How does Public Attitude towards Dogecoin Change Between Surges and Plunges Periods?

Course Code: COM 5507

Course Title: Social Media Data Acquisition and Processing (2021/2022)

Course Instructor: Dr. LIU Xiaofan

Group-8

Yuqing Yan 56916222, Zebin Wu 56923919,

Xinzhe Li 57015847, Jiayu Jin 56805509

Communication and New Media

City University of Hong Kong

CONTENTS

Author List

Xinzhe Li, Jiayu Jin, Yuqing Yan and Zebin Wu are jointly responsible for the data acquisition, data cleaning and data processing as well as the report writing of the whole project.

Abstract

This project aims to unfold textual differences in public discussion about Dogecoin on social media platform in China between typical surges and plunges in 2021.

We choose Weibo as our data source because it is the most popular public social media platform in China. According to the price movements of Dogecoin in 2021, we select two periods (2021.4.13-4.17, 4.28-5.8) to collect our textual samples in surges and two periods (2021.5.9-5.13, 6.3-6.22) for plunges. Based on the conventional content analysis such as word frequency analysis, sentiment analysis is conducted to reveal the potential differences in public emotion and attitude alongside the surges and plunges.

Concerning findings, discussions about dogecoin are mainly about the financial value that it has currently. Also, they are closely related to hot news events, especially related to tweets posted by Elon Musk. Besides, in surges, discussions are mainly about the price trend. People are willing to talk about investment behavior decisions. However, in plunges, people may easily be in negative emotions. Therefore, people may be intended to find something to rely on emotionally and psychologically.

Introduction

The origin of Dogecoin was a joke, which has made it have a long and problematic history of scams. Similar to many other cryptocurrencies, Dogecoin has been described by some commentators as a form of Ponzi scheme. Critics allege that Dogecoin investors who purchased Dogecoins early on, have a significant financial incentive to draw others into purchasing more Dogecoins in order to drive the price up, therefore benefitting the early investors financially at the direct expense of later purchasers.

For example, there is a claim that Elon Musk's actions amount to market manipulation of Dogecoin due to the price of it frequently experiencing dramatic movements in hours shortly after Dogecoin-related tweets are released by Elon Musk. And he does frequently uses his Twitter to express his views on Dogecoin. Palmer who is the co-founder of Dogecoin also called Elon Musk a self-serving scammer on Twitter. In his other Twitter, he pointed out that the cryptocurrency industry has already built a cult-like illusion of overnight riches to gain profit from those who are crazy about getting rich and lack common financial knowledge using an ambiguous network of corporate connections, paid public opinion leaders and paid media networks.

However, since 2021, an increasing number of individual Chinese investors have been participating in Dogecoin investments for its potential profit. Dramatic surges and plunges are still happening in turn in hours. This ongoing phenomenon we mentioned above motivated us to figure out textual and emotional differences in public discussion about Dogecoin on Weibo between these typical surges and plunges in 2021. We believe that figuring out an overview of typical public discussions about

Dogecoin will help ordinary investors rethink financial risks about Dogecoin from a rational perspective.

Consequently, we want to figure out the following questions:

What are the common topics on Weibo during these plunges and surges?

What is the most significant difference in textual content between these plunges and surges?

What is the most significant difference in text sentiment orientation in public discussion between these plunges and surges?

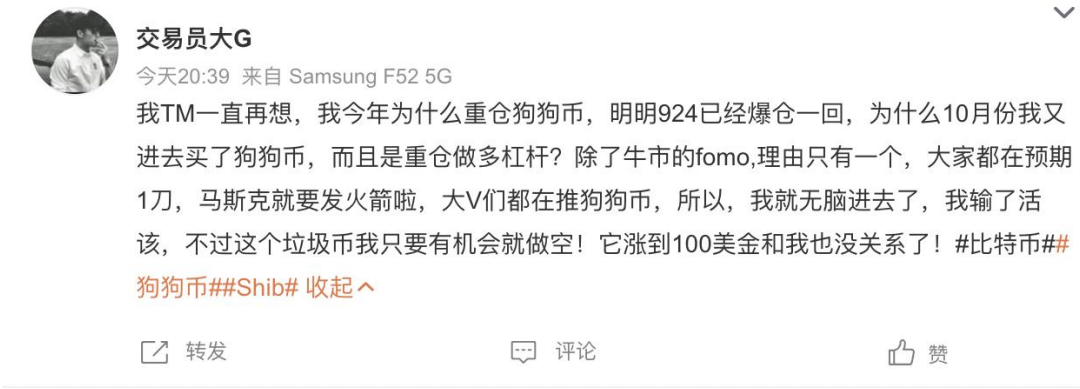
Method

Data acquisition

Variables

We crawled 61, 82, 84, and 26 popular tweets separately in 4 time periods we choose: 2021.4.13-4.17, 4.28-5.8, 2021.5.9-5.13, 6.3-6.22, for a total of 253 tweets.

As GraphX shows, our spiders crawling location is “card” and only crawl the text part of it.



Graph 1. The target content of our spider

Through this step we obtained variables for three dimensions: username, tweet content, time and source.

Procedure

We used selenium to open the Weibo searching URL, manually entered the username and password to log in, and then entered our search term “狗狗币” after the selenium location search key.

```
driver = webdriver.Chrome('/Users/lixinzhe/Desktop/COM5507/chromedriver')
driver.get("https://login.sina.com.cn/signup/signin.php")
```

```
s_input = driver.find_element_by_css_selector('#search_input')

s_input.send_keys("狗狗币")
# 定位搜索键
confirm_btn = driver.find_element_by_css_selector('#search_submit')

confirm_btn.click()
```

Graph 2. Login and key words searching

Then we clicked the "Advanced Search" button through selenium, selected the "Popular" option, and then manually selected the time period we needed to crawl and click Search.

```
#点击"高级搜索"键
driver.switch_to.window(driver.window_handles[-1])
Advanced = driver.find_element_by_css_selector("#pl_feedtop_top")
Advanced.click()
```

```
#点击热门
Hot = driver.find_element_by_css_selector("#radio02")
Hot.click()
#此处手动输入时间并点击搜索
```

Graph 3. Advanced search

In this part we encountered two difficulties:

First, the advanced search is a hover window, and the code to enter the time through selenium is difficult to locate and too complicated for us at the moment, so we finally chose to enter the time period manually and click the search button. This is a point we need to improve in the future.

Second, there is a 50-page search limit for Weibo, i.e., no matter how many tweets there are in a period of time, only 50 pages can be displayed at most. In order to avoid that some time periods cannot be obtained because of the number of pages displayed, we chose to crawl popular tweets, and the number of popular tweets are all within 50 pages, so as to ensure equal treatment for each time period crawled.

Then we crawled the content of each Weibo card through a loop. First, we created three lists that store elements, then we used selenium's "find_element_by_css_selector" function to determine whether each tweet is collapsed because it is too long, and we used the " click to expand" command for collapsed tweets. Then we continued to use "find_element_by_css_selector" to find the three variables we need and store them in the list.

```
comment = []
username = []
datentime = []

# 抓取节点：每个评论为一个节点（包括用户信息、评论、日期等信息），如果一页有20条评论，那么nodes的长度就为20
nodes = driver.find_elements_by_css_selector('div.card > div.card-feed > div.content')
print(len(nodes))

# 对第一页每个节点进行循环操作
for i in range(0, len(nodes), 1):
    # 判断每个节点是否有"展开全文"的链接
    flag = False
    try:
        nodes[i].find_element_by_css_selector("p>a[action-type='fl_unfold']").is_displayed()
        flag = True
    except:
        flag = False
    if (flag and nodes[i].find_element_by_css_selector("p>a[action-type='fl_unfold']").text.startswith('展开c')):
        nodes[i].find_element_by_css_selector("p>a[action-type='fl_unfold']").click()
        comment.append(nodes[i].find_element_by_css_selector('p[node-type="feed_list_content_full"]').text)
    else:
        comment.append(nodes[i].find_element_by_css_selector('p[node-type="feed_list_content"]').text)

    username.append(nodes[i].find_element_by_css_selector("div.info>div:nth-child(2)>a").text)
    datentime.append(nodes[i].find_element_by_css_selector("p.from").text)
```

Graph 4. Elements crawling

Next, through a "for" loop, we first found the number of pages of popular tweets in one time period, then manually entered the page numbers, repeated the crawling action mentioned in the previous paragraph for each remaining page, and stored all the data in the list.

```
#抓取接下来每一页的节点，并提取信息放入列表
for page in range(6):
    print(page)
    nextpage_button = driver.find_element_by_link_text('下一页')

    driver.execute_script("arguments[0].click();", nextpage_button)
    wait = WebDriverWait(driver, 5)

    nodes1 = driver.find_elements_by_css_selector('div.card > div.card-feed > div.content')
    for i in range(0, len(nodes1), 1):
        flag = False
        try:
            nodes1[i].find_element_by_css_selector("p>a[action-type='fl_unfold']").is_displayed()
            flag = True

        except:
            flag = False
        if (flag and nodes1[i].find_element_by_css_selector("p>a[action-type='fl_unfold']").text.startswith('展开c')):
            nodes1[i].find_element_by_css_selector("p>a[action-type='fl_unfold']").click()
            comment.append(nodes1[i].find_element_by_css_selector('p[node-type="feed_list_content_full"]').text)
        else:
            comment.append(nodes1[i].find_element_by_css_selector('p[node-type="feed_list_content"]').text)
        username.append(nodes1[i].find_element_by_css_selector("div.info>div:nth-child(2)>a").text)
        datentime.append(nodes1[i].find_element_by_css_selector("p.from").text)
```

Graph 5. Crawl the rest pages

Then we used the built-in function of the pandas package to organize the data into a dataframe and then saved it into an excel file.

```
data = pd.DataFrame({'username': username, 'comment': comment, 'datentime': datentime})
data.to_excel("狗狗币4.13-4.17.xlsx")
```

Our raw data is like this:

	A	B	C	D	E	F	G
1		username	comment	datentime			
2	0	坡坡Depo	#狗狗币大涨# 深圳还是卧虎藏龙。那天喝茶的时候，旁边坐了两个人，在聊狗狗币，现在想来，他俩这两天脸都要笑烂了。	04月17日 01:14 来自 iPhone 12 Pro			
3	1	新浪财经	#狗狗币还能买吗#【疯狂！单日暴拉200% #狗狗币一年狂飙180倍# 泡沫恐将破裂？】据CoinMarketCap.com的数据，狗狗币在埃隆·马斯克和马克·库班等人的推动下，周五（4月16日）一度上涨超200%。目前狗狗币总市值超490亿美元，超越"稳定币"USDT排名加密货币市值第五。与一年前的0.002美元和2.5亿美元相比，它现在上涨了18000%。O疯狂！这一加密货币单日暴拉200% 一年狂飙18000% 泡沫恐将破裂？ 收起d	04月17日 09:19 来自 微博 weibo.com			
4	2	翟文杰-	狗狗币1.2美元第三目标 第二目标是0.75 第一目标是0.48 请相信我 反正你们是不会买的，看戏就好#狗狗币#	04月17日 17:04 来自 iPhone 12			
5	3	东莞身边	#狗狗币为什么这么火#一种和比特币类似的叫Doge Coin(狗狗币）加密货币最近在疯长，4月16日，狗狗币日内涨幅超200%，报0.397美元/枚，狗狗币总市值超490亿美元，近一年狂飙18000%。#狗狗币大涨#	04月17日 14:43 来自 iPhone 12 Pro Max			

Graph 6. Raw data

Data Visualization

Data Cleaning: Split Words

First, we imported raw data, because the word frequency analysis is only for microblog content, so we extracted the "comment (that is, microblog content)" part of the dataframe and transformed it into a convenient string format for word separation.

```
commentData = data['comment']
print(commentData.isnull().sum())

comdata = []
for i in commentData:
    comdata.append(i)

Comdata = str(comdata)
print(type(Comdata))
```

Graph 7. Transform data to string form

Then, we used Harbin Institute of Technology University's stopwords list to remove some unuseful words in the text, such as tone words, numbers, etc. We also added some stopwords for our data, such as "put away, day, month", etc.

```
stopwords = pd.read_csv('/Users/lixinzhe/Desktop/COM5507/Final_project/哈工大stopwords.csv', encoding='utf8', names=['st
print(type(stopwords))
stop_list = stopwords['stopword'].tolist()
delword = ['还','收起','\\','d','O','日',' ','月','都','']
for deli in delword:
    stop_list.append(deli)
print(type(stop_list))
```

Graph 8. Stop the unuseful words

Next, we used Jieba packages to split the text into words. Before splitting, we added a userdict based on the text content of the data, which contains meaningful terms such as "dogcoin", "bitcoin", "shanzhaicoin ", we did not want them to be separated in the word splitting.

```
Data = []
jieba.load_userdict('/Users/lixinzhe/opt/anaconda3/lib/python3.8/site-packages/jieba/dogbit.txt')
seg_list = jieba.cut(Comdata, cut_all=False)
for word in seg_list :
    if word not in stop_list:
        print(word)
        Data.append(word)
```

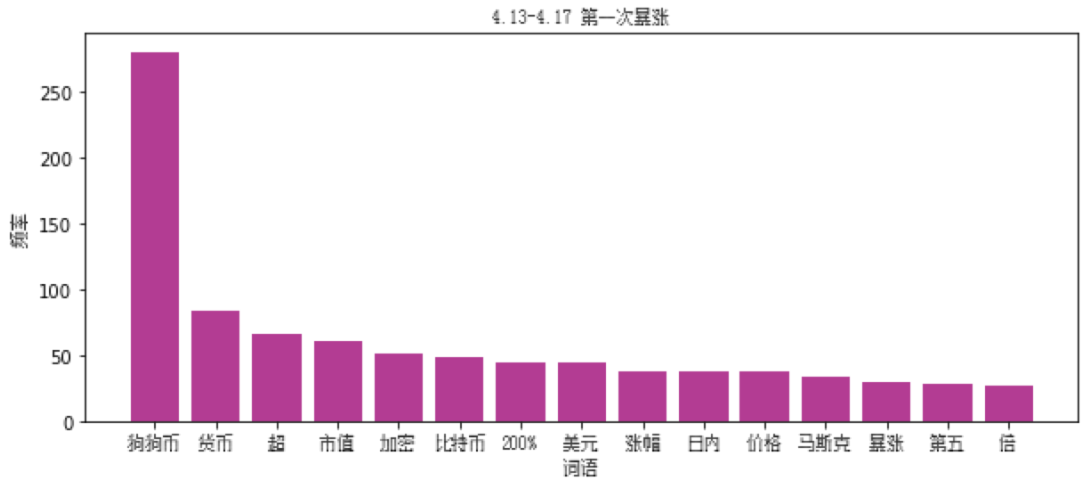
Finally, we counted the divided words by Words Counting function and arranged them in the order from highest to lowest.

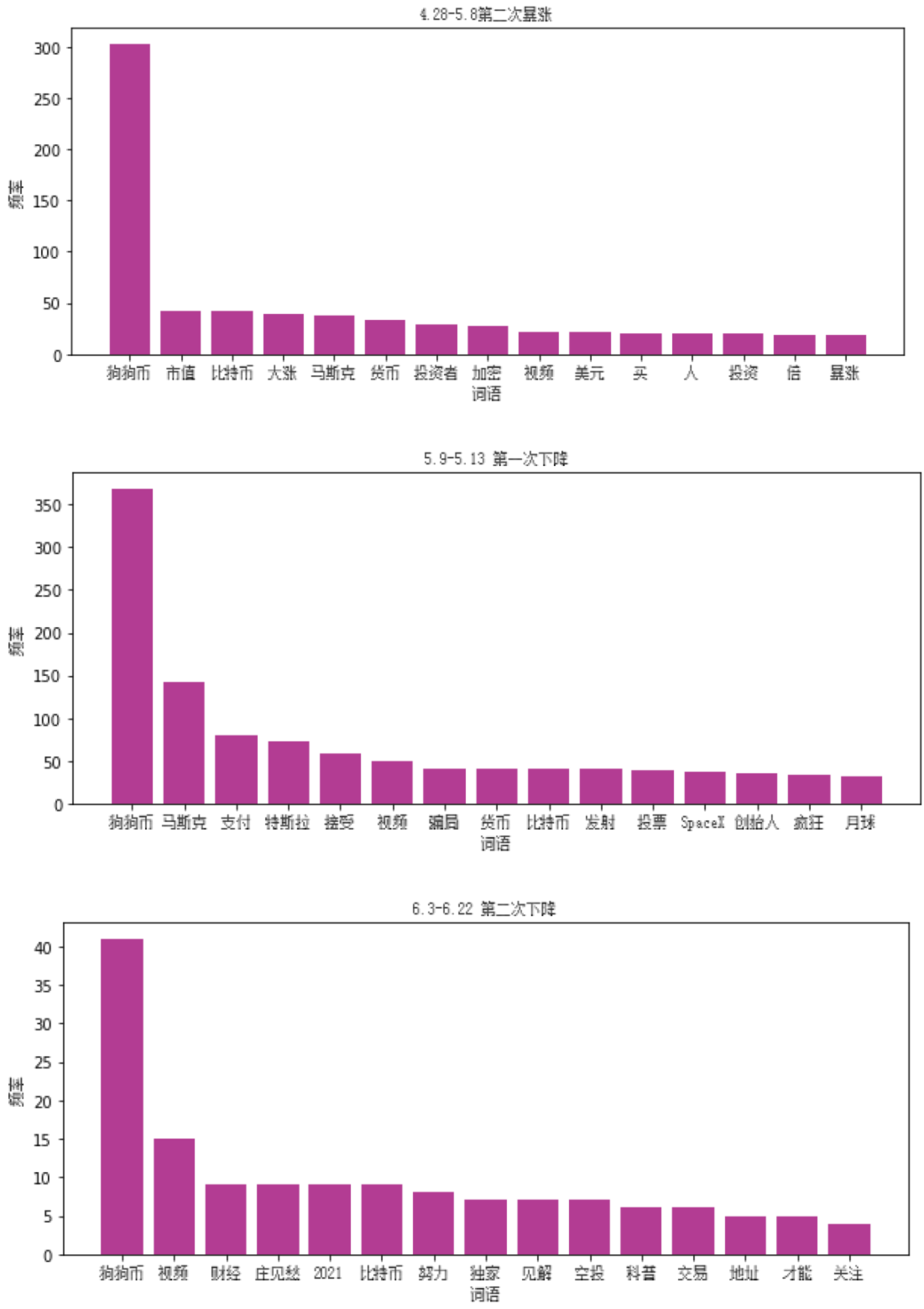
```
wordcount = {}
for word in Data:
    wordcount[word] = wordcount.get(word, 0)+1
sorted(wordcount.items(), key=lambda x: x[1], reverse=True)[:20]
```

Graph 9. Wordcount

Data Processing

As for the data visualization, we use the package of Matplotlib to transfer our results into bar charts in four time periods respectively which can show the word frequency more intuitionistic than the direct results of wordcount package.





Graph 10. Bar charts for 4 periods

For further data visualization, we decided to make a word cloud to show the frequency of data. Wordcloud, PIL, numpy and matplotlib are installed and after that we read the dataset. In order to make the picture more interesting, different kinds of

pictures including dogs and dollars are uploaded and are set as the backgrounds.

```
#Backgroud image loading.
#设置背景图片。
background = Image.open("doge.jpg")
graph = np.array(background)
```

Graph 11. Background image uploading

But during the process of testing the word cloud, we met the problem that some words like “dogecoin” and “bitcoin” will appear more than once. It is quite inappropriate for a word cloud, in order to guarantee the accuracy of visualization, we deal with the difficulties by adding an extra code “collocation=false” to avoid and eliminate these repeated words. We also modified the index of the word cloud including the width and height and uploaded a new font for it.

```
# Image format setting.
# 设置图片制式。
wcd=WordCloud(font_path=r'/Users/jin/Desktop/SEM_A/5507-data-
              collocations=False,
              width=500,
              height=400,
              background_color='white',
              max_words=100,
              mask=graph,
              scale=1.5).generate(txt)
```

Graph 12. Image format setting

Then the word cloud is generated and came out in the way of specified pictures. And word clouds are the four stages of dogecoin which are demonstrated in different figures are as below.

4.13-4.17

4.28-5.8

5.9-5.13

6.3-6.22

Graph 13. Word cloud for 4 periods

Sentiment Analysis

To find out the change of mood in these four periods, we conduct the sentiment analysis. We applied the “Chinese-word-vectors” known as Word + Ngram, which is an open source from the researchers in Institute of Chinese Information Processing, Beijing Normal University and DBIIR Laboratory, Renmin University of China. We also used the Chinese corpus from Dr. Tan Songbo as our training text. The corpus is about the user reviews of the hotel on Ctrip, with 2000 texts being negative, and 2000 being positive.

Data cleaning

First, we unzipped the downloaded Chinese-word-vectors file and used gensim to load this pre-training word vectors model. In this model, each word is corresponded by a vector with a length of 300.

```
ut the dowloaded zip file into embeddings folds
nzip the downloaded Chinese-word-vectors file
h open("C:/Users/12529/Desktop/datala/embeddings/sgns.zhihu.bigram", 'wb') as new_file, open("C:/Users/12529/Desktop/datala/embed
 decompressor = bz2.BZ2Decompressor()
 for data in iter(lambda : file.read(100 * 1024), b''):
     new_file.write(decompressor.decompress(data))
```

```
# use genism to load pre-training word vector
cn_model = KeyedVectors.load_word2vec_format('C:/Users/12529/Desktop/datala/embeddings/sgns.zhihu.bigram',
                                             binary=False)
```

```
embedding_dim = cn_model['香港城市大学'].shape[0]
print('词向量的长度为{}'.format(embedding_dim))
cn_model['香港城市大学']
```

```
词向量的长度为300

array([ 0.023013,  0.099276, -0.188544,  0.014892, -0.140755, -0.275439,
       -0.095019,  0.064443, -0.205539, -0.046788, -0.229846,  0.408116,
        0.173516, -0.053986,  0.278121, -0.487348,  0.006504,  0.028538,
        0.321313,  0.185489,  0.20306 ,  0.034164, -0.163552, -0.02637 ,
       -0.166576, -0.070172, -0.12176 , -0.017719,  0.218962,  0.21611 ,
       -0.127903, -0.001921, -0.290569,  0.077451,  0.026128,  0.062851,
        0.045356, -0.007478, -0.057944,  0.051172, -0.232546, -0.172068,
       -0.247718,  0.099174,  0.127989,  0.098246,  0.125703,  0.071   ,
        0.006947,  0.078257, -0.109834, -0.232685, -0.005075, -0.080271,
```

Graph 14. Loading the word vectors

As the LSTM neural network model cannot process Chinese text directly, the words need to be segmented and converted into vectors. The main package to cut the words is jieba. First, we put all the training text in the form of a list, where each review was stored as a string. Then we removed the punctuation, followed by using jieba to participle and convert the result into a list. Next, we indexed these words, making each word in the texts an index number corresponding to the words in the pre-training vector model.

```
# Segmentation and tokenize
# train_tokens is a list with 4,000 strings
train_tokens = []
for text in train_texts_orig:
    # Remove the punctuation
    text = re.sub("[\s+\.\!\/_,$%^*(+\"\']+|[+——！，。？、~@#￥%……&*（）]+", "",text)
    # Participle
    cut = jieba.cut(text)
    # The output is a generator
    # convert generator to list
    cut_list = [ i for i in cut ]
    for i, word in enumerate(cut_list):
        try:
            # convert word to index
            cut_list[i] = cn_model.key_to_index[word]
        except KeyError:
            cut_list[i] = 0
    train_tokens.append(cut_list)
```

```
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\12529\AppData\Local\Temp\jieba.cache
Loading model cost 0.557 seconds.
Prefix dict has been built successfully.
```

Graph 15. Participle and tokenize

Data processing

We used the Keras in TensorFlow to build the model.

As the preparation, we standardized the length of tokens first. Since the length of each review was different, it would be a waste of computing resources to take the longest review and fill the others to the same length. We checked the average length of all the tokens, and it was 71. Then we checked the token length of the longest text, and it was around 1,500. We need to take a compromise length that would cover most of the training samples without losing much information. Matplotlib was used to visualize the distribution of tokens. We assumed that the token lengths were normally distributed, taking the mean of the tokens and adding two standard deviations. The compromise length was 236, with 95% of the tokens could be covered.

```
# Taking the mean of the tokens and adding two standard deviations
# We assumed that the tokens lengths were normally distributed.
# the value of max_tokens could contain 95% of the tokens
max_tokens = np.mean(num_tokens) + 2 * np.std(num_tokens)
max_tokens = int(max_tokens)
max_tokens
```

```
236
```

```
# 95% of the tokens could be covered when the length is 236
np.sum( num_tokens < max_tokens ) / len(num_tokens)
```

```
0.9565
```

Graph 16. Standardization of the length of tokens

Then we used the Keras to create an embedding matrix with a dimension of (num words, matrix of embeddingdim). Num words was the number of words we used. There were 2.6 million words in the pre-training model, but our computer can only support us with a maximum of the first 250,000 words. The embedding dimension was 300, which is the length of vectors in our word vector model. Therefore, the matrix was (250000, 300).

After checking the index correspondence, we padded the under-length tokens and truncated the over-length ones. If the word exceeded 250,000 word vectors, it would be replaced by 0.

```
num_words = 250000
embedding_matrix = np.zeros((num_words, embedding_dim))
# embedding_matrix: [num_words · embedding_dim]
# 250000 * 300
for i in range(num_words):
    embedding_matrix[i,:] = cn_model[cn_model.index_to_key[i]]
embedding_matrix = embedding_matrix.astype('float32')
```

```
np.sum( cn_model[cn_model.index_to_key[333]] == embedding_matrix[333] )
```

```
300
```

```
embedding_matrix.shape
```

```
(250000, 300)
```

```
train_pad = pad_sequences(train_tokens, maxlen=max_tokens,
                          padding='pre', truncating='pre')
```

```
train_pad[ train_pad>=num_words ] = 0
train_pad[33]
```

```
array([     0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
            0,      0,      0,      0,      0,      0,      0,      0,
```

Graph 17. Creating the matrix & Padding and truncating tokens

We also defined a function to convert these tokens to texts for us to read, in case we might debug it later

```
def reverse_tokens(tokens):
    text = ''
    for i in tokens:
        if i != 0:
            text = text + cn_model.index_to_key[i]
        else:
            text = text + ' '
    return text
```

```
reverse = reverse_tokens(train_tokens[0])
```

Graph 18. Converting tokens to texts

Then we prepared the target vector. The first 2,000 samples were 1, and the last 2,000 samples were 0. We imported train_test_split from sklearn to separate the training sample from the test sample. 90% of the samples were used for training and the remaining 10% were used for testing. The random_state was also used to scramble the order of samples, as the regular arrangement was not conducive to model training.

```
# Prepared the target vector. The first 2,000 samples were 1, and the last 2,000 samples were 0.
train_target = np.concatenate( (np.ones(2000),np.zeros(2000)) )
```

```
# Separate the training sample from the test sample
from sklearn.model_selection import train_test_split
```

```
# 90% of the samples were used for training and the remaining 10% were used for testing
# The random_state was also used to scramble the order of samples
X_train, X_test, y_train, y_test = train_test_split(train_pad,
                                                    train_target,
                                                    test_size=0.1,
                                                    random_state=12)
```

Graph 19. Preparing the target vector

Finally, we come to the LSTM model building. The first and most important layer was embedding. The matrix we input was (batchsize, maxtokens). Batchsize referred to the number of samples, while maxtokens was the length of tokens, which was 236 that we have standardized before. The output matrix was (batchsize, maxtokens, embeddingdim). The second layer was Bidirectional LSTM with 32 units. This layer was used to return sequences after embedding. Then, these sequences will be imported to a 16-unit LSTM, the third layer, which did not return any sequences but the final result. This model ended with a full connected layer that used the sigmoid activation function to output the result. There were around 75 thousand trainable params in total. The learning rate was set to ben 0.001. Based on this output, we were able to determine whether the text belonged to a positive or negative emotion.

```
# Creat the first layer, the embedding layer.
model.add(Embedding(num_words,
                    embedding_dim,
                    weights=[embedding_matrix],
                    input_length=max_tokens,
                    trainable=False))
```

```
# Creat the second and third layer
model.add(Bidirectional(LSTM(units=32, return_sequences=True)))
model.add(LSTM(units=16, return_sequences=False))
```

```
# Creat the full connected layer
model.add(Dense(1, activation='sigmoid'))
# Use adam to optimize with 0.001 learning rate
optimizer = Adam(lr=1e-3)
```

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
```

Graph 20. Building the LSTM model

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 236, 300)          75000000

 bidirectional (Bidirectiona  (None, 236, 64)          85248
 l)

 lstm_1 (LSTM)               (None, 16)                5184

 dense (Dense)               (None, 1)                 17

=================================================================
Total params: 75,090,449
Trainable params: 90,449
Non-trainable params: 75,000,000
_________________________________________________________________
```

Graph 21. Model summary

To achieve greater accuracy in output, we have tried several neural network structures, such as GRU. The accuracy of the test sample could achieve 87%. However, when we tested our own Weibo samples, the final outputs were always around 0.5, which did not meet our needs. When the model identified a negative text, we expect its output to be closer to 0 rather than 0.5 to avoid misclassification. Then we found more confidence in the output of the BiLSTM, with more polarized values.

```
# The code of GRU
# model.add(GRU(units=32, return_sequences=True))
# model.add(GRU(units=16, return_sequences=True))
# model.add(GRU(units=4, return_sequences=False))
```

Graph 22. The code of GRU

Then, we defined three callback functions, optimizing the training process by adjusting some parameters automatically. The first callback function was the checkpoint, which was used to save the weights only when the validation loss was improved. An early stopping was also defined to stop training if the validation loss did

not improve within 3 epochs. The third callback was used to reduce learning rate automatically. Then, the train was started. 10% of the training sample was taken as a cross-validation sample as we said before.

```
# Create a checkpoint
path_checkpoint = 'sentiment_checkpoint.keras'
checkpoint = ModelCheckpoint(filepath=path_checkpoint, monitor='val_loss',
                                        verbose=1, save_weights_only=True,
                                        save_best_only=True)
```

```
# load the trained model
try:
    model.load_weights(path_checkpoint)
except Exception as e:
    print(e)
```

```
# Define an early stoping
earlystopping = EarlyStopping(monitor='val_loss', patience=3, verbose=1)
```

```
# Automatically reduce the learning rate
lr_reduction = ReduceLROnPlateau(monitor='val_loss',
                                         factor=0.1, min_lr=1e-5, patience=0,
                                         verbose=1)
```

```
# define these three callback function
callbacks = [
    earlystopping,
    checkpoint,
    lr_reduction
]
```

Graph 23. Defining callback functions

```
# Start training
model.fit(X_train, y_train,
          validation_split=0.1,
          epochs=20,
          batch_size=128,
          callbacks=callbacks)
```

```
Epoch 1/20
26/26 [==============================] - ETA: 0s - loss: 0.2593 - accuracy: 0.8981
Epoch 00001: val_loss improved from inf to 0.33366, saving model to sentiment_checkpoint.keras
26/26 [==============================] - 14s 400ms/step - loss: 0.2593 - accuracy: 0.8981 - val_loss: 0.3337 - val_accuracy: 0.
8750 - lr: 0.0010
Epoch 2/20
26/26 [==============================] - ETA: 0s - loss: 0.2222 - accuracy: 0.9170
Epoch 00002: val_loss improved from 0.33366 to 0.32969, saving model to sentiment_checkpoint.keras
26/26 [==============================] - 9s 352ms/step - loss: 0.2222 - accuracy: 0.9170 - val_loss: 0.3297 - val_accuracy: 0.8
806 - lr: 0.0010
Epoch 3/20
26/26 [==============================] - ETA: 0s - loss: 0.2144 - accuracy: 0.9148
Epoch 00003: val_loss improved from 0.32969 to 0.32854, saving model to sentiment_checkpoint.keras
26/26 [==============================] - 9s 360ms/step - loss: 0.2144 - accuracy: 0.9148 - val_loss: 0.3285 - val_accuracy: 0.8
667 - lr: 0.0010
Epoch 4/20
26/26 [==============================] - ETA: 0s - loss: 0.1810 - accuracy: 0.9318
Epoch 00004: val_loss improved from 0.32854 to 0.31852, saving model to sentiment_checkpoint.keras
26/26 [==============================] - 9s 356ms/step - loss: 0.1810 - accuracy: 0.9318 - val_loss: 0.3185 - val_accuracy: 0.8
972 - lr: 0.0010
```

Graph 24. Start training

According to the model.evaluate function, the accuracy of the test sample could

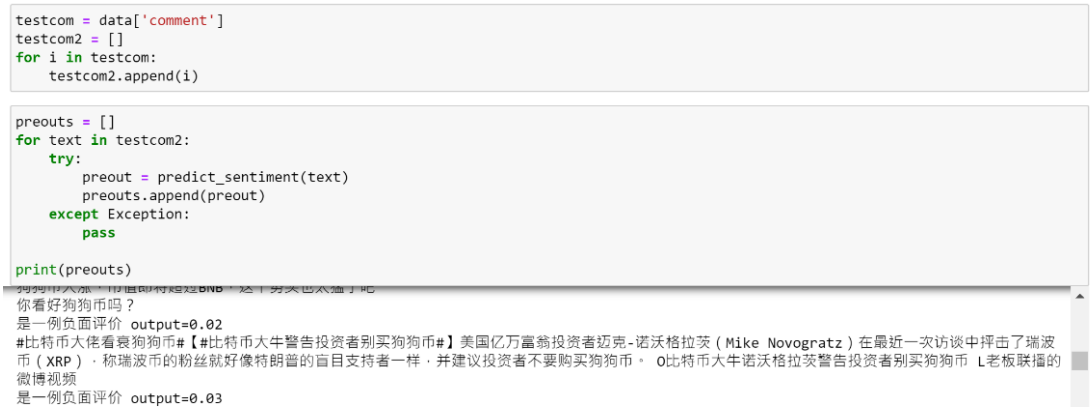
achieve around 87%. An important issue was that we found the accuracy after each run of the model was different. Although the difference in values was small, it still had an impact on the output. The Online reference sources suggested that this was because there were many random operations in the neural network algorithm. Also, the version of GPU we used might also cause randomness.

Lastly, we defined a prediction function to predict the sentiment coefficient of the input text. The whole process in this function, including removing punctuation, splitting words, tokenize and padding, was almost the same as the previous steps. The coefficient indicated the emotion of the text: if it was greater than or equal to 0.5, then the sentence was believed to have a positive emotion and if not, it would have a negative emotion.

```
def predict_sentiment(text):
    print(text)
    # removing punctuation
    text = re.sub("[\s+\.\!\/_,$%^*(+\"\']+|[+——！，。？、~@#￥%……&*（）]+", "",text)
    # cut the words
    cut = jieba.cut(text)
    cut_list = [ i for i in cut ]
    # tokenize
    for i, word in enumerate(cut_list):
        try:
            cut_list[i] = cn_model.key_to_index[word]
        except KeyError:
            cut_list[i] = 0
    # padding
    tokens_pad = pad_sequences([cut_list], maxlen=max_tokens,
                           padding='pre', truncating='pre')
    # prediction
    result = model.predict(x=tokens_pad)
    coef = result[0][0]
    if coef >= 0.5:
        print('是一例正面评价','output=%.2f'%coef)
    else:
        print('是一例负面评价','output=%.2f'%coef)
    return coef
```

Graph 25. Predicting the sentiment coefficient

In order to test our own sample and calculate the mean of the coefficient in the four periods, we imported pandas and numpy. We first stored the contents of Weibo in a list and then predicted the polarity of each content. Then, we stored the coefficients in a list and calculated their mean.



Graph 26. Predicting the polarity of contents

```
plunge2 = mean(preouts)
print(plunge2)
```

```
0.1836508
```

Graph 27. Calculating the mean

During the data processing, we encountered more than 50 times of the memory error. Also, since our model only contained 240,000 words, after importing our data, the program would report an invalid argument error and stops as soon as meeting a word outside the matrix. To solve this problem, we used the try except syntax. However, we found that invalid argument error is not a standard error, which means that we cannot skip it by using the “except invalid argument error”. Finally, we adopted the except exception instead of it, which finally solved the problem.

We also found out those texts that were misclassified, which was a total of 56 out of 400 sample texts. We found that the emotion of these misclassified texts was mostly ambiguous and even humans could not easily judge the polarity. For instance, the sentence with the index of 101 did not seem to have any positive attitude, but it was marked as positive in the training sample, and the prediction of negative evaluation made by this model seemed reasonable, which indicated the accuracy of our model.

```
# Check the misclassified texts
y_pred = model.predict(X_test)
y_pred = y_pred.T[0]
y_pred = [1 if p>= 0.5 else 0 for p in y_pred]
y_pred = np.array(y_pred)
```

```
y_actual = np.array(y_test)
```

```
# Find the misclassified index
misclassified = np.where( y_pred != y_actual )[0]
```

```
misclassified
```

```
array([  1,  14,  15,  21,  24,  29,  42,  59,  72,  73,  81,  87,  93,
       101, 108, 117, 118, 122, 127, 130, 135, 138, 151, 157, 160, 177,
       182, 186, 189, 194, 211, 215, 218, 238, 246, 253, 258, 267, 275,
       277, 283, 289, 292, 298, 302, 311, 316, 317, 340, 342, 347, 352,
       365, 373, 382, 394], dtype=int64)
```

```
# Output the misclassified indexs
len(misclassified)
print(len(misclassified))
```

```
56
```

Table 28. Checking the misclassified texts

```
# Find a sample of misclassified errors.
idx=101
print(reverse_tokens(X_test[idx]))
print('预测的分类', y_pred[idx])
print('实际的分类', y_actual[idx])
```

```
由于2007年新装修有一些新问题可能还没来得及解决我因为工作需要经常要住那里所以慎重的提出以下意见建议：1新装修后 的淋浴间淋浴喷头的位置都太高 我换了房间还是一样很不好用2新开业后的一些管理和服务还很不到位尤其是前台入住和结帐时代效率太低每次结帐都超过10分钟好像不符合 宾馆的要求
预测的分类 0
实际的分类 1.0
```

Table 29. Sample of misclassified texts

Results

In this study, according to research question 1, during the surges stage, people tend to talk about the “amount of increase”, “surpass” and “encryption”, excluding the keywords of “dogecoin” and “Maske”. While in the process of plunges, the netizens usually talk about “hoax”, “launch”, and “SpaceX”, “the moon” and “anxious”. It demonstrated that people will pay more attention to cryptocurrency if there is a

prosperous perspective. On the other hand, during the period of sustained slump, people will suspect the dogecoin and view it to be vulnerable. As for our second question, what is the most significant difference in textual content between these plunges and surges?

We can find that the words mean “increase” disappear which are replaced by “worry” and “hoax”. As for the netzines’ sentiment orientation between plunges and surges, they tend turn to a rather down mood when facing the constant plunges and score of plunges are 0.17233229 and 0.1836508. While the score of surges are 0.2190962 and 0.24133001. It shows a slightly difference between plunges and surges.

4.13 – 4.17

```
surge1 = mean(preouts)
print(surge1)
```

```
0.2190962
```

4.28 – 5.8

```
surge2 = mean(preouts)
print(surge2)
```

```
0.24133001
```

5.9 – 5.13

```
plunge1 = mean(preouts)
print(plunge1)
```

```
0.17233229
```

6.3 – 6.22

```
plunge2 = mean(preouts)
print(plunge2)
```

```
0.1836508
```

Graph 28. Sentiment analysis

But due to the fact that we only obtained the hot blogs instead of all the blogs, the sentiment analysis might not be accurate enough. Especially given that in the period of 6.3 – 6.22, almost no ordinary users express their opinions on the public platform, which will influence the result of the analysis.

On the other hand, the government of China crackdowns on currency speculation which resulted that the market of dogecoin thrinks. Ordinary customers quit the game and seldom make comments, by contrast, the market accounts keep upload and release information about the dogecoin. We also found that ordinary users made comments below theses market accounts, but we only get the content of the blogs instead of the comments below the blogs due to the limited time and technology. It also might impact the analysis.

Conclusion

Although the sentiment analysis scores in surges and plunges are both below 0.5 which means a negative trend in sentiment, the score in surges reflects a very slightly positive attitude compared with the plunge. Economic terms may reflect the financial value that dogecoin has currently. Nouns in news may suggest that the discussion is closely related to hot news events, especially related to tweets posted by Elon Musk. Scientific terms such as Bitcoin, crypto, etc. reflect technical benchmarks or principles to Dogecoin, and their word frequency is not as high as economic terms or nouns in news. This may indicate that the ordinary is more concerned with the financial value

and the possible relationship between price trends and news events rather than the technical principles of Dogecoin. Besides, in surges, Nouns such as rise or surge also indicate that discussions are mainly about the price trend. The verbs such as buy or invest, etc. indicate the behaviour tendency of people during this period. However, in the plunges, nouns such as sadness, effort, etc. may express people' s negative emotions. At that point, words such as exclusive and insight became the new high-frequency words, as people may try to find something to rely on psychologically and emotionally. Furthermore, a voice has emerged calling dogecoin a scam.

However, this study has two limitations. On the one hand, our data sample may lack representativeness because we merely collected data from popular Weibos during certain periods that we selected. Meanwhile, Weibo is perhaps not the most typical social media community for cryptocurrency investment. Besides, some Weibos about dogecoin may have been deleted by the platform before we conduct this research in accordance with regulatory requirements. On the other hand, the algorithm model we used for sentiment analysis may not be accurate enough.

In future research, a comprehensive and representative data source would be helpful to refine this study. Besides, it will be helpful to take social network analysis into application in this research. That will help us gain insight into the dissemination chain of popular information. Also, we will train the model with more and larger corpus than just the one on hotel reviews, to improve the accuracy of the model predictions.