From Public to Individual:
How Others' Sentiment Influences Yours?

Huaman Sun & Junhui Qian

12/16/2021

Introduction

In the digital era, social media such as Twitter and Facebook, have gradually become an invaluable platform for online social interactions (Parslow, 2011). This phenomenon enables social media to influence people's attitudes in different ways.

Social media are capable of guiding individuals' attitudes and feelings. Firstly, social media context has a certain degree of blindness (Forelle et al., 2015). Some users just blindly follow and support opinion leaders, thereby posting new content that further exacerbates the trend of public opinion, which may influence more individuals. Besides, information overload makes social media users more vulnerable to manipulations of their attitudes by occupying most of their cognitive resources (Ferrari, 2010). Without sufficient cognitive resources, it will become more difficult for individuals to search for evidence in memory to strengthen the original attitude. And since people are unable to focus on the logic and evidence of the provided information, they are more prone to be influenced (Lee et al., 2016; Niu et al., 2020).

On the other hand, even with the high volume of social media information, it is not an easy task to persuade individuals to change their minds. People tend to maintain a stable attitude rather than to shift it occasionally. They use different strategies, such as attitude bolstering, counter-arguing, negative affect, selective exposure, social validation, and source derogation, to resist attitude change (Jacks & Cameron, 2003). In addition, unlike authoritative mass media, anyone can publish a post on social media at any time (Hayes & King, 2014). It possibly causes people not to trust social media as much as mainstream media, which also leads social media users to be less sensitive to the public trends of attitudes in social media context.

Existing literature does not have much empirical evidence on how public attitudes in social media context may influence individual attitudes correspondingly. The only supporting evidence of this effect we found is that the ratio between positive and negative tweets, which represents the public attitudes, will significantly contribute to the prediction of future sentiment trends (Nguyen et al., 2021).

Therefore, the present study is aimed to examine the influence of public attitudes on individual attitudes in the social media context. Specifically, we utilize tweets on the topic of the omicron variant to test whether the two dimensions of the sentiment of individual tweets (i.e., subjectivity and polarity) change with thier public counterparts.

Data

Data Collection

Twitter Data

The data of tweets was collected via Search Tweets endpoint, Twitter API v2 (https://api.twitter.com/2/tweets/search/recent). The present study collected a sample of tweets that were on the topic of the omicron variant during 10 am., November 27 to 04 am., December 4 (in the UTC timezone).

Considering the reverse-chronological order feature of the Search Tweets endpoint, we set 2 hours as a time unit, then retrieved 1000 tweets from each time unit. In this way, we were able to maximize the uniformity of the data distribution over time. In each time unit, the query was restricted to tweets that were generated with a hashtag of #Omicron and in English. Retweets were excluded from the request, as retweets usually didn't contain sufficient information for sentiment analysis of those who retweeted them.

```python
"""
    Note:
        With current authoritation, the maximum of tweets returned per response is 100.
        In order to retrieve tweets more than that,
        looking for and adding next_token to the query repeatedly is needed here.
"""

# Loop all time units
for i in range(0, len(start_times)):

    # Initialize settings for each time unit
    query_params = {
        'query': '#Omicron (lang:en) -is:retweet',
        'max_results': '100',
        'start_time': None,
        'end_time': None
    }
    next_token = None
    text_data = pd.DataFrame(columns=['id', 'text', 'start_time'])

    # Iterate the request by pagination (next_token)
    for j in range(0, 10):
        query_params.update({
            'start_time': start_times[i],
            'end_time': end_times[i]
        })

        # Update the query; Break the loop if there is no next_token any more
        if j:
            if next_token:
                query_params.update({'next_token':next_token})
            else:
                break

        # Request tweets
        json_response = connect_to_endpoint(ENDPOINT, query_params)

        df = pd.DataFrame(json_response['data'])
        df['start_time'] = start_times[i]
        text_data = pd.concat([text_data, df])

        # Look for next_token
        if 'next_token' in json_response['meta'].keys():
            next_token = json_response['meta']['next_token']
        else:
            next_token = None
```

Covariates

In order to control the effect of the pandemic growth, we included the global number of new cases and the global number of new deaths as covariates in our model. The data was obtained from a public Covid-19 database by Our World in Data, the University of Oxford.

Data Processing

Resampling

Considering the high volume of our original dataset and the time of computation, we resampled 50% data for each time unit.

```
def text_sample_join(folderpath, frac, df):
    """
    Resample all datasets in the folder with a given fraction,
    Join them into one dataset

    Parameters:
        folderpath (str): the path of the folder holding all datasets
        frac (float): resampling fraction
        df (DataFrame): the dataset to join all the subsets into

    Return:
        df (DataFrame): resampled and joined dataset
    """

    # Traverse all files in the folder
    for file in os.listdir(folderpath):

        # Generate the file path
        filepath = os.path.join(folderpath, file)

        # Resample and join to df if the path is assigned a file
        if os.path.isfile(filepath):
            text_data = pd.read_csv(filepath, lineterminator='\n')
            text_sample = text_data.sample(frac=frac, random_state=1216)
            df = pd.concat([df, text_sample])

    return df
```

Data Cleaning

In order to obtain more accurate estimates of sentiment scores, data cleaning was needed before sentiment analysis. All @(Reply), #(Hashtag), and URL were removed from the original text, since they were not able to be recognized and scored with the Vader lexicon but would be counted in the sentiment analysis, therefore shrinking the weight of sentiment words. By removing these strings, an underestimation to the Vader scores was avoided.

```
def data_cleaning(text):
    """
    Regular Expression is used to remove all @(Reply), #(Hashtag), and URL
    Parameters:
       text (str): original text

    Return:
        text_cleaned (str): cleaned version of the original text
    """

    # Remove @(Reply)
    text_1 = re.sub(r'@[^\s]*', "", text)
    # Remove #(Hashtag)
    text_3 = re.sub(r'#[^\s]*', "", text_1)
    # Remove URL
    text_cleaned = re.sub(r'[a-zA-Z]+://[^\s]*', "", text_3)

    return text_cleaned
```

Sentiment Analysis

In sentiment analysis, we calculated subjectivity and polarity with the library textblob and nltk respectively.

Both of the two libraries are based on lexicon-based approaches, which requires a pre-defined dictionary. A sentiment measure is defined by the semantic orientation and the intensity of each word in the sentence. Generally, an input text will be represented by bag of words. After assigning individual scores to all the words, the final sentiment is calculated by some pooling operation such as weighted average.

Textblob is an efficient library for sentiment analysis, but its lexicon is not capable of recognizing many of the emojis in social media data. It may result in a less accurate estimation, especially for polarity analysis. Therefore, we choose Vader for the assessment of polarity, which is optimized for social media data such as twitter and facebook.

Subjectivity In the present study, subjectivity was calculated by the library textblob. The score quantifies the amount of personal opinion and factual information contained in the text. It lies between [0,1], where 0 represents "very objective" and 1 represents "very subjective". A higher subjectivity score means that the text contains more personal opinion rather than factual information.

```
def get_subjectivity(text):
    """
    Parameters:
        text (str): source data

    Return:
        subjectivity_score (float)
    """

    return TextBlob(text).sentiment.subjectivity
```

Polarity The Vader assessment from the library nltk of polarity consists of compound score, positive score, negative score and neutral score. Compound score measures the general polarity of the text, which varies from -1 to 1. -1 defines as a totally negative sentiment and 1 defines as a totally positive sentiment. A compound score around 0 means that the input text is with neutral sentiment. The other three score estimate the probability of a given input to be positive, negative, or neutral respectively. These three probabilities will add up to 1.

```python
def get_vader_score(text):
    """
    Caluculate the average vader assessment scores for the input text

    Parameters:
        text (str): source data

    Return:
        vader_scores (dict): a dict contains compound score, negitive score,
            neutral score, positive score.
    """

    # Initialize settings
    sentences = tokenize.sent_tokenize(text)
    vader_scores = {
        'compound': [],
        'neg': [],
        'neu': [],
        'pos': [],
    }
    sid = SentimentIntensityAnalyzer()

    # Calculate Vader scores for each individual sentence
    for sentence in sentences:
        polarity = sid.polarity_scores(sentence)
        for key in polarity.keys():
            vader_scores[key].append(polarity[key])

    # Calculate the average Vader scores for the input text
    for key, val in vader_scores.items():
        vader_scores[key] = np.nanmean(val)

    return vader_scores
```

Data Formatting

After all the data processing steps, a final csv file was generated for statistical analysis. This csv file was formatted to make it easily read and use in R.

For individual level data, all redundant variables were removed. Cases with any missing value and/or failing sentiment analysis were dropped as well.

Since the present study is aimed to analysis the influence of public sentiment on individual's sentiment, we also include some public level variables. Target variables were grouped by time unit, then the mean of those variables were assigned to variables naming mean_*.

In the last step, covariates were added to the dataset.

The final dataset contains 9 variables and 34,348 observations.

```r
individual <-
  read.csv2("tweet_sentiment.csv", sep=",", row.names=NULL) %>%
  select(start_time, subjectivity, compound) %>%
  mutate(start_time = as_datetime(start_time)) %>%
  mutate(date = date(start_time)) %>%
  mutate_at(vars(subjectivity, compound), as.numeric) %>%
  na.omit() %>%
  filter(compound != 0 | subjectivity != 0)

public <-
  individual %>%
  group_by(start_time) %>%
  summarise(mean_sub = mean(subjectivity),
            mean_com = mean(compound),
            total = n())

ind_pub <- left_join(individual, public,
                     by = c("start_time" = "start_time"))

covar <-
  read.csv2("covars.csv", sep=",", row.names=1) %>%
  mutate(date = date(date)) %>%
  mutate_at(vars(case_increase, death_increase), as.numeric)

all_set <- left_join(ind_pub, covar,
                     by = c("date" = "date"))

write.csv2(all_set, file = "omicron_processed.csv")
```

Statistical Analysis

Multiple regression is used to estimate the effects of public sentiment on the individual sentiment.

We first examined the effects of public sentiment in a subset of the data. Individual subjectivity and individual polarity were assigned as the dependent variables separately to fit a base model, which only used public subjectivity and public polarity as the independent variables. We then included the covariates into the models, so that we could control the influence of the pandemic growth.

In order to obtain a more robust result, we also examined the effects of public sentiment by bootstrap resmapling.

```r
omicron <- read.csv2("omicron_processed.csv", row.names = 1)

omicron_sub <-
  omicron %>%
  sample_n(size=2000)
```

```r
# Multiple Regressions in the Sub Dataset (base models)
reg_sub <- lm(subjectivity ~ mean_sub + mean_com, data = omicron_sub)

reg_com <- lm(compound ~ mean_sub + mean_com, data = omicron_sub)

# Multiple Regressions in the Sub Dataset (models with covariates)
reg_sub_c <- lm(subjectivity ~ mean_sub + mean_com + case_increase + death_increase, data = omicron_sub

reg_com_c <- lm(compound ~ mean_sub + mean_com + case_increase + death_increase, data = omicron_sub)
```

```r
betas <-
  function(formula, data, indices){
    d <- data[indices,]
    fit <- lm(formula, d)
    return(fit$coef)
  }

boot_sub <- boot(data=omicron, statistic=betas, R=300,
                    formula=subjectivity ~ mean_sub + mean_com)
boot_com <- boot(data=omicron, statistic=betas, R=300,
                    formula=compound ~ mean_sub + mean_com)
```

Results

Table 1 shows the multiple regression results of the four models in a sub dataset.

The first model indicats that public subjectivity has a significantly positive influence on individual subjectivity, while public polarity does not have a significant impact on individual subjectivity.

As the results of the second model show, individual polarity is significantly influenced by public polarity. The higher the public polarity is, the more likely individual polarity to be positive correspondingly. Public subjectivity is also a significant predictor of individual polarity, which is negatively related with individual polarity. This finding is different from the first model.

In the third and fourth model, we could see that covariates have few effect on individual subjectivity and polarity. Controlling the impact of the pandemic growth, the relationships among public and individual sentiment are still held.

The results of Bootstrap show a very slight bias of our estimations, which suggests that our findings is robust.

```r
print(boot_sub)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = omicron, statistic = betas, R = 300, formula = subjectivity ~
##     mean_sub + mean_com)
##
##
## Bootstrap Statistics :
##         original        bias    std. error
## t1* 2.269662e-14 -0.0012693865  0.02711650
## t2* 1.000000e+00  0.0028604397  0.06428831
## t3* 2.576929e-14  0.0004116624  0.04168928
```

```r
print(boot_com)
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
```

```
##
## Call:
## boot(data = omicron, statistic = betas, R = 300, formula = compound ~
##     mean_sub + mean_com)
##
##
## Bootstrap Statistics :
##          original       bias    std. error
## t1*  1.940385e-14 -0.003678963  0.03382074
## t2* -5.027921e-14  0.008881415  0.07977852
## t3*  1.000000e+00 -0.002162266  0.05431097
```

```
stargazer(reg_sub, reg_com, reg_sub_c, reg_com_c,
          title = "Table 1. Regression Results in the Sub Dataset (N=2000)",
          dep.var.labels = c("Individual Subjectivity", "Individual Polarity",
                             "Individual Subjectivity", "Individual Polarity"),
          covariate.labels = c("Public Subjectivity", "Public Polarity",
                               "Number of New Cases", "Number of New Deaths"),
          column.sep.width = "1pt",
          omit.stat = c("ll", "aic", "bic", "f", "ser"),
          font.size = "small",
          header = FALSE)
```

Table 1: Table 1. Regression Results in the Sub Dataset (N=2000)

	Dependent variable:			
	Individual Subjectivity	Individual Polarity	Individual Subjectivity	Individual Polarity
	(1)	(2)	(3)	(4)
Public Subjectivity	0.865^{***}	-0.057	0.834^{***}	-0.046
	(0.261)	(0.335)	(0.265)	(0.341)
Public Polarity	-0.0004	1.031^{***}	0.008	1.041^{***}
	(0.185)	(0.238)	(0.192)	(0.247)
Number of New Cases			0.00000	-0.000
			(0.00000)	(0.00000)
Number of New Deaths			-0.00001	-0.00000
			(0.00001)	(0.00001)
Constant	0.046	0.009	0.034	0.011
	(0.110)	(0.142)	(0.111)	(0.143)
Observations	2,000	2,000	2,000	2,000
R^2	0.006	0.010	0.006	0.010
Adjusted R^2	0.005	0.009	0.004	0.008
Note:				*p<0.1; **p<0.05; ***p<0.01

Discussion

In the present study, we find that there is a significant effect of public sentiment on individual sentiment in social media context. Public subjectivity has a positive influence on individual subjectivity and a negative influence on individual polarity. On the other hand, public polarity only has a positive relationship with individual polarity. This result validates our assumption that individual sentiment (or moer braodly, attitudes) changes with the sentiment of the whole social media community.

The present study also has some limitations. First, it would be better if we could include more relevant covariates representing the pandemic growth (i.e. the number of new omicron variant cases), into our regression models. Unfortunately, it's hard to obtain daily reports regarding new cases of a specific variant. Some other covarites, such as the demographic and social-economic variables, should also been taken into account if they could be linked to the tweets data. This also leads to our second limitation. Due to the limited access level, we are unable to retrieve geographic information of the tweets from Twitter API. Therefore, our conclusions may be too generalized for specific countries or regions, as we cannot tell whether the effects of public sentiments hold in different cultures. Further studies are expacted in this field.

Reference

Ferrari, J. R. (2001). Procrastination as self-regulation failure of performance: effects of cognitive load, self-awareness, and time limits on 'working best under pressure'. European journal of Personality, 15(5), 391-406.

Forelle, M., Howard, P. N., Monroy-Hernández, A., & Savage, S. (2015). Political bots and the manipulation of public opinion in Venezuela. Available at SSRN 2635800.

Hayes, J. L., & King, K. W. (2014). The social exchange of viral ads: Referral and coreferral of ads among college students. Journal of Interactive Advertising, 14(2), 98-109.

Jacks, J. Z., & Cameron, K. A. (2003). Strategies for resisting persuasion. Basic and applied social psychology, 25(2), 145-161.

Lee, A. R., Son, S. M., & Kim, K. K. (2016). Information and communication technology overload and social networking service fatigue: A stress perspective. Computers in Human Behavior, 55, 51-61.

Niu, G., Yao, L., Tian, Y., Sun, X., & Zhou, Z. (2020). Information overload and the intention to reduce SNS usage: the mediating roles of negative social comparison and fatigue. Current Psychology, 1-8.

Nguyen, L. T., Wu, P., Chan, W., Peng, W., & Zhang, Y. (2012, August). Predicting collective sentiment dynamics from time-series social media. In Proceedings of the first international workshop on issues of sentiment discovery and opinion mining (pp. 1-8).

Parslow, G. R. (2011). Commentary: How the internet is changing the way we think, read and remember. Biochemistry and Molecular Biology Education, 39(3), 228.

Supplement Materials

GitHub Repository

https://github.com/hmsun0813/surv727_final

Python Code for Data Collection

```python
import requests
import json
import time
import pandas as pd

bearer_token = '<Bearer Token>'
ENDPOINT = "https://api.twitter.com/2/tweets/search/recent"


def bearer_oauth(r):
    """
    Method required by bearer token authentication.
    """

    r.headers["Authorization"] = f"Bearer {bearer_token}"
    r.headers["User-Agent"] = "v2RecentSearchPython"
    return r


def connect_to_endpoint(url, params):
    """
    Request tweets from twitter api v2.

    Parameters:
        url (str): the endpoint url
        params (dict): a dict of the query parameters

    Return:
        response.json() : a json object containing the response from twitter api
    """

    response = requests.get(url, auth=bearer_oauth, params=params)
    print(response.status_code)
    if response.status_code != 200:
        raise Exception(response.status_code, response.text)
    return response.json()


def datetime(date_list, time_list):
    """
    Iteratively generate formatted start_time and end_time for query

    Parameters:
        date_list (list): list of dates to collect data
        time_list (list): list of times to collect data

    Return:
        (start_time, end_time) (tuple): a tuple contaings two lists,
            start_time list and the correspoding end_time list.
    """

    start_time = []
    end_time = []

    for date in date_list:
        for time in time_list:
            time_index = time_list.index(time)

            if time_index == 12:
                continue

            start_time.append(f'{date}T{time}Z')
            end_time.append(f'{date}T{time_list[time_index+1]}Z')

    return (start_time, end_time)


def main():
    """
    Note:
        With current authoritation, the maximum of tweets returned per response is 100.
        In order to retrieve tweets more than that,
        looking for and adding next_token to the query repeatedly is needed here.
    """

    # Set time units
    date_list = [
        '2021-11-27', '2021-11-28', '2021-11-29', '2021-11-30',
        '2021-12-01', '2021-12-02', '2021-12-03', '2021-12-04'
    ]

    time_list = [
        '00:00:00', '02:00:00', '04:00:00', '06:00:00',
        '08:00:00', '10:00:00', '12:00:00', '14:00:00',
        '16:00:00', '18:00:00', '20:00:00', '22:00:00', '23:59:59'
    ]

    start_times, end_times = datetime(date_list, time_list)
    start_times = start_times[5:-9]
    end_times = end_times[5:-9]

    # Loop all time units
    for i in range(0, len(start_times)):

        # Initialize settings for each time unit
        query_params = {
            'query': '#Omicron (lang:en) -is:retweet',
            'max_results': '100',
            'start_time': None,
            'end_time': None
        }
        next_token = None
        text_data = pd.DataFrame(columns=['id', 'text', 'start_time'])

        # Iterate the request by pagination (next_token)
        for j in range(0, 10):
            query_params.update({
                'start_time': start_times[i],
                'end_time': end_times[i]
            })

            # Update the query; Break the loop if there is no next_token any more
            if j:
                if next_token:
                    query_params.update({'next_token':next_token})
                else:
                    break

            # Request tweets
            json_response = connect_to_endpoint(ENDPOINT, query_params)

            df = pd.DataFrame(json_response['data'])
            df['start_time'] = start_times[i]
            text_data = pd.concat([text_data, df])

            # Look for next_token
            if 'next_token' in json_response['meta'].keys():
                next_token = json_response['meta']['next_token']
            else:
                next_token = None

            print(f'Success -- Start_time{start_times[i]}, page:{j+1}')
            j += 1
            time.sleep(10)

        text_data.to_csv(f'text_{i}.csv')


if __name__ == "__main__":
    main()
```

Python Code for Data Processing

```
import os
import pandas as pd
import re


def data_cleaning(text):
    """
    Regular Expression is used to remove all @(Reply), #(Hashtag), and URL

    Parameters:
        text (str): original text

    Return:
        text_cleaned (str): cleaned version of the original text
    """

    # Remove @(Reply)
    text_1 = re.sub(r'@[^\s]*', "", text)
    # Remove #(Hashtag)
    text_3 = re.sub(r'#[^\s]*', "", text_1)
    # Remove URL
    text_cleaned = re.sub(r'[a-zA-Z]+://[^\s]*', "", text_3)

    return text_cleaned


def text_sample_join(folderpath, frac, df):
    """
    Resample all datasets in the folder with a given fraction,
    Join them into one dataset

    Parameters:
        folderpath (str): the path of the folder holding all datasets
        frac (float): resampling fraction
        df (DataFrame): the dataset to join all the subsets into

    Return:
        df (DataFrame): resampled and joined dataset
    """

    # Traverse all files in the folder
    for file in os.listdir(folderpath):

        # Generate the file path
        filepath = os.path.join(folderpath, file)

        # Resample and join to df if the path is assigned a file
        if os.path.isfile(filepath):
            text_data = pd.read_csv(filepath, lineterminator='\n')
            text_sample = text_data.sample(frac=frac, random_state=1216)
            df = pd.concat([df, text_sample])

    return df


def main():

    df = pd.DataFrame(columns=['id', 'text', 'start_time'])
    text_joined = text_sample_join('text_data', 0.5, df)
```

```python
    text_joined['text_cleaned'] = text_joined['text'].apply(data_cleaning)

    text_joined.drop(['Unnamed: 0', 'withheld'], axis=1).to_csv('text_processed.csv')


if __name__ == "__main__":
    main()
```

Python Code for Sentiment Analysis

```python
import pandas as pd
import numpy as np
from textblob import TextBlob
from nltk import tokenize
from nltk.sentiment.vader import SentimentIntensityAnalyzer


def get_subjectivity(text):
    """
    Parameters:
        text (str): source data

    Return:
        subjectivity_score (float)
    """

    return TextBlob(text).sentiment.subjectivity


def get_vader_score(text):
    """
    Caluculate the average vader assessment scores for the input text

    Parameters:
        text (str): source data

    Return:
        vader_scores (dict): a dict contains compound score, negitive score,
            neutral score, positive score.
    """

    # Initialize settings
    sentences = tokenize.sent_tokenize(text)
    vader_scores = {
        'compound': [],
        'neg': [],
        'neu': [],
        'pos': [],
    }
    sid = SentimentIntensityAnalyzer()
    # Calculate Vader scores for each individual sentence
    for sentence in sentences:
        polarity = sid.polarity_scores(sentence)
        for key in polarity.keys():
            vader_scores[key].append(polarity[key])

    # Calculate the average Vader scores for the input text
    for key, val in vader_scores.items():
        vader_scores[key] = np.nanmean(val)

    return vader_scores


def main():

    tweets = pd.read_csv("text_processed.csv")

    tweets['subjectivity'] = tweets['text_cleaned'].apply(get_subjectivity)
    tweets['polarity'] = tweets['text_cleaned'].apply(get_vader_score)
    tweet_sentiment = pd.concat([tweets, tweets['polarity'].apply(pd.Series)], axis=1)
    tweet_sentiment = tweets.drop(['polarity', 'Unnamed: 0'], axis=1)

    tweet_sentiment.to_csv('tweet_sentiment.csv')


if __name__ == "__main__":
    main()
```