For homework3:

I didn’t used the csv file given since I had already processed the json file into dictionary again before given the csv file.

import json  
import csv  
#impot json and csv  
  
  
  
f = open('C:/Users/d2844/Desktop/SPRING2021/IST664/hw1/01.json',encoding="utf-8")  
p = f.readlines()  
data1 = [json.loads(i) for i in p]  
f = open('C:/Users/d2844/Desktop/SPRING2021/IST664/hw1/02.json',encoding="utf-8")  
p = f.readlines()  
data2 = [json.loads(i) for i in p]  
  
data3=data1+data2  
#read the data and join two data  
  
dict1={}  
dict1['author']=[d['author'] for d in data3]  
dict1['facebook']=[d['thread']['social']['facebook'] for d in data3]  
dict1['title']=[d['title'] for d in data3]  
dict1['published']=[d['published'] for d in data3]  
dict1['url']=[d['url'] for d in data3]  
dict1['replies\_count']=[d['thread']['replies\_count'] for d in data3]  
dict1['country']=[d['thread']['country'] for d in data3]  
dict1['facebook']=[str(d)for d in dict1['facebook']]  
dict1['text']= [d['text'] for d in data3]  
#make dictionary

import nltk  
from nltk.corpus import sentence\_polarity  
import random  
import re  
from nltk.sentiment.vader import SentimentIntensityAnalyzer  
from nltk.sentiment.util import \*  
from nltk import tokenize  
from nltk.tokenize import sent\_tokenize  
from nltk.tokenize import word\_tokenize  
from nltk.corpus import brown  
from nltk.tag import pos\_tag, map\_tag  
import nltk.classify  
#import necessary package

Then, I define the word filter for non-alpha element and stopword in English

def alpha\_filter(w):  
 pattern = re.compile('[^a-z]+')  
 if (pattern.match(w)):  
 return True  
 else:  
 return False  
#remove non alpha symbol  
stopwords = nltk.corpus.stopwords.words('english')  
  
  
def ryzen5990x(w):  
 g=nltk.word\_tokenize(w)  
 k = [a.lower( ) for a in g]  
 l = [b for b in k if not alpha\_filter(b)]  
 m = [c for c in l if c not in stopwords]  
 return m  
#remove stopword and non alpha word.

Then, I use this word filter to generate a word list and freq.dist the list for word feature usage. This is really important because I don’t want any word feature related to stopword or period’.’

wordlist=[]  
for d in data3:  
 wordlist+=ryzen5990x(d['text'])  
#process and join all the words  
sentlist=[]  
for d in data3:  
 sentlist += sent\_tokenize(d['text'])  
all\_words2 = nltk.FreqDist(wordlist)  
#process and join all the sents  
  
word\_items2 = all\_words2.most\_common(2000)  
word\_features2 = [word for (word, freq) in word\_items2]  
#generate word list for word\_feature

Then, I had trained a dataset using vader to generate a neg,pos,neu labelled dataset

traintest=[]  
for d in data3[:1000]:  
 for sent in sent\_tokenize(d['text']):  
 scores = SentimentIntensityAnalyzer().polarity\_scores(sent)  
 if scores['compound']>=0.05:  
 traintest.append((word\_tokenize(sent),'pos'))  
 if scores['compound']> -0.05 and scores['compound']<0.05:  
 traintest.append((word\_tokenize(sent),'neg'))  
 if scores['compound']< -0.05:  
 traintest.append((word\_tokenize(sent),'neu'))  
#use vader to train a labelled dataset

random.shuffle(traintest)  
#shuffle the traintest

Then, I define the word feature method and use it to generate a word feature using the word list I processed. Then, I use nativebayes.classifier() to train a classifier

def document\_features(document, word\_features):  
 document\_words = set(document)  
 features = {}  
 for word in word\_features:  
 features['contains({})'.format(word)] = (word in document\_words)  
 return features  
#def feature method  
  
  
featuresets = [(document\_features(d,word\_features2), c) for (d,c) in traintest]  
train\_set, test\_set = featuresets[1000:], featuresets[:1000]  
classifier = nltk.NaiveBayesClassifier.train(train\_set)  
print (nltk.classify.accuracy(classifier, test\_set))  
classifier.show\_most\_informative\_features(30)  
#train classifier using this feature

print (nltk.classify.accuracy(classifier, test\_set))

classifier.show\_most\_informative\_features(30)

0.711

Most Informative Features

contains(killed) = True neu : pos = 111.5 : 1.0

contains(ensure) = True pos : neg = 79.8 : 1.0

contains(severe) = True neu : neg = 70.8 : 1.0

contains(low) = True neu : neg = 64.6 : 1.0

contains(fear) = True neu : neg = 62.2 : 1.0

contains(interest) = True pos : neg = 59.2 : 1.0

contains(isolation) = True neu : neg = 58.3 : 1.0

contains(significant) = True pos : neg = 51.2 : 1.0

contains(emergency) = True neu : neg = 50.4 : 1.0

contains(number) = True pos : neg = 46.3 : 1.0

contains(strain) = True neu : neg = 45.1 : 1.0

contains(suspended) = True neu : neg = 44.8 : 1.0

contains(free) = True pos : neu = 43.7 : 1.0

contains(toll) = True neu : neg = 41.8 : 1.0

contains(help) = True pos : neg = 40.8 : 1.0

contains(ill) = True neu : neg = 37.8 : 1.0

contains(profit) = True pos : neg = 37.7 : 1.0

contains(illness) = True neu : neg = 34.8 : 1.0

contains(reached) = True neu : neg = 34.6 : 1.0

contains(infected) = True neu : neg = 34.0 : 1.0

contains(priority) = True pos : neg = 33.8 : 1.0

contains(top) = True pos : neg = 33.5 : 1.0

contains(best) = True pos : neg = 31.8 : 1.0

contains(confidence) = True pos : neg = 31.7 : 1.0

contains(stopped) = True neu : neg = 31.6 : 1.0

contains(strong) = True pos : neg = 31.2 : 1.0

contains(weaker) = True neu : neg = 30.9 : 1.0

contains(alert) = True pos : neg = 30.6 : 1.0

contains(death) = True neu : pos = 30.2 : 1.0

contains(ban) = True neu : neg = 30.2 : 1.0

Then, I import the sl file we used for class and define a sl-feature method. Using that method, I trained a sl feature based classifier using nativebayes. This feature is basically calculating the possibility of a word’s relationship to the possibility of sent being pos, neg, neu

def readSubjectivity(path):  
 flexicon = open(path, 'r')  
 # initialize an empty dictionary  
 sldict = { }  
 for line in flexicon:  
 fields = line.split() # default is to split on whitespace  
 # split each field on the '=' and keep the second part as the value  
 strength = fields[0].split("=")[1]  
 word = fields[2].split("=")[1]  
 posTag = fields[3].split("=")[1]  
 stemmed = fields[4].split("=")[1]  
 polarity = fields[5].split("=")[1]  
 if (stemmed == 'y'):  
 isStemmed = True  
 else:  
 isStemmed = False  
 # put a dictionary entry with the word as the keyword  
 # and a list of the other values  
 sldict[word] = [strength, posTag, isStemmed, polarity]  
 return sldict  
#method reading SL  
  
def SL\_features(document, word\_features2, SL):  
 document\_words = set(document)  
 features = {}  
 posscore=0  
 for word in word\_features2:  
 features['contains(%s)' % word] = (word in document\_words)  
  
 for word in document\_words:  
 if word in SL:  
 strength, posTag, isStemmed, polarity = SL[word]  
 if strength == 'weaksubj' and polarity == 'positive':  
 posscore+=1  
 if strength == 'strongsubj' and polarity == 'positive':  
 posscore += 2  
 if strength == 'weaksubj' and polarity == 'negative':  
 posscore -= 1  
 if strength == 'strongsubj' and polarity == 'negative':  
 posscore -= 2  
 features['positivecount']= posscore  
 features['negativecount']= (-posscore)  
 features['neutralcount']= 2-abs(posscore)  
 return features  
#def SL feature with neg, pos, neu score  
SLpath = 'subjclueslen1-HLTEMNLP05.tff'  
SL = readSubjectivity(SLpath)  
#import SL  
  
SL\_featuresets = [(SL\_features(d, word\_features2, SL), c) for (d,c) in traintest]  
#def SL feature dataset

SL\_featuresets = [(SL\_features(d, word\_features2, SL), c) for (d,c) in traintest]  
#def SL feature dataset  
  
train\_set2, test\_set2 = SL\_featuresets[1000:], SL\_featuresets[:1000]  
classifier2 = nltk.NaiveBayesClassifier.train(train\_set2)  
print(nltk.classify.accuracy(classifier2, test\_set2))  
#train a SL feature classifier

classifier2.show\_most\_informative\_features(30)

print(nltk.classify.accuracy(classifier2, test\_set2))

0.675

classifier2.show\_most\_informative\_features(30)

^CMost Informative Features

contains(killed) = True neu : pos = 111.5 : 1.0

contains(ensure) = True pos : neg = 79.8 : 1.0

contains(severe) = True neu : neg = 70.8 : 1.0

contains(low) = True neu : neg = 64.6 : 1.0

contains(fear) = True neu : neg = 62.2 : 1.0

contains(interest) = True pos : neg = 59.2 : 1.0

contains(isolation) = True neu : neg = 58.3 : 1.0

negativecount = -6 pos : neg = 52.8 : 1.0

positivecount = 6 pos : neg = 52.8 : 1.0

contains(significant) = True pos : neg = 51.2 : 1.0

contains(emergency) = True neu : neg = 50.4 : 1.0

contains(number) = True pos : neg = 46.3 : 1.0

contains(strain) = True neu : neg = 45.1 : 1.0

contains(suspended) = True neu : neg = 44.8 : 1.0

contains(free) = True pos : neu = 43.7 : 1.0

contains(toll) = True neu : neg = 41.8 : 1.0

contains(help) = True pos : neg = 40.8 : 1.0

contains(ill) = True neu : neg = 37.8 : 1.0

contains(profit) = True pos : neg = 37.7 : 1.0

contains(illness) = True neu : neg = 34.8 : 1.0

contains(reached) = True neu : neg = 34.6 : 1.0

contains(infected) = True neu : neg = 34.0 : 1.0

contains(priority) = True pos : neg = 33.8 : 1.0

contains(top) = True pos : neg = 33.5 : 1.0

contains(best) = True pos : neg = 31.8 : 1.0

contains(confidence) = True pos : neg = 31.7 : 1.0

contains(stopped) = True neu : neg = 31.6 : 1.0

contains(strong) = True pos : neg = 31.2 : 1.0

contains(weaker) = True neu : neg = 30.9 : 1.0

negativecount = -5 pos : neg = 30.7 : 1.0

Then, I used classifier2 with SL feature to classify the dataset and count the neg,pos,neu sent while extracting them into separate list for further usage. I personally are in favor more about SL features over regular feature because of the involve of subjectivity improves the robust(needed validation, I like this feature also because I had modified it).

negstcount=[]  
posstcount=[]  
neustcount=[]  
#generate list of negsent possent neusent count  
negst=[]  
posst=[]  
neust=[]  
#extract all the neg, pos, neu sent into individual list  
  
for d in data3:  
 tempneg=0  
 temppos=0  
 tempneu=0  
 for sent in sent\_tokenize(d['text']):  
 result=classifier2.classify(SL\_features(word\_tokenize(sent),word\_features2,SL))  
 if result =='neg':  
 negst.append(sent)  
 tempneg+=1  
 if result =='pos':  
 posst.append(sent)  
 temppos+=1  
 if result =='neu':  
 neust.append(sent)  
 tempneu+=1  
 negstcount.append(tempneg)  
 posstcount.append(temppos)  
 neustcount.append(tempneu)  
#generate neg,pos,neu count

Then, I had imported chunking package and write a chunking grammar used to extract adjective phrase, verb phrase and adverb phrase into separate list. The chunking grammar I defined extracts the AdjP containing NP and PP, AdvP containing VP. I do it in this way because those NP,PP,VP contains very important information regards the adj. or adv. For example, rapid soled doesn’t mean much, but rapid spread can give me way more information to figure the context.

import nltk.chunk  
import itertools  
#import package for chunking  
  
  
grammar = r"""  
 NP: {<DT|NN.\*>+} # Chunk sequences of DT, JJ, NN  
 PP: {<IN><NP|VB.\*>} # Chunk prepositions followed by NP  
 VP: {<VB.\*><NP|PP|CLAUSE>+$} # Chunk verbs and their arguments  
 AJP: {<JJ.\*>|<JJ.\*><NP|PP>} # Chunk adjetive phrase  
 AVP: {<VB.\*><RB.\*>|<RB.\*><PP>} # Chunk adverb phrase  
 CLAUSE: {<NP><VP>} # Chunk NP, VP  
 """  
#def chunking grammar  
text1=word\_tokenize("I badly at playing")  
text2=word\_tokenize("I am bad")  
cp = nltk.RegexpParser(grammar)  
cp.parse(nltk.pos\_tag(text1))  
cp.parse(nltk.pos\_tag(text2))  
#test grammar  
tree=cp.parse(nltk.pos\_tag(text1))  
for subtree in tree.subtrees():  
 if subtree.label() == 'AVP':  
 print(nltk.chunk.regexp.UnChunkRule(grammar, subtree))  
#test classfiying method  
  
posadj=[]  
posadv=[]  
posvp=[]  
negadj=[]  
negadv=[]  
negvp=[]  
#create list of neg , pos adj phrase, verb phrase, adv phrase  
for sent in posst:  
 tree = cp.parse(nltk.pos\_tag(word\_tokenize(sent)))  
 for subtree in tree.subtrees():  
 if subtree.label() == 'VP':  
 posvp.append(subtree)  
 if subtree.label() == 'AJP':  
 posadj.append(subtree)  
 if subtree.label() == 'AVP':  
 posadv.append(subtree)  
#generate these phrases  
for sent in negst:  
 tree = cp.parse(nltk.pos\_tag(word\_tokenize(sent)))  
 for subtree in tree.subtrees():  
 if subtree.label() == 'VP':  
 negvp.append(subtree)  
 if subtree.label() == 'AJP':  
 negadj.append(subtree)  
 if subtree.label() == 'AVP':  
 negadv.append(subtree)  
#extract string in these list for freqdist

Then, I define a untoken method to flatten the subtree I extracted in order to freq.dist them. It is basically a reverse token and tag method.

def untoken(docu):  
 leavelist = []  
 for d in docu:  
 temppp = ''  
 for (e, c) in d.leaves():  
 temppp += ' '  
 temppp += e  
 leavelist.append(temppp)  
 return leavelist  
#def method generate string for analyze  
  
  
negvpst=untoken(negvp)  
negadjst=untoken(negadj)  
negadvst=untoken(negadv)  
posvpst=untoken(posvp)  
posadjst=untoken(posadj)  
posadvst=untoken(posadv)  
#extract string in these list for freqdist  
fdnvp=nltk.FreqDist(negvpst)  
fdnav=nltk.FreqDist(negadvst)  
fdnaj=nltk.FreqDist(negadjst)  
fdpvp=nltk.FreqDist(posvpst)  
fdpav=nltk.FreqDist(posadvst)  
fdpaj=nltk.FreqDist(posadjst)  
#freq.dist these strings

fdpav.most\_common(50)  
fdpaj.most\_common(50)  
fdpvp.most\_common(50)  
fdnaj.most\_common(50)  
fdnvp.most\_common(50)  
fdnav.most\_common(50)  
#show the top 50 for all freq.dist

50 most frequent positive adv phrase are:

[(' is not', 6914),

(" do n't", 5797),

(' are not', 5446),

(' does not', 4020),

(' is also', 4009),

(' did not', 3836),

(' do not', 3065),

(' is still', 2933),

(' has also', 2419),

(' is now', 2396),

(' are also', 2339),

(' ’ s', 2303),

(' are still', 2158),

(' was not', 1872),

(' has not', 1869),

(' have also', 1703),

(" does n't", 1663),

(' are now', 1652),

(' have not', 1438),

(' was also', 1362),

(" 's not", 1360),

(' ’ ve', 1347),

(' have already', 1251),

(" did n't", 1239),

(' do so', 1221),

(' is very', 1191),

(' is just', 1151),

(' has already', 1149),

(' is currently', 1070),

(' go back', 1053),

(" is n't", 1022),

(' were not', 1004),

(' ’ m', 994),

(' are very', 942),

(' come back', 941),

(' get back', 928),

(' is always', 874),

(' is well', 858),

(" 're not", 851),

(' is more', 796),

(' were also', 785),

(" are n't", 769),

(' s not', 766),

(' had already', 753),

(' go ahead', 740),

(' be more', 738),

(' ’ re', 727),

(' are currently', 722),

(' is already', 708),

(' are more', 706)]

Here, you can see usage of very and more a lot

50 most frequent positive adj phrase are:

[(' new', 37013),

(' other', 34238),

(' more', 31193),

(' Chinese', 31088),

(' last', 22153),

(' positive', 20856),

(' novel', 20771),

(' first', 19489),

(' s', 19315),

(' global', 18427),

(' good', 17020),

(' “', 16662),

(' economic', 16168),

(' such', 16155),

(' medical', 16022),

(' public', 14464),

(' best', 14054),

(' many', 13975),

(' ’', 13388),

(' strong', 13175),

(' due', 12142),

(' important', 11531),

(' next', 11185),

(' financial', 11167),

(' top', 11074),

(' local', 10970),

(' free', 10133),

(' international', 9790),

(' able', 9586),

(' same', 9583),

(' current', 9427),

(' potential', 8644),

(' major', 8553),

(' possible', 8394),

(' higher', 8370),

(' net', 7900),

(' sure', 7493),

(' second', 7451),

(' significant', 7450),

(' epidemic', 7415),

(' large', 7406),

(' latest', 7325),

(' much', 7268),

(' full', 7186),

(' high', 7180),

(' better', 7161),

(' further', 7077),

(' recent', 6823),

(' own', 6815),

(' social', 6794)]

Most of the informative adj. are related to newness of covid, recent, present, new.

50 most frequent positive verb phrase are:

[(' GET IT', 31),

(' protected ]', 21),

(' miss a story', 21),

(' < >', 13),

(' begins exile in Hawaii', 9),

(' realise Quote', 7),

(' s assault on the rule of law By Ian Millhiser', 7),

(' is Global Research', 6),

(' Related articles', 5),

(' re-purpose wardrobe staples These Advertisement', 5),

(' Related News', 5),

(' ” The Associated Press', 4),

(' do Steve Jobs', 4),

(' kelowna News', 4),

(' See All', 3),

(' update SOURCE Public Health Agency of Canada', 3),

(' lifts H1 profit', 3),

(' Press Latest Digitimes news', 3),

(' cut profit', 3),

(' Trending topics', 3),

(' provide housing support…', 3),

(' quoted in these press reviews', 3),

(' correct at time of publication Topics', 3),

(' have Advertisement Home', 3),

(' are for entertainment', 2),

(' spotted in Dubai', 2),

(' alarmed investors', 2),

(' Follow the markets', 2),

(' Published by HT Digital Content Services with permission from HT Gurgaon ....',

2),

(' Related posts', 2),

(' s trade with China', 2),

(' read disclaimer', 2),

(' is a correspondent at the Post', 2),

(' procedures. ’ ‘', 2),

(' coronavirus impact', 2),

(' eFM News.■ National', 2),

(' ship off Japan ask for help', 2),

(' help Aust', 2),

(' .. wow pic.twitter.com/Up9PSgeAKz', 2),

(' apologises over coronavirus video Continue reading', 2),

(' Recommended Shopping', 2),

(' throws glass at NME awards crowd', 2),

(' Related Stories', 2),

(' copyright Getty Images', 2),

(' c.2020 The New York Times Company More stories', 2),

(' caused by the disease.=FRESH NEWS', 2),

(' flatten. ” — Reuters', 2),

(' get help in seeking refunds for cancelled holiday plans', 2),

(' receive the watermelons', 2),

(' evacuated after virus case', 2)]

Most positive verb phrase are related to procedure treating covid

50 most frequent negative adv phrase are:

[(' more', 29014),

(' new', 27693),

(' other', 23800),

(' Chinese', 21801),

(' last', 18551),

(' first', 16397),

(' s', 11242),

(' “', 9814),

(' many', 8606),

(' ’', 8293),

(' medical', 7881),

(' confirmed', 7813),

(' such', 7473),

(' next', 7101),

(' same', 7095),

(' due', 6945),

(' local', 6222),

(' second', 6206),

(' latest', 6051),

(' More', 5538),

(' Japanese', 5294),

(' few', 5222),

(' several', 4775),

(' recent', 4731),

(' Canadian', 4614),

(' full', 4375),

(' total', 4354),

(' early', 4320),

(' public', 4198),

(' different', 4175),

(' previous', 4094),

(' daily', 4091),

(' most', 4061),

(' available', 4015),

(' higher', 3957),

(' global', 3944),

(' international', 3912),

(' major', 3895),

(' central', 3893),

(' biggest', 3852),

(' British', 3774),

(' past', 3737),

(' late', 3580),

(' least', 3544),

(' own', 3520),

(' much', 3516),

(' American', 3466),

(' further', 3335),

(' foreign', 3280),

(' big', 3195)]

On contrary to post sent, negative sentence’s adv phrase are all sole adverb. Representing newness of the virus

50 most frequent negative adj phrase are:

[(' is not', 5378),

(' did not', 4687),

(' are not', 3480),

(' is also', 3396),

(' is now', 2896),

(" do n't", 2878),

(' has not', 2751),

(' has also', 2570),

(' are still', 2548),

(' are now', 2373),

(' was not', 2303),

(' are also', 2261),

(' do not', 2165),

(' have also', 2144),

(' does not', 1825),

(' is still', 1692),

(' was also', 1657),

(' was first', 1637),

(' was up', 1570),

(' have not', 1509),

(' ’ s', 1464),

(' is currently', 1349),

(' were also', 1314),

(" did n't", 1293),

(' are currently', 1215),

(' have already', 1210),

(" 's not", 1171),

(' had not', 1170),

(' Read more', 1080),

(' has already', 1033),

(' were not', 1011),

(" does n't", 953),

(' was down', 831),

(' ’ ve', 811),

(' have now', 789),

(' are already', 776),

(' had already', 775),

(' has now', 760),

(' were up', 739),

(' is already', 736),

(' ’ m', 728),

(' was originally', 697),

(' is just', 641),

(" is n't", 611),

(' s not', 598),

(' is only', 569),

(' do so', 563),

(' had recently', 545),

(' come back', 526),

(" have n't", 524)]

Neg adv phrase use a lot of negation and past tense. Also, interesting, ‘come back’ is informative that it represent people’s fear of virus

50 most frequent negative verb phrase are:

[(' miss a story', 155),

(' Related News', 129),

(' Related news', 70),

(' mailing list Enter Your Email Address', 56),

(' Leave a Reply', 51),

(' improves Markets', 46),

(' powered by TradingView.com Retirement Intelligence', 39),

(' Related Content', 37),

(' Related Tags', 31),

(' s PPK', 31),

(' Join the Conversation', 27),

(' Join the conversation', 22),

(' //twitter.com/AP\_Sports The Associated Press', 19),

(' Sourced from Pixabay', 19),

(' post a comment Login photo gallery', 17),

(' download CCTV.com Global app', 16),

(' Suggested Articles', 16),

(' Related News ///', 16),

(' Read an Analyst Note', 16),

(' Related content', 16),

(' > >', 15),

(' Suggested Event', 14),

(' provided by StockMarketWire.com', 14),

(' protected ]', 13),

(' skip all comments', 12),

(' existing subscription', 12),

(' targeting > >', 11),

(' Related videos', 10),

(' @ dtn.com', 10),

(' Powered by PressPatron', 10),

(' | Privacy Policy Secondary navigation', 9),

(" DO N'T MISS", 9),

(' commenting as Logout', 9),

(' Trending in Travel', 9),

(' Translated by Uyen Phuong', 9),

(' Related Post', 9),

(' Related Articles', 9),

(' Report a Typo', 8),

(' Leave a comment', 8),

(' Edited by MUF', 7),

(' continues below advertisement', 7),

(' Related Story', 7),

(' Updated Privacy Policy', 7),

(' Sourced from scoop.co.nz', 7),

(' Sounds From MetLife Stadium', 7),

(' Translated by Ngoc Huynh Share', 7),

(' @ barrons.com', 6),

(' am Share this article', 6),

(' Edited by TSB', 6),

(' Read this article on OilPrice.com', 6)]

Negative verb phrase are all describing news, because there are a lot of bad news in pandemic times..

Then, I convert my dict into a valid dict and write it into a csv:

dict1.keys()  
dict2={'author':dict1['author'], 'facebook':dict1['facebook'],'title':dict1['title']  
,'published':dict1['published'], 'url':dict1['url'],'replies\_count':dict1['replies\_count'], 'country':dict1['country'], 'text':dict1['text']  
,'negcount':dict1['negcount'], 'poscount':dict1['poscount'], 'neucount':dict1['neucount'] }  
#create valid dict  
import csv  
f = open('dict2.csv','w', encoding="utf-8")  
w = csv.DictWriter(f,dict2.keys())  
w.writeheader()  
w.writerow(dict2)  
f.close()  
#write dict into csv