Bitcoin Sentiment Analysis Using Tweets

Mihai Atimut – Exchange Student-Erasmus University of Rotterdam
Arpad Gerber – MBF-HSG
Sandor Lukacs – MBF-HSG

Ladislaus von Bortkiewicz Professor of Statistics
Humboldt-Universität zu Berlin
BRC Blockchain Research Center
lvb.wiwi.hu-berlin.de
Charles University, WISE XMU, NCTU 玉山学者

link to Github

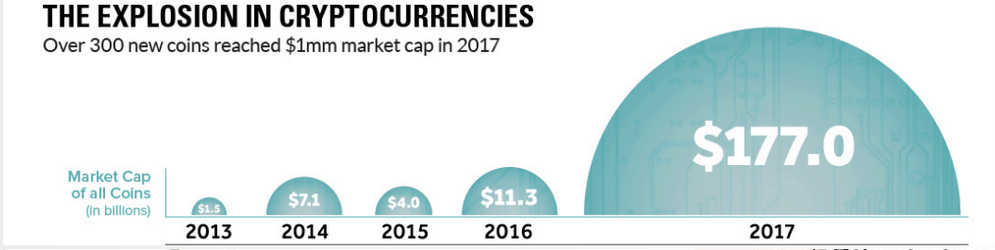
Outline

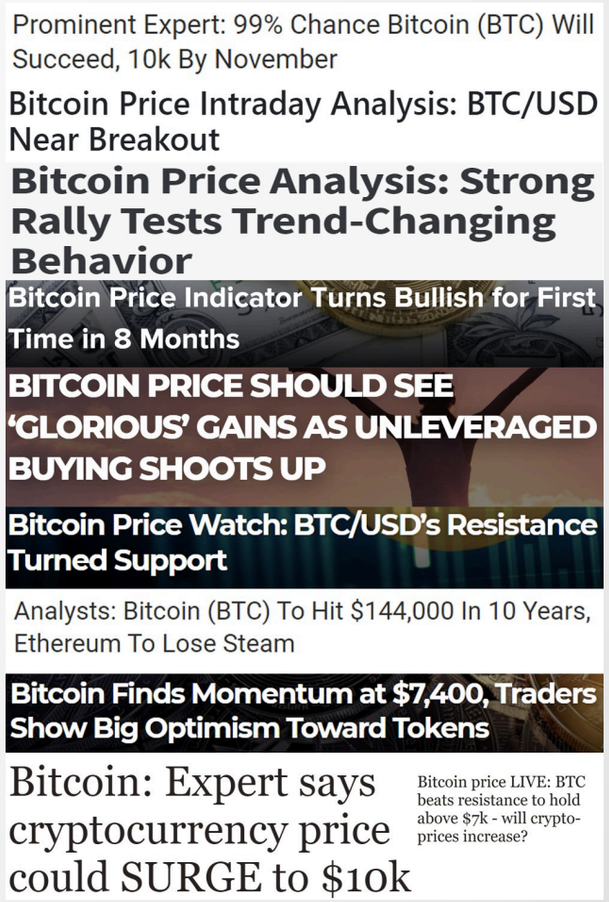
1. Motivation
2. Requirements and Description
3. Acquiring Tweets
4. Cleaning Tweets
5. Calculating Raw Sentiment Score
6. Aggregating Minutely Sentiment
7. Fetching Crypto Prices
8. Calculating returns, volatility and their derivative
9. Calculating cross-correlation between variables

Extra: Scagnostics

1: Motivation

- Bitcoin — no fundamentals & driven by sentiment
- Recently passed 20k dollars per BTC
- Cryptocurrencies are extremely volatile
- Goal: find if there is correlation between meaningful sentiments and measurable crypto performance using Twitter data





2: Description of the script and repository

- Main file to run:
  SDA_2020_St_Gallen_Bitcoin_Sentiment_Analysis_Using_Tweets.py
- Working directory of the project is:
  SDA_2020_St_Gallen_Bitcoin_Sentiment_Analysis_Using_Tweets
  (main folder of the QuantLet)
- Sample plots can be found in the main folder as well
- In the BTC folder:
  - tweets_raw.csv – The program saves the scraped data there
  - tweets_sample.csv – Prepared sample data
  - crypto_prices.csv – Prepared sample data
- We recommend to run the script using Spyder!
- The script has a simple UI that runs in the console
- All plots show under the plot tab after the code is finished in Spyder

2: Important prerequisites to run the code

- Before running the code, execute the following steps:
  - Install required libraries, see in the code! (Done manually)
  - Set the working directory! (Done manually)
  - Set the timezone for the environment (Automatically)

Relevant part in the code

```
import pandas as pd
from twython import Twython
from vaderSentiment.vaderSentiment import *
import cryptocompare
from datetime import datetime, timedelta
import re
from time import *
import os
import sys
import math
import numpy as np
import matplotlib.pyplot as plt
from pyscagnostics import scagnostics
from sklearn import preprocessing
import seaborn as sns
os.environ['TZ'] = 'Europe/London'
os.chdir("")
```

Install the following packages:

```
pip install twython
pip install vaderSentiment
pip install cryptocompare
pip install seaborn
pip install sklearn
pip install pyscagnostics
```

Set the working directory:

```
import os
os.chdir(directory)
```

Set timezone:

```
os.environ['TZ'] = 'Europe/London'
```

3: Scraping Tweets – The algorithm

Relevant part in the code
See function ExtractTweets in the code!

- Accessing the Twitter API with the Twython package
- We retrieve tweets containing the hashtags #bitcoin and #btc
- There are many limitations of the API, summarized in the table below:

No	Limitation	Our Solution
1	100 tweets/API call allowed	We only fetch 100 tweets per API call by using a loop that appends the new data to the output csv file ("BTC/tweets_raw.csv)
2	450 API calls/15 minutes allowed	We use a while(True) loop that runs until there are available tweets, waiting 15 minutes between every 45th API call (between every 45k tweet)
3	Cannot specify timeframe, only returns tweets before a given tweet id	We use a bootstrapping method, initially fetching the most recent 100 tweets and then acquiring the id of the oldest. Then, the script fetches the 100 tweets before this id.
4	Past 7 days available	There is no real loophole around this limitation. We use aggregation by the minute to have enough observations.

	id	text	fav	retw	time
0	1335694030555013120	RT @ante_hero: https://t.co/KvoJW3hNsh Flow G...	0	20	Sun Dec 06 21:14:07 +0000 2020
1	1335687997677654016	This is why #BitcoinCash #BCH, never #Bitcoin ...	2	0	Sun Dec 06 20:50:09 +0000 2020
2	1335687899858292736	RT @nonoeelse: #Bitcoin is in a class of its o...	0	2	Sun Dec 06 20:49:45 +0000 2020
3	1335686984677912583	RT @ante_hero: https://t.co/KvoJW3hNsh Flow G...	0	20	Sun Dec 06 20:46:07 +0000 2020
4	1335685773346136064	Would you feel more comfortable using a platfo...	2	0	Sun Dec 06 20:41:18 +0000 2020
5	1335685545624698880	RT @decoin_io: Epic scratch and win is back! a...	0	2	Sun Dec 06 20:40:24 +0000 2020
6	1335684288583323649	RT @nonoeelse: #Bitcoin is in a class of its o...	0	2	Sun Dec 06 20:35:24 +0000 2020

3: Scraping Tweets – The user interface

- Due to the limitations of the API, scraping all tweets takes approximately 2.5 hours
- When running the code, the user has the option to use prepared sample data, which is structured the same way as the scrapped data but runs much faster
  - To use pre-prepared data, write “yes” and hit enter – we recommend this solution
  - In app:

```
Step 1: Scraping Twitter
------------------------------------------------------------
Scraping all available raw tweets takes around 8 hours.

Would you like to use preapred and processed sample data instead? (yes/no):
```

- If, however, the user chooses scraping (writing “no”):
  - The script starts to making API calls for 100 tweets at a time
  - The script waits 15 minutes after scraping each block of 45000 tweets
  - In app:

```
You can quit the scraping anytime by hitting ctr+c, but code only works if the first 45k
tweets are downloaded
Scraped 9500 out of 45000
```

  - The user can stop the scraping at any point by hitting CTRL+C once
    - The script will automatically use the tweets acquired until that point
    - NOTE: Minimum tweets required is 45000
- NOTE: Code only runs if the API is reset to full capacity (45k tweets)
  - Otherwise: SystemExit: Not enough tweets downloaded!

4: Cleaning Tweets

- We clean the text of the tweets to be used in the sentiment analysis
- The script runs more efficiently on cleaned data
- We remove the following:
  - Hashtags (#)
  - Links (https://)
  - Retweet tags (RT:)
  - Special characters (@)

Example:
"RT #BTC over 23k USD, what a time to live in. https://btc.com"

↓

"BTC over 23k USD, what a time to live in"

Relevant part in the code

```
def CleanseTweets(df):

    local_tweets = df
    # Removing hashtags and retweets
    df.Text = [tweet.replace("#", "").replace("RT", "") for tweet in df.Text]
    # Removing most URLs and user taggings
    local_tweets['Text'] = local_tweets['Text'].apply(lambda
                                                      x: re.split('https:\/\/.*',
                                                                  str(x))[0])
    local_tweets['Text'] = local_tweets['Text'].apply(lambda
                                                      x: re.sub("@[A-Za-z0-9]+","",
                                                                str(x)))
    return(local_tweets)
```

5: We calculate the sentiment for each tweet

- Method: **V**alence **A**ware **D**ictionary for S**E**ntiment **R**easoning (VADER)
- VADER uses a dictionary that maps lexical features to emotion intensities, and aggregates them to sentiment scores.
- VADER considers negations, contradictions, punctuation, capital letters, emojis, intensifiers, acronyms, etc. to calculate the scores.
- VADER is especially suited to retrieve sentiments from social media content
- Example input:
  - `“btc near to take a decision, but we should be optimistic. What you think where bitcoin will go UP or DOWN?”`
- Example output:
  - `“{‘neg': 0.0, 'neu': 0.508, 'pos': 0.492, 'compound': 0.4404}”`

We use the “compound” measure:

$$compound = \frac{\sum_i Score_i}{\sqrt{(\sum_i Score_i)^2 + \alpha}} \qquad i \in (neg, neu, pos)$$

5: We calculate the sentiment for each tweet

- After running the code below we get the following data frame
  - It can take couple of minutes to run!

Relevant part in the code

```
def AnalyzeSentiment(df):
    # Creating a local instance of the tweets
    local_tweets = df
    # Initializing Vader
    analyzer = SentimentIntensityAnalyzer()
    # Creating containers for the results
    pos = []
    neg = []
    neu = []
    compound = []
    counter = 1
for tweet in local_tweets.Text:
    sys.stdout.write("\rAnalyzed {0} scores out of {1}".format(counter,
                                                               len(local_tweets)))
    sys.stdout.flush()
    vs = analyzer.polarity_scores(tweet)
    pos.append(vs['pos'])
    neg.append(vs['neg'])
    neu.append(vs['neu'])
    compound.append(vs['compound'])
    counter = counter + 1
df_tweets_analyzed = pd.DataFrame({'neg':neg,'pos':pos,'neu':neu, 'compound':compound})
df_tweets_concat = pd.concat([local_tweets, df_tweets_analyzed], axis = 1)
return(df_tweets_concat)
```

text	fav	retw	time	sent
in Bitcoin from 2017 or later y...	0	0	1607267742	0.3400
lly betting that ETH 2.0 won't...	0	2	1607266545	-0.2755
eaway\n\nWin\n1.5kTRX\n1...	0	23	1607265931	0.7430
at Grayscale are any indicatio...	0	417	1607264704	0.6369

$$sent \in [-1; 1]$$

6: Aggregating sentiment for each minute

⊡ We use minutely sentiment
We aggregate the sentiment scores for every minute

⊡ We aggregate by calculating the weighted average of scores
We weight the scores by the retweet count of each tweet

⊡ We also consider the tweets retweeted zero times
We add +1 to each retweet count, so that all tweets are included

⊡ Formula:

$$Agg.Sentiment = \frac{\sum (RT_i + 1) Score_i}{\sum RT_i + 1}$$

CreatedAt	0
2020-12-20 23:06:00+00:00	0.485418
2020-12-20 23:07:00+00:00	0.677282
2020-12-20 23:08:00+00:00	0.588683
2020-12-20 23:09:00+00:00	0.262508
2020-12-20 23:10:00+00:00	0.267591
2020-12-20 23:11:00+00:00	0.551651
2020-12-20 23:12:00+00:00	0.557517

Time index Sentiment

⊡ Relevant part in the code

```
def WeightedSentiment(df):
    # Creating a local instance of the tweets
    weight = df['Retw'].values+1
    elements = df['compound'].values
    weighted_sentiment = sum(weight*elements)/sum(weight)
    return(weighted_sentiment)

def ResampleDataframe(df):
    # Creating a local instance of the tweets
    local_tweets = df
    local_tweets['CreatedAt'] = pd.to_datetime(local_tweets['CreatedAt'])
    local_tweets = local_tweets.set_index('CreatedAt')
    return(local_tweets.groupby(pd.Grouper(freq='1Min')).apply(WeightedSentiment))
```

7: Fetching crypto prices

- **CryptoCompare API** — tool to fetch cryptocurrencies
- Below code snipped shows the function to retrieve the Bitcoin
- Limitations: can only get 2000 most recent cryptocurrencies
- Our script automatically matches the timeframe of the crypto prices to the one of the tweets

Relevant part in the code

```
def FetchCryptoprices(df):

   num = df.size
    recent = df.index[-1].to_pydatetime()+timedelta(minutes=1)
    num_of_iterations = math.ceil(num/2000)
    last_limit = num-(num_of_iterations - 1)*2000
    btc_prices = pd.DataFrame(columns=['time','close'])

    for i in list(range(num_of_iterations)):
        btc_time  = []
        btc_high  = []
        btc_low   = []
        btc_open  = []
        btc_close = []

        if i==0:
          btc = cryptocompare.get_historical_price_minute('BTC', 'USD', limit=2000,
                                                          exchange='CCCAGG', toTs=recent)
```

8: Calculating returns, volatility and their derivative

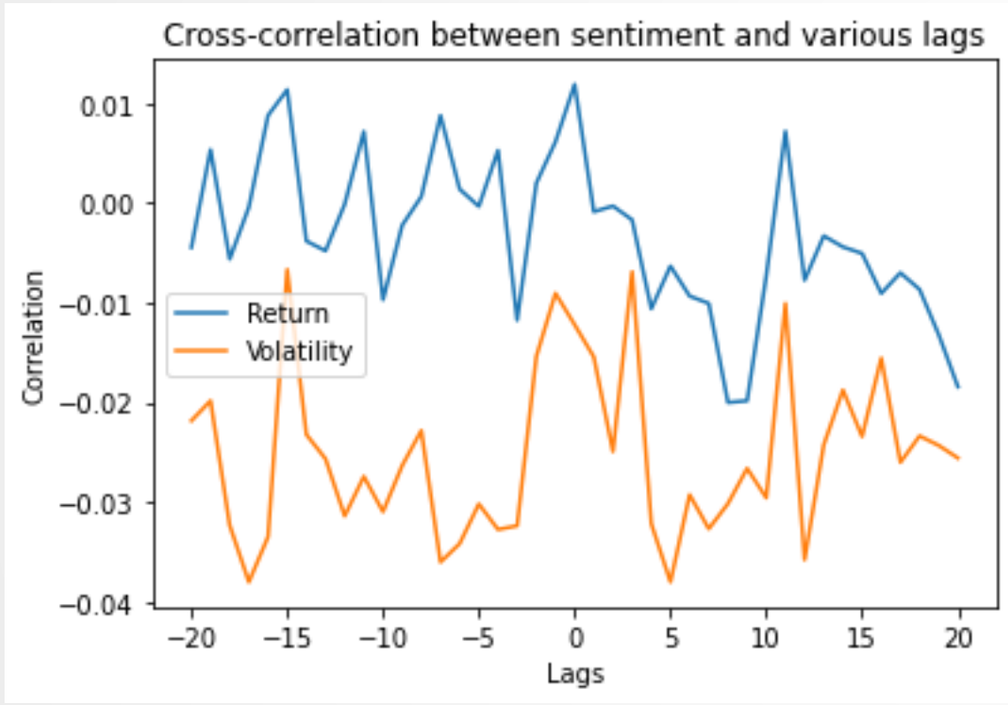
- We unify the data frame and calculate return and volatility measures
- Returns — logarithmic function
- Why derivatives? — we want to correlate the increase or decrease of the cryptocurrency with the increase or decrease in sentiment
  - When the derivative of crypto currency is a peak » the increase is maximum
  - When the derivative of crypto currency is flat » the crypto has stabilized
  - We use the np.gradient() function to calculate derivative

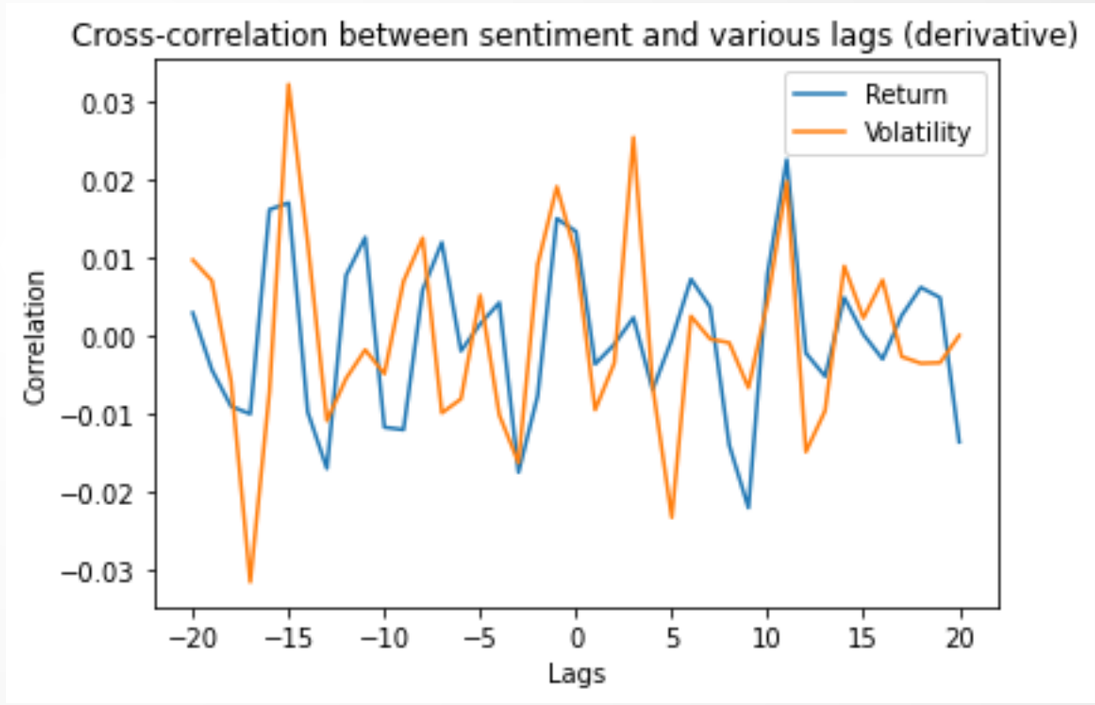
Relevant part in the code

```
def CreateDerivative(df):

    local_df = df.copy()
    for column in local_df.columns:
        local_df[column] = np.gradient(local_df[column].values)
    return(local_df)

def FinalizeData(sentiment, prices):

    sentiment = sentiment.iloc[::-1].tz_convert(None)
    final_dataframe = pd.concat([sentiment,prices], axis = 1)
    final_dataframe.columns = ['Sentiment', 'Close']
    final_dataframe['Return'] = np.log(final_dataframe.Close)
                                      - np.log(final_dataframe.Close.shift(1))
    final_dataframe['Vola'] = final_dataframe["Return"] * final_dataframe["Return"]
    return(final_dataframe)
```

9: Calculating cross-correlation between variables

- **Cross-correlation** — adds a lags which permit to shift one of the time series left or right to possibly find a better correlation
  - We use lags of ±20 minutes
- This is coherent with our problem as the changes in returns and volatility may come before or after the tweet's sentiments.





- Conclusion for the sample data:
  - In the sample data period there is no significant correlation between any of the variables, but this may change at any of the runs of the code as the most recent tweets are scraped every time

9: Calculating cross-correlation between variables

Relevant part in the code

```
  def CreateCorrelplots(df, name):

      local_df = df
      for i in np.arange(-20,21):
          return_name = "Retur_"+str(i)
          volat_name = "Volat_"+str(i)
          local_df[return_name] = local_df['Return'].shift(i)
          local_df[volat_name] = local_df['Vola'].shift(i)
      corre_matrix = local_df.corr()
      correllations = pd.DataFrame(index=np.arange(-20,21),
                                   columns=["Return", "Volatility"])
      for i in correllations.index:
          correllations['Return'][i] = corre_matrix['Retur_'+str(i)]['Sentiment']
          correllations['Volatility'][i] = corre_matrix['Volat_'+str(i)]['Sentiment']
      correllations.plot(title="Cross-correlation"+name)
```

Extra: We have also included in the script a function that creates scagnostics

Relevant part in the code
See function create_scagnostics in the code!

- **Scagnostics** —We include the calculation of the scagnostics measures for the following variables
  - Sentiment
  - Closing price
  - Returns
  - Volatility