COVID-19 Quarantine Tweets Sentiment Analysis using Knuth-Morris-Pratt Algorithm

Faris Muhammad Kautsar 13518105 (*Author*)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): farismuhammad8201@gmail.com

***Abstract*—The pandemic of COVID-19 had drastically altered human interactions and activities, with the scope and impact that might be approximately gauged through social media activities. Assessing the sentiments expressed within these social media activities might be one of the factors considered in making policy responses in the pandemic. In this paper the author will demonstrate a simple sentiment analysis of tweets made in context of COVID-19 quarantine using the pattern-matching Knuth-Morris Pratt algorithm.**

Keywords—COVID-19; Knuth-Morris-Pratt; String-Matching; Sentiment Analysis; Opinion Mining

I. INTRODUCTION

Few had anticipated in the New Year's Eve of 2020 that in a few months, human activities and interactions will be drastically overhauled, transformed, overturned, and global economy crippled by the fear of a contagious global outbreak. Although SARS-CoV-19 was first identified in Wuhan in December 2019, very few recognized the microbe's catastrophic potential, even among national governments and esteemed international institutions.

Within the space of approximately five months since the strain's first identification, the virus had infected at least over three million individuals globally, killing hundreds of thousands. Nearly two months has passed since it is declared as a global pandemic. The virus had since shifted away from its native homeland in the central Chinese Mainland, ravaging picturesque tourist spots of Spain and northern regions of Italy, shutting down the global financial center of New York City, putting billions worldwide into *de facto* house arrest as governments scramble into increasingly draconian measures to prevent uncontrollable outbreak in their territory.

It is safe to say that the pandemic had changed the rhythm of human activities like it had never been. The freefall of stock exchanges, unemployment spikes, and complete shutdown of once-bustling public spaces and commercial establishments gave obvious hints on the ongoing new normal. Of course, it is possible that the transformation is mere temporary readjustment. It is highly likely that life will return to normalcy immediately as the virus recedes through one way or another: pressured into extinction through herd immunity, vanquished by new vaccines, or merely having its outbreak cycle halted through aggressive lockdown. Which policy is the most prudent is, of course, heavily contested by many.

In fact, the role of policy—especially responses from central national governments and international agencies—is also very important in the current trajectory of events. As humans are now living wealthier than they have ever been in history, the governments have more resources than ever in employing the means necessary to deal with the pandemic as they see fit. Past pandemics faced much weaker, poorer governments, incapable to do anything against what was practically a wrathful act of God. Current governments are more capable than ever to stare the eye of the storm.

The problem is, as the world has barely passed the first wave of the pandemic's infection, no one has certain idea about the actual, overall effect of any of the policies, let alone assessing their unintended consequences.

For example, as most governments embark on limiting activities outside home, it remains unknown whether this significantly affects the psychological health of their citizens one way or another. Of course, this should not be the only thing taken into account in policy decisions—plenty of things, such as the probability of uncontrollable outbreak in itself, also matters. However, it could be factored into one of the variables considered when undertaking a pandemic policy.

There are several ways to identify the state of psychological health during "quarantine" or "lockdown" policy, although each has their own flaws. In this paper, we will analyze popular sentiments during COVID-19 isolation through crude sentiment analysis using Knuth-Morris-Pratt algorithm.

II. FUNDAMENTAL THEORY

A. Twitter

Twitter is a microblogging and social networking service created in 2006 by Jack Dorsey, Noah Glass, Biz Stone, and Evan Williams[1]. Today, Twitter is one of the most popular social media platforms available, with over 100 million daily users and 500 million tweets daily.

Within Twitter, users might send and interact with short messages known as "tweets". Tweets were restricted up to 280 characters, up to 140 characters prior to 2017. Registered users might interact with the tweets in various ways: post, like, or

retweet, but unregistered users might only read tweets from non-protected accounts.

For its implementation, Twitter relies mostly on open-source software. Twitter web interface uses Ruby on Rails framework, deployed on a performance-enhanced Ruby Enterprise Edition implementation of Ruby. The service's application programming interface (API) also allows other web services and applications to integrate with Twitter.

Users might access Twitter through its website interface, Short Message Service (SMS), or its mobile application.

B. Sentiment Analysis

Sentiment analysis is contextual mining of text which identifies and extracts subjective information in source material using natural language processing, text analysis, computational linguistics, biometrics, and other such methods[2]. It analyzes people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards certain entities[3].

There are also terms that are broadly similar in definition such as *sentiment analysis, opinion mining, opinion extraction, sentiment mining, subjectivity analysis, affect analysis, emotion analysis, review mining,* etc. They generally fall under the same umbrella as sentiment analysis, and this is the term that is more commonly used.

In general, sentiment analysis has been instigated at three main levels:

- Document level, classifying whether a whole opinion document representing a positive or negative sentiment. This particular task is known as *document-level sentiment classification*.
- Sentence level, determining whether each sentence expressed opinion that can be roughly summarized into three categories: positive, negative, or neutral.
- Entity and aspect level, also known as feature level (feature-based opinion mining and summarization). Performing finer-grained analysis, such that the assessment is directed directly at the opinion itself instead of the language structure. The goal of this level of analysis is to discover sentiments on entities and/or their aspects.

Broadly speaking, there are two kinds of opinions. Regular opinions express a sentiment only on an particular entity or an aspect of the entity. Comparative opinions, meanwhile, compare multiple entities based on some of their shared aspects.

C. String-Matching Algorithm

String-matching algorithm is an algorithm intended to search a certain pattern or string inside a larger pattern or text[4]. The string-matching problem is the problem of finding all valid shifts with which a given pattern P occurs in a given text T.

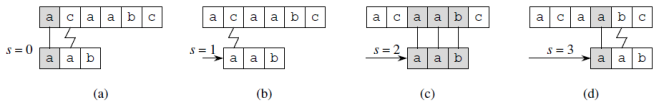

Figure 1. Source: Cormen et al., Introduction to Algorithms.

The most basic, brute-force algorithm to resolve the string-matching problem is often called the naïve algorithm (see Fig. 1). The naive algorithm finds all valid shifts using a loop that checks the condition P[1 . .m] = T [s + 1 . . s + m] for each of the n − m + 1 possible values of s. The naive string-matching procedure can be interpreted graphically as sliding a "template" containing the pattern over the text, noting for which shifts all of the characters on the template equal the corresponding characters in the text. Procedure naive string-matcher takes time O((n − m + 1)m), and this bound is tight in the worst case.

For obvious reason, the naïve algorithm is not the optimal procedure for this problem.

D. Knuth-Morris-Pratt Algorithm

One of the algorithms that might be applied to solve string-matching problems is the Knuth-Morris-Pratt algorithm, often abbreviated as the KMP algorithm. The KMP algorithm is an algorithm that uses preprocessing approach of patterns to avoid trivial comparisons. Unlike the brute force or naïve algorithm, it avoids recomputing matches. It compares string from left to right, and uses linear time for exact matching.

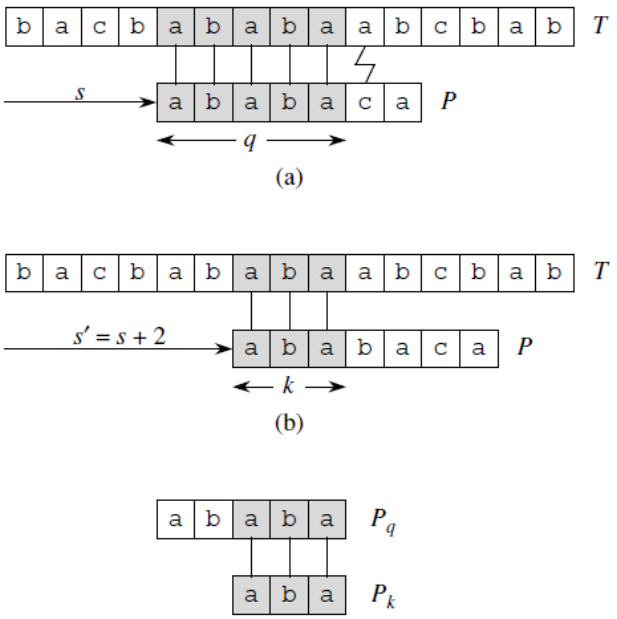

Figure 2. Source: Cormen et al., Introduction to Algorithms.

The Knuth-Morris-Pratt algorithm utilizes something known as the prefix function (see Fig. 2). The prefix function π for a pattern encapsulates knowledge about how the pattern matches against shifts of itself. This information can be used to avoid testing useless shifts in the naive pattern-matching algorithm or to avoid the precomputation of δ for a string-matching automaton.

The Knuth-Morris-Pratt algorithm is given in the pseudocode as the following. The first is the main KMP-Matcher algorithm:

```
KMP-MATCHER(T, P)
   1. n ← length[T ]
   2. m ← length[P]
   3. π ← COMPUTE-PREFIX-FUNCTION(P)
   4. q ← 0 ✄ Number of characters
      matched.
   5. for i ← 1 to n ✄ Scan the text from
      left to right.
   6. do while q > 0 and P[q + 1]  = T [i
      ]
   7. do q ← π[q] ✄ Next character does
      not match.
   8. if P[q + 1] = T [i ]
   9. then q ← q + 1 ✄ Next character
      matches.
   10. if q = m ✄ Is all of P matched?
   11. then print "Pattern occurs with
       shift" i − m
   12. q ← π[q] ✄ Look for the next match.
```

The next algorithm to be implemented in the pseudocode is the COMPUTE-PREFIX-FUNCTION, which is the border/prefix function intended to calculate the longest prefix which is also the suffix in the pattern.

```
COMPUTE-PREFIX-FUNCTION(P)
   1. m ← length[P]
   2. π[1] ← 0
   3. k ← 0
   4. for q ← 2 to m
   5. do while k > 0 and P[k + 1]  = P[q]
   6. do k ← π[k]
   7. if P[k + 1] = P[q]
   8. then k ← k + 1
   9. π[q] ← k
   10. return π
```

The running time of COMPUTE-PREFIX-FUNCTION is O(m). Since the number of outer-loop iterations is O(m), and since the final potential function is at least as great as the initial potential function, the total actual worst-case running time of COMPUTE-PREFIX-FUNCTION is O(m). Similarly, the running time of KMP-MATCHER is O(n). Therefore, the running time of KMP algorithm overall is O(m+n).

III. IMPLEMENTATION

This section will explain the strategy and implementation of *Opinion Mining* for COVID-19 quarantine tweets using Knuth-Morris-Pratt algorithm written in Python language.

A. Preparation

For the preparation, Tweepy package for Python must be downloaded through pip command like the following:

```
pip install tweepy
```

This is necessary as Tweepy is the Python client for the official Twitter API.

In order to fetch tweets through Twitter API, an user needs to register an app through their twitter account. This is done through accessing https://developer.twitter.com/en/apps and afterwards filling the application details in order to create an app.

After the app has been created, the user then might access the app's consumer key, consumer secret key, access token, and secret access token. These will be used in the next step.

B. Sentiment Keywords and File-Reading

The next step is designing positive and negative keywords to gauge the emotional level of a certain tweet. This will be implemented as file .txt, and for the sake of simplicity will exclude factors such as double negatives and sarcasm, thereby relying on the fundamental assumption that the tweeter tweets their opinion in straightforward manner. The keywords are implemented in English.

The first is the positive keywords, written in the file positive.txt, as follows.

```
fun
enjoy
love
relax
glad
happy
nice
good
merry
jolly
ecstatic
```

Next is the negative keywords, keywords which express the tweeter's negative emotion. They're written in negative.txt, as follows.

```
horrible
awful
bad
terrible
annoyed
sad
stuck
mad
angry
trapped
worst
```

Finally, all files involved will be read into a readFile method, which receives the parameter of the file's directory

before converting the file of .txt format into an array of lines. The method is as follows.

```
def readFile(filepath):
        f = open(filepath, 'r')
        x = f.readlines()
        f.close()
        return x
```

C. Knuth-Morris-Pratt Algorithm

The next step of the implementation is the code for Knuth-Morris-Pratt Algorithm, the pattern-matching algorithm applied in this case.

Broadly speaking, the KMP algorithm can be divided into the main function and the border function. The border function's role is to determine the size of the largest prefix in the text that is also the largest suffix in that text.

Below is the implementation of the KMP main function:

```
def KMPSearch(pat, txt):
    m = len(pat)
    n = len(txt)
    fail = self.computeFail(pat)
    i = 0
    j = 0
    while (i < n):
       if (pat[j].casefold() ==
txt[i].casefold()):
           print(pat[j].casefold())
           if (j == m - 1):
               return i - m + 1
               i+=1
               j+=1
           elif (j > 0):
               j = fail[j-1]
           else:
                i+=1
        return -1
```

The next is the implementation of the KMP border function:

```
def computeFail(self, pat):
    fail = [0]*len(pat)
    fail[0] = 0
    m = len(pat)
    j = 0
    i = 1
    while (i < m):
         if (pat[j].casefold() ==
pat[i].casefold()):
             fail[i] = j + 1
             i+=1
             j+=1
             elif (j > 0):
                 j = fail[j-1]
             else:
                 fail[i] = 0
                 i+=1
   return fail
```

D. Twitter API

The next step is building the Twitter client to acquire tweets that will be analyzed by the sentiment analyzer[5]. The first preparation for this step is ensuring that Tweepy package is already imported, as the following. Regex will also need to be imported in order to clean the tweets. Regex already had a built-in package in Python which could be called by 're'.

```
import tweepy
import re
```

The next step is creating a class of a Twitter client, which contains all the methods necessary for sentiment analysis (including the KMP algorithm above). The client will have an attribute of an array of positive keywords and of negative keywords, respectively. Also, the client will be initialized through the consumer key, consumer secret key, access token, and secret access token that previously had been saved when the app is created. Then the initializer will also attempt to authenticate these keys and tokens.

```
class TwitterClient(object):
    pospat = []
    negpat = []
    def __init__(self):
        consumer_key = XXXX
        consumer_secret = XXXX
        access_token = XXXX
        access_token_secret = XXXX
        try:
         self.auth =
OAuthHandler(consumer_key,
consumer_secret)
self.auth.set_access_token(access_token,
access_token_secret)
         self.api = tweepy.API(self.auth)
         except:
            print("Error: Authentication
Failed")
```

The next method to create inside the client is a tweet cleaner. Using regular expression, this will remove links and special characters for the tweet so that it may be processed in a more simplified manner.

```
   def clean_tweet(self, tweet):
        return ' '.join(re.sub("(@[A-Za-
z0-9]+)|([^0-9A-Za-z\t])|(\w+:\/\/\S+)", "
", tweet).split())
```

The next method to create is intended to return certain sentiments expressed inside the tweet according to the parameters already defined (namely, positive and negative keywords).

The method is implemented as follows: the text will iterate through each keyword, searching whether they contain the positive or negative keyword. If the word is acquired, then the tweet's sentiment will be set depending on whether the word is positive or negative. If none exists, then it will be categorized as neutral instead. The sentiment is returned in the form of a string.

The implementation in Python is as the following.

```
def get_tweet_sentiment(self, tweet):
    post = -1
    neg = -1
```

```
    text = self.clean_tweet(tweet)
    for i in range(len(self.pospat)):
        checkpost =
self.KMPSearch(self.pospat[i], text)
        if (checkpost != -1):
            post = checkpost
            break
        if (post == -1):
            for j in
range(len(self.negpat)):
            checkneg =
self.KMPSearch(self.negpat[j], text)
            if (checkneg != 1):
                neg = checkneg
                break
    if (post != -1):
            return 'positive'
    elif (neg != -1):
            return 'negative'
    else:
        return 'neutral'
```

The next is the function to fetch and parse tweets. The function will call Twitter API to fetch tweets, storing them in an empty array. Then it will parse tweets acquired one-by-one. As the tweets are being parsed, it will acquire the tweet's full text as well as the sentiment acquired from that tweet through get_tweet_sentiment() function.

The full implementation of the function is as follows.

```
def get_tweets(self, query, count = 10):
    tweets = []
    try:
        fetched_tweets = self.api.search(q
= query, count = count,
tweet_mode='extended')
        for tweet in fetched_tweets:
            parsed_tweet = {}
            parsed_tweet['text'] =
tweet.full_text
            parsed_tweet['sentiment'] =
self.get_tweet_sentiment(tweet.full_text)
            if tweet.retweet_count > 0:
                if parsed_tweet not in
tweets:
                    tweets.append(parsed_tweet)
            else:
   tweets.append(parsed_tweet)
                return tweets

    except tweepy.TweepError as e:
        print("Error : " + str(e))
```

Last but not least is the implementation of the main() function, which will run when the program is called. The main will read positive.txt and negative.txt to the global array attributes. For query used for Twitter search, we will use 3 of most popular COVID-19 hashtags: #pandemicin5words, #stayhome, and #quarantineandchill.

After entering the query, the result will display the percentage of positive, negative, and neutral tweets, respectively. Also, the result will also print select chosen tweets that are perceived to be positive and negative.

The implementation in Python is as the following.

```
def main():
    api = TwitterClient()
    api.pospat =
api.readFile("positive.txt")
    api.negpat =
api.readFile("negative.txt")
    print("Enter your query: ")
    tweets = api.get_tweets(query =
"#stayhome", count = 100) +
api.get_tweets(query =
"#quarentineandchill", count = 100) +
    api.get_tweets(query =
"#pandemicin5words", count = 1000)
    ptweets = [tweet for tweet in tweets
if tweet['sentiment'] == 'positive']
    print("Positive tweets percentage: {}
%".format(100*len(ptweets)/len(tweets)))
      ntweets = [tweet for tweet in
tweets if tweet['sentiment'] ==
'negative']
    print("Negative tweets percentage: {}
%".format(100*len(ntweets)/len(tweets)))
    print("Neutral tweets percentage: {}
%".format(100*(len(tweets) - len(ntweets)
- len(ptweets))/len(tweets)))
    print("\n\nPositive tweets:")
      for tweet in ptweets[:10]:
          print(tweet['text'])
    print("\n\nNegative tweets:")
      for tweet in ntweets[:10]:
        print(tweet['text'])
```

At last, the main function could be initialized.

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

IV. RESULT

Below is the percentage-point result of the sentiment analysis:

```
C:\Users\Faris>py sentimentassess.py
Positive tweets percentage: 32.27513227513227 %
Negative tweets percentage: 9.523809523809524 %
Neutral tweets percentage: 58.2010582010582 %
```

Followed by a sample of positive tweets:

```
Positive tweets:
What a treat from @nancygilesnyc @CBSNews Thanks f
or this fun look at dressing to cheer your spirits
 as you… https://t.co/mVooHjNWyT
Well it's Sunday fun-day. How's the end of everyon
e's quarantine going? #selfie #SelfieSunday #Quare
ntineLife… https://t.co/GooBPBDCRI
#quarentineandchill #quarantine #drunk #high #wait
forit #waituntiltheend #funny #covid19 #coronaviru
s… https://t.co/YBe1ZFATii
Fun with the dogs outside. #quarentineandchill #qu
arentinelife #quarentinedogs #rescuedogs… https://
t.co/Wcfnv0pFME
```

And then the negative ones.

```
Negative tweets:
RT @theory_football: 4v3 to end line/goal;
- Goalkeeper dribble to draw defenders
- Disguise of movement/pass
- Blind side runs off defende…
RT @KPMG_NG: The country's National budget for 202
0 is not left out of the harsh realities we have t
o deal with in the wake of the #coronav…
RT @JordanSchachtel: Ban cars
Ban alcohol
Ban unhealthy food
Ban dangerous jobs
Ban sports
Ban travel
Ban electricity
Ban human contact
```

V. CONCLUSION

The pattern-matching Knuth-Morris-Pratt algorithm could be used to construct a crude sentiment analysis program for tweets using mere word-identification that is string-matched into the tweets. The end result the author acquires in general is that positive sentiment is more commonly expressed than negative ones, while most of the sentiments expressed are neutral. However, this is not without its flaws.

One of the reason why positive sentiments might be more likely to be expressed might be the choosing of the queries. It is possible that users using the queries searched in the program are more likely to express positive sentiments during their quarantine/stay-at-home/lockdown. For this reason, in the future it might be possible to select for better queries, or allow query input directly to the program so that different sentiments might be assessed differently with different queries.

Another thing to consider is that mere word-identification is generally far from being sufficient in expressing sentiments. Words might be preceded by negations that express the opposite sentiment overall ("This isn't bad at all" will be counted as a negative sentiment in this program, for example). Therefore, the algorithm needs more fine-grained, comprehensive analysis to produce more accurate results.

Regardless, sentiment analysis of COVID-19 quarantine tweets might express general public psychological health and opinion on the crisis, which might be useful for policy-making in response to this pandemic. This paper is intended, at the very least, to be a stepping stone for this scheme.

VIDEO LINK AT YOUTUBE

https://youtu.be/o_w3xkvwEhQ

ACKNOWLEDGMENT

The author would like to thank God for His blessings and guidance in allowing the author to finish the paper. The author would also thank Dr. Ir. Rinaldi Munir, M.T., as the lecturer of Algorithm Design (IF2211), the author's family, and friends for their support in the making of this paper.

REFERENCES

[1] https://esrc.ukri.org/research/impact-toolkit/social-media/twitter/what-is-twitter/

[2] https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-6c94d6f58c17

[3] Bing Liu. "Sentiment Analysis and Opinion Mining", Morgan & Claypool Publishers, May 2012.

[4] Cormen, Thomas; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford. "Introduction to Algorithms (Second ed.)". MIT Press and McGraw-Hill, 2001.

[5] https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2020

Faris Muhammad Kautsar 13518105