Spring 2023

SENTIMENTAL VALUE: Developing a model for sentiment & linguistic analysis of cable news

Caitlin Bannister

Sentiment & Linguistic Analysis Script

HONORS CAPSTONE by Caitlin Bannister

First load packages

Otherwise nothing will work

In [ ]:
```
from newspaper import Article
import nltk
from textblob import TextBlob
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import pandas as pd
import os
from nrclex import NRCLex as nrc
import re
from scipy.stats import f_oneway
import numpy as np
```

Define variables to be used for the script

NOTE: Windows users will need to precede file paths with the letter 'r'; i.e. r'C:/Users/path/to/file' whereas a linux user would simply enter 'C:/Users/path/to/file'

In [ ]:
```
#Transcript_path should be the folder that all transcripts are in
transcript_path = r'C:\path\to\transcripts\folder'

#This is the location of the moral foundations csv file
moralterm_file = r'C:\path\to\moral\foundations\file.csv'

#Output_path should be the location where you'd like to have the output saved
output_path = r'C:\path\to\output\storage'

#The searchterm_file should be a .csv file with a single column named 'Topics' followed by all the terms of interest
#be sure that the terms of interest are written EXACTLY as they should be search
searchterm_file = r'C:\path\to\search\term\file.csv'

#Filesave should be the name of the file you want your analysis output to be stored in
#Be sure that you DO NOT include the '.csv' at the end
sentiment_output_file = 'sentiment_output_file_name'
```

Import Moral Foundations Dictionary

This will create a list of the words for each category of Moral Foundations to be analyzed

Users can utilize the attached .csv file of the Moral Foundations dictionary sourced from https://moralfoundations.org/wp-content/uploads/files/downloads/moral%20foundations%20dictionary.dic

In [ ]:
```
#create empty lists that will be appended below
HarmVirtuelist = []
HarmVicelist = []
FairnessVicelist = []
FairnessVirtuelist = []
IngroupVicelist = []
IngroupVirtuelist = []
AuthorityVicelist = []
AuthorityVirtuelist = []
PurityVicelist = []
PurityVirtuelist = []
MoralityGenerallist = []

#read the Moral Foundations dictionary file
moralterms = pd.read_csv(moralterm_file)

#Define a function to place each term into the appropriate list
def mffunction(colname):
    if row.loc[colname]==1:
```

```
        HarmVirtuelist.append(termadd)
    elif row.loc[colname]==2:
        HarmVicelist.append(termadd)
    elif row.loc[colname]==3:
        FairnessVirtuelist.append(termadd)
    elif row.loc[colname]==4:
        FairnessVicelist.append(termadd)
    elif row.loc[colname]==5:
        IngroupVirtuelist.append(termadd)
    elif row.loc[colname]==6:
        IngroupVicelist.append(termadd)
    elif row.loc[colname]==7:
        AuthorityVirtuelist.append(termadd)
    elif row.loc[colname]==8:
        AuthorityVicelist.append(termadd)
    elif row.loc[colname]==9:
        PurityVirtuelist.append(termadd)
    elif row.loc[colname]==10:
        PurityVicelist.append(termadd)
    elif row.loc[colname]==11:
        MoralityGenerallist.append(termadd)

for index,row in moralterms.iterrows():
    #alter each term to contain the appropriate RegEx operator rather than a "*"
    if '*' in row.loc['Topic']:
        term = row.loc['Topic'].split('*')
        row.loc['Topic'] = f"{term[0]}.+"
    #if no "*", make no alteration
    else:
        row.loc['Topic'] = {row.loc['Topic']}

    #convert the term to a string
    termadd = str(row.loc['Topic'])

    #run previously defined function over all categories
    mffunction("Value_Category1")
    mffunction("Value_Category2")
    mffunction("Value_Category3")
```

Read Search Term Input

OPTIONAL: filter & remove certain parts of speech from the searchlist

This is helpful if the search list contains words that you do not want to be analyzed (i.e. "the", "is", "it"). If this function is not of interest, the for loop below can be edited to remove that function

NOTE: Additional TextBlob part-of-speech "tags" exist, allowing for other parts of speech tags to be filtered as well, this is not an all inclusive filtering. Refer to TextBlob documentation for more information.

In [ ]:
```
#read searchterms file defined above
searchterms = pd.read_csv(searchterm_file)

#create empty list to place final search list, will append below
searchlist = searchterms.Topics.tolist()

'''
IMPORTANT:
If your terms are not written EXACTLY the way you intend them to be search
you will need to write additional code to manipulate your list here
'''
```

Import transcript data

This will be based on the variables that were established above, make sure they are correct before running this.

This block of code is set up to address multiple folder hierarchies where the the transcripts are stored at 'transcript_path/author_folder/month_folder/transcript.txt'; this will need to be adjusted if that is not how the files are stored

In [ ]:
```
#create empty lists to fill below
author_folder_list = []
month_folder_list = []
transcript_list = []
```

```
#go through every folder in the transcript path
#if there are not multiple folders in the transcript path, remove this part of the code
for author_folder in os.listdir(transcript_path):
    author_folder = os.path.join(transcript_path, author_folder)
    author_folder_list.append(author_folder)

#go through every month_folder in the author_folder
for author_folder_path in author_folder_list:
    month_folders = os.listdir(author_folder_path)
    for month_folder in month_folders:
        month_folder = os.path.join(author_folder_path, month_folder)
        month_folder_list.append(month_folder)

#add each transcript in each folder to the transcript_list
for month_folder in month_folder_list:
    file_list = os.listdir(month_folder)
    for fn in file_list:
        if fn.endswith('.txt'):
            fn = os.path.join(month_folder, fn)
            transcript_list.append(fn)
```

Run Sentiment Analysis

This script will capture all sentences that contian the search terms of interest. For each sentence it will calculate:

- Subjectivity of the sentence (TextBlob, 0 to 1, objective to subjective)
- Polarity of the sentence (TextBlob, -1 to +1, negative to positive)
- Negative/Neutral/Positive of the sentence (Vader Sentiment, these represent the portion of the sentence that is negative, neutral, or positive, all three should add up to 1)
- Compound of the sentence (Vader Sentiment, -1 to +1, negative to positive)
- Number of "emotion" words in the sentence (NRC Emotion Lexicon; ten categories: negative, anticipation, joy, positive, trust, sad, anger, disgust, fear, and surprise)
- Number of "Moral Foundations" words in the sentence (Dictionary from moralfoundations.org; measures across eleven categories: harm vice/virtue, fairness vice/virtue, ingroup vice/virtue, authority vice/virtue, purity vice/virtue, and "morality general")

This script assumes that files are written in the format of "author_month_date_year.txt"; if files are not named this way, the script will need to be edited to accommodate that

This script is set up to only search for EXACT matches of the search terms. If it is desired to search for non-exact terms (i.e. looking for "kill+" would be able to capture alternate word suffixes such as "kills", "killing", "killer"), consult RegEx documentation for Python to adjust the code.

NOTE: See the docstring below; many transcript files contain formatting that is not necessary for analysis. If the files contain information that should not be analyzed, you must replace the docstring with appropriate code to remove that extra information OR remove that information from the files before running this cell.

In [ ]:
```
#import sentiment analyser from vader sentiment
analyzer = SentimentIntensityAnalyzer()

#set up dataframe for output
df2 = pd.DataFrame(columns = ['Date', 'Author', 'Term Flag', 'Text', 'Subjectivity', 'Polarity', 'Negative',
                              'Neutral', 'Positive', 'Compound', 'Negative Words', 'Anticipation Words',
                              'Joy Words', 'Positive Words', 'Trust Words', 'Sad Words', 'Anger Words',
                              'Disgust Words', 'Fear Words', 'Surprise Words', 'HarmVirtue', 'HarmVice',
                              'FairnessVirtue', 'FairnessVice', 'IngroupVirtue', 'IngroupVice',
                              'AuthorityVirtue', 'AuthorityVice', 'PurityVirtue', 'PurityVice', 'MoralityGeneral'])

#go through each transcript individually
for fn in transcript_list:
    #read each transcript file and separate into individual lines
    text = open(fn, 'r')
    text = text.readlines()

    #establish the data, month, and author of each file
    #IMPORTANT: make sure the naming system of the transcripts works with this block of code!
    basefn = os.path.basename(fn)
    basefn = basefn.split('.')
    basefn = basefn[0]
    basefn = basefn.split('_')
```

```
date = f"{basefn[1]}_{basefn[2]}_{basefn[3]}"
monthyear = f"{basefn[1]}_{basefn[3]}"
author = basefn[0]

'''
IMPORTANT:
If your transcripts contain extra lines of formatting or
information you do not want to run the analysis on,
you will need to insert a set of code HERE that will remove
the extraneous details
'''

#convert the transcript into a dataframe
df = pd.DataFrame(text, columns = ['Line'])

#create an output dataframe for flagged sentences that contain term of interest
df1 = pd.DataFrame(columns = ['Term', 'Sentence'])

#for every line in the transcript, look for every term in the search list
#if the line contains a term(s) from the search list, it will be added to df1
for index,row in df.iterrows():
    line = row.loc['Line']
    linewords = line.split(' ')
    for term in searchlist:
        result = linewords.count(term)
        if result > 0:
            df1.loc[len(df1.index)] = [term, line]
        else:
            continue
#drop any duplicates that were added
df1 = df1.drop_duplicates()

#run sentiment analysis on each flagged sentence
for index,row in df1.iterrows():
    #some formatting changes to make it compatible with the analysis
    line = row.loc['Sentence']
    linewords = line.split(' ')
    line = TextBlob(line)
    termword = row.loc['Term']

    #TextBlob sentiment
    tbsentiment = line.sentiment
    polarity = tbsentiment.polarity
    subjectivity = tbsentiment.subjectivity

    #VaderSentiment sentiment
    vssentiment = analyzer.polarity_scores(line)
    negative = vssentiment['neg']
    neutral = vssentiment['neu']
    positive = vssentiment['pos']
    compound = vssentiment['compound']

    #convert to string for compatibility
    sentstr = str(line)

    #run emotion analysis and appropriate compatibility formatting
    emotion = nrc(sentstr)
    emotions = emotion.raw_emotion_scores
    emotions = str(emotions)
    emotions = emotions.split('{')
    emotions = emotions[1]
    emotions = emotions.split('}')
    emotions = emotions[0]
    emotions = emotions.split(',')
    emotionnamelist = ["negative", "anticipation", "joy", "trust", "positive",
                       "sadness", "anger", "disgust", "fear", "surprise"]
    for emotion in emotions:
        if len(emotion) == 0:
            continue
        else:
            emotionname = emotion.split("'")
            emotionname = emotionname[1]
            emotion = emotion.split(':')
            emotion = emotion[1]
        if emotionname == "negative":
            negnum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "anticipation":
            antnum = emotion
```

```
            emotionnamelist.remove(emotionname)
        elif emotionname == "joy":
            joynum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "positive":
            posnum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "trust":
            trustnum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "sadness":
            sadnum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "anger":
            angnum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "disgust":
            disnum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "fear":
            fearnum = emotion
            emotionnamelist.remove(emotionname)
        elif emotionname == "surprise":
            surnum = emotion
            emotionnamelist.remove(emotionname)
        else:
            print(f"Error: emotion name not included, emotion is {emotionname}")
    for emotionname in emotionnamelist:
        if emotionname == "negative":
            negnum = 0
        elif emotionname == "anticipation":
            antnum = 0
        elif emotionname == "joy":
            joynum = 0
        elif emotionname == "positive":
            posnum = 0
        elif emotionname == "trust":
            trustnum = 0
        elif emotionname == "sadness":
            sadnum = 0
        elif emotionname == "anger":
            angnum = 0
        elif emotionname == "disgust":
            disnum = 0
        elif emotionname == "fear":
            fearnum = 0
        elif emotionname == "surprise":
            surnum = 0

    #run moral foundations analysis and appropriate compatability formatting
    mgcount = []
    apcount = []
    ancount = []
    incount = []
    ipcount = []
    hpcount = []
    hncount = []
    fncount = []
    fpcount = []
    ppcount = []
    pncount = []

    for word in linewords:
        for term in MoralityGenerallist:
            if re.search(term, word):
                mgcount.append(term)
        for term in FairnessVirtuelist:
            if re.search(term,word):
                fpcount.append(term)
        for term in FairnessVicelist:
            if re.search(term,word):
                fncount.append(term)
        for term in IngroupVirtuelist:
            if re.search(term,word):
                ipcount.append(term)
        for term in IngroupVicelist:
            if re.search(term,word):
                incount.append(term)
        for term in AuthorityVirtuelist:
```

```
                if re.search(term,word):
                    apcount.append(term)
            for term in AuthorityVicelist:
                if re.search(term,word):
                    ancount.append(term)
            for term in PurityVirtuelist:
                if re.search(term,word):
                    ppcount.append(term)
            for term in PurityVicelist:
                if re.search(term,word):
                    pncount.append(term)
            for term in HarmVirtuelist:
                if re.search(term,word):
                    hpcount.append(term)
            for term in HarmVicelist:
                if re.search(term,word):
                    hncount.append(term)
        mg = len(mgcount)
        ap = len(apcount)
        an = len(ancount)
        ing = len(incount)
        ip = len(ipcount)
        hp = len(hpcount)
        hn = len(hncount)
        fn = len(fncount)
        fp = len(fpcount)
        pp = len(ppcount)
        pn = len(pncount)

        #add final output to df2
        df2.loc[len(df2.index)] = [date, author, termword, sentstr, subjectivity, polarity, negative,
                                   neutral, positive, compound, negnum, antnum,
                                   joynum, posnum, trustnum, sadnum, angnum, disnum,
                                   fearnum, surnum, hp, hn, fp, fn, ip, ing, ap, an, pp, pn, mg]

#save output to a file
#this will contain all analyzed data amongst all authors, transcripts, and captured terms
filesave = f"{sentiment_output_file}.csv"
df2.to_csv(os.path.join(output_path, filesave))
```

Statistical Analysis

This statistical analysis: 1) only uses subjectivity and compound out of all of the functions defined in the previous cell, can add more 2) categorizes a subset of terms from the original analysis (in this case, Ukraine, Russia, Democrats, Republicans, Biden, Trump, and Big Tech)

Define input and output variables here (NOTE: Windows users will need to precede file paths with the letter 'r'; i.e. r'C:/Users/path/to/file' whereas a linux user would simply enter 'C:/Users/path/to/file')

In [ ]:
```
#the input_file should be your sentiment analysis data
#if you just ran the analysis and have not erased your defined variables, you can define your input file as:
#input_file = os.path.join(output_path, filesave)
input_file = r'C:/path/to/file.csv'

#the output_path should be where you want your stats analysis to be saved
output_path = r'C:/path/to/save/output'

#the output_file should be whatever you'd like to name your stats analysis file
#be sure that you DO NOT include the '.csv' at the end
stats_output_file = 'define_output_file_name_here'
```

ANOVA

This will produce final output for each term analyzed including:

- ANOVA of all three authors
- ANOVA of each author pair (Carlson vs. Cooper, Cooper vs. Odonnell, Odonnell vs. Carlson)
- Count of term mentions across all authors
- Count of term mentions per author
- Average score of subjectivity and compound across all authors
- Average score of subjectivity and compound across each author

NOTE: Although this defines separate categories (Ukraine, Russia, Democrats, Republicans, Biden, Trump, and Big Tech) within the script, the output does not retain those categories in the output; output will be based on the original term only and can be recategorized from there

In [ ]:
```
#define a function for ANOVA analysis
def anova(anovaname, input1, input2, input3=None)
    anovaname = f_oneway(input1, input2, input3)
    anovaname = str(anovaname)
    anovaname = anovaname.split('=')
    anovaname = anovaname[-1]
    anovaname = anovaname.split(')')
    anovaname = anovaname[0]

#read input file + reformat the dataframe
df = pd.read_csv(input_file)
df = df.drop(['Unnamed: 0'], axis=1)

#generate a list of columns for analysis
#add additional columns to this list if there are more that are of interest
collist = ['Subjectivity', 'Compound']

#create a dataframe for the output
#adjust the output columns if necessary
dfsummary = pd.DataFrame(columns = ['Term', 'TermCount', 'ExperCond', 'Avg',
                                    'AllAnova', 'CoopCarlAnova', 'CarlOdonAnova',
                                    'OdonCoopAnova', 'CooperAvg', 'CooperCount',
                                    'CarlsonAvg', 'CarlsonCount', 'OdonnellAvg',
                                    'OdonnellCount'])

#generate dataframe slices for categories of interest
#must match exactly to the 'Term Flag' in input file
#ukraine terms
ukrslice = df.loc[(df['Term Flag'] == 'ukraine') | (df['Term Flag'] == 'ukrainian') |
                  (df['Term Flag'] == 'ukrainians') | (df['Term Flag'] == 'kyiv') |
                  (df['Term Flag'] == 'zelensky')]
#russia terms
russlice = df.loc[(df['Term Flag'] == 'russia') | (df['Term Flag'] == 'russian') |
                  (df['Term Flag'] == 'russians') | (df['Term Flag'] == 'moscow') |
                  (df['Term Flag'] == 'putin')]
#democrat terms
demslice = df.loc[(df['Term Flag'] == 'pelosi') | (df['Term Flag'] == 'schumer') |
                  (df['Term Flag'] == 'harris') | (df['Term Flag'] == 'aoc') |
                  (df['Term Flag'] == 'cortez')]
#republican terms
repslice = df.loc[(df['Term Flag'] == 'cruz') | (df['Term Flag'] == 'mcconnell') |
                  (df['Term Flag'] == 'pence') | (df['Term Flag'] == 'mccarthy')]
#biden term
bidenslice = df.loc[df['Term Flag'] == 'biden']
#trump term
trumpslice = df.loc[df['Term Flag'] == 'trump']
#big tech terms
bigtechslice = df.loc[(df['Term Flag'] == 'facebook') | (df['Term Flag'] == 'meta') |
                  (df['Term Flag'] == 'microsoft') | (df['Term Flag'] == 'musk') |
                  (df['Term Flag'] == 'twitter') | (df['Term Flag'] == 'zuckerberg') |
                  (df['Term Flag'] == 'google') | (df['Term Flag'] == 'amazon') |
                  (df['Term Flag'] == 'apple') | (df['Term Flag'] == 'bezos')]

#create a list of slice dictionaries in order to define category names
#allows for 'for loop' iteration over each list
sliceslist = {'Ukraine': ukrslice, 'Russia': russlice, 'Democrats': demslice,
              'Republicans': repslice, 'Biden': bidenslice, 'Trump': trumpslice,
              'BigTech': bigtechslice}

#perform stats analysis on each term in each list
#term is the category name i.e. 'Ukraine' or 'Russia'
#dfslice is the slices defined above
for term, dfslice in sliceslist.items():
    #set up total count of the term in all authors
    termcount = len(dfslice)
    #create a slice for Cooper use of the term
    coopslice = dfslice.loc[dfslice['Author'] == 'Cooper']
    #set up total count of the term in Cooper only
    coopcount = len(coopslice)
    #create a slice for Carlson use
    carlslice = dfslice.loc[dfslice['Author'] == 'Carlson']
    #set up total count of the term in Carlson only
    carlcount = len(carlslice)
```

```
#create a slice for Odonnell use
odonslice = dfslice.loc[dfslice['Author'] == 'Odonnell']
#set up total count of the term in Odonnell only
odoncount = len(odonslice)

#run stats for each column defined in the collist above
for column in collist:
    #run scenario #1 where one of the authors has zero mentions
    #this will define the appropriate stats as "invalid" if there is a zero
    if coopcount == 0 or carlcount == 0 or odoncount == 0:
        #run scenario #1.a where there is a zero only in Cooper
        if coopcount == 0 and carlcount > 0 and odoncount > 0:
            carlslice1 = carlslice[column].to_numpy()
            odonslice1 = odonslice[column].to_numpy()
            threeanova = "invalid"
            ccanova = "invalid"
            coanova = "invalid"
            anova(ocanova, carlslice1, odonslice1)
            coldat = dfslice[column].mean()
            colcoop = "invalid"
            colcarl = carlslice[column].mean()
            colodon = odonslice[column].mean()
            dfsummary.loc[len(dfsummary.index)] = [term, termcount, column, coldat,
                    threeanova, ccanova, ocanova,
                    coanova, colcoop, coopcount, colcarl,
                    carlcount, colodon, odoncount]
        #run scenario #1.b where there is a zero only in Carlson
        elif carlcount == 0 and coopcount > 0 and odoncount > 0:
            coopslice1 = coopslice[column].to_numpy()
            odonslice1 = odonslice[column].to_numpy()
            threeanova = "invalid"
            ccanova = "invalid"
            ocanova = "invalid"
            anova(coanova, coopslice1, odonslice1)
            coldat = dfslice[column].mean()
            colcoop = coopslice[column].mean()
            colcarl = "invalid"
            colodon = odonslice[column].mean()
            dfsummary.loc[len(dfsummary.index)] = [term, termcount, column, coldat,
                    threeanova, ccanova, ocanova,
                    coanova, colcoop, coopcount, colcarl,
                    carlcount, colodon, odoncount]
        #run scenario #1.c where there is a zero only in Odonnell
        elif odoncount == 0 and carlcount > 0 and coopcount > 0:
            coopslice1 = coopslice[column].to_numpy()
            carlslice1 = carlslice[column].to_numpy()
            threeanova = "invalid"
            ocanova = "invalid"
            coanova = "invalid"
            anova(ccanova, coopslice1, carlslice1)
            coldat = dfslice[column].mean()
            colcoop = coopslice[column].mean()
            colcarl = carlslice[column].mean()
            colodon = "invalid"
            dfsummary.loc[len(dfsummary.index)] = [term, termcount, column, coldat,
                    threeanova, ccanova, ocanova,
                    coanova, colcoop, coopcount, colcarl,
                    carlcount, colodon, odoncount]
        #run scenario #1.d where there is a zero in two or more authors
        else:
            print(f"carl: {carlcount}, odon: {odoncount}, coop: {coopcount}")
            threeanova = "invalid"
            ccanova = "invalid"
            coanova = "invalid"
            ocanova = "invalid"
            coldat = "invalid"
            colcoop = "invalid"
            colcarl = "invalid"
            colodon = "invalid"
            dfsummary.loc[len(dfsummary.index)] = [term, termcount, column, coldat,
                            threeanova, ccanova, ocanova,
                            coanova, colcoop, coopcount, colcarl,
                            carlcount, colodon, odoncount]
        continue
    #run scenario #2 where all authors have mentioned the term
    elif coopcount > 0 and carlcount > 0 and odoncount > 0:
        coopslice1 = coopslice[column].to_numpy()
        carlslice1 = carlslice[column].to_numpy()
        odonslice1 = odonslice[column].to_numpy()
```

```
            anova(threeanova, coopslice1, carlslice1, odonslice1)
            anova(ccanova, coopslice1, carlslice1)
            anova(coanova, coopslice1, odonslice1)
            anova(ocanova, carlslice1, odonslice1)
            coldat = dfslice[column].mean()
            colcoop = coopslice[column].mean()
            colcarl = carlslice[column].mean()
            colodon = odonslice[column].mean()
            dfsummary.loc[len(dfsummary.index)] = [term, termcount, column, coldat,
                        threeanova, ccanova, ocanova,
                        coanova, colcoop, coopcount, colcarl,
                        carlcount, colodon, odoncount]
        #run scenario #3 where something has gone wrong in data
        #this will print an error statement and break the loop
        #go back and fix the data if this occurs
        else:
            print(f"ERROR: there's a zero in {term}")
            break
dfsummary = dfsummary.drop_duplicates()
filesave = f"{stats_output_file}.csv"
dfsummary.to_csv(os.path.join(output_path, filesave))
```