combines tfidf vertorization and logistic
regression to predict non-quotation tweets
#
======================================================================
=======
    def senti_analysis(self,threshold=0.5):
        csv='clean_df.csv'
        clean_df=pd.read_csv(csv,encoding='utf-8')
        clean_df = clean_df.dropna(subset=['text'])
        clean_df = clean_df.reset_index(drop=True)#import labbelled
dataset as train data
        x = clean_df.text
        y = clean_df.sentiment
        tvec = TfidfVectorizer(max_features=100000,ngram_range=(1, 3))
        tvec.fit(x)#vectorize tweet in train data
        x_tfidf = tvec.transform(x)
        lr_with_tfidf = LogisticRegression()
        lr_with_tfidf.fit(x_tfidf,y)#create sentiment model
        self.unquotation_tweet = [v for i,v in
enumerate(self.cleaned_tweet) if i not in quotation_index] #only
select non-quotation tweets for sentiment analysis
        self.tfidf = tvec.transform(self.unquotation_tweet)
        self.yhat_lr =
pd.DataFrame(lr_with_tfidf.predict_proba(self.tfidf))
        self.yhat_lr['predict'] = np.where(self.yhat_lr[1]>threshold,
1, 0)#we can adjust the threshold to classify the sentiment more
accurate
        self.predict=self.yhat_lr['predict']
        self.old_data =
pd.DataFrame({'date':self.date,'ID':self.ID,'text':self.text,

'tags':self.tags,'retweet_count':self.retweet_count,

'favorite_count':self.favorite_count,

'user_screen_name':self.user_screen_name,
                                    'user_name':self.user_name,

'user_followers_count':self.user_followers_count,
                                    'author':self.author})
        self.unquotation_data =
self.old_data.drop(self.old_data.index[self.quotation_index])
        self.unquotation_data['sentiment']=self.predict
        self.quotation_data = self.old_data.ix[self.quotation_index]
        self.quotation_data['sentiment']=1 #assume all quotation
tweets are positive
        self.Data=self.unquotation_data.append(self.quotation_data)
        return self.Data
#
======================================================================
=======
# author_filter function is used to select those authors who are
significantly positive
#
======================================================================
=======
    def author_filter(self,threshold = 0.8):
        self.Data = self.Data.dropna(subset=['ID'])
        self.Author = list(set(self.Data['author']))
        positive_dic = {k: [] for k in self.Author}
        for a in self.Author:
            sublist = self.Data.loc[self.Data['author'] == a]
            positive_ratio =
float(list(sublist['sentiment']).count(1))/len(sublist)
            positive_dic[a] = positive_ratio
        self.new_author = [key for key in list(positive_dic.keys()) if
positive_dic[key]>=threshold]
        self.new_data =
self.Data[self.Data['author'].isin(self.new_author)]
        self.new_data = self.new_data[self.new_data['sentiment']==1]
        return self.new_data
#
======================================================================
=======
# user_filter function is used to select tweets that meet certain
user's feature requirements
#
======================================================================
=======
    def user_filter(self,t_retweet_count = 1,t_favorite_count = 1,
t_user_followers_count = 50):
        self.new_data['retweet_count'] =
self.new_data['retweet_count'].astype(int)
        self.new_data['favorite_count'] =
self.new_data['favorite_count'].astype(int)
        self.new_data['user_followers_count'] =
self.new_data['user_followers_count'].astype(int)
        self.final_data =
self.new_data[self.new_data['retweet_count']>t_retweet_count]
        self.final_data =
self.final_data[self.final_data['favorite_count']>t_favorite_count]
        self.final_data =
self.final_data[self.final_data['user_followers_count']>t_user_followe
rs_count]
        return self.final_data


#Generate Final Data
result = Sentiment(data['date'],data['ID'],data['text'],data['tags'],
        data['retweet_count'],data['favorite_count'],
        data['user_screen_name'],data['user_name'],
        data['user_followers_count'],data['author'])
cleaned_tweet = result.clean_tweet()
quotation_index = result.quotation()
Data = result.senti_analysis()
new_data = result.author_filter()
final_data = result.user_filter()#include the final tweets we want to
```

send messages