Controlling social media content

Raja Sengupta
Professor, Dept. of Civil Engineering
University of California, Berkeley
Berkeley, CA

Bougrine Elias
Student, Dept. of Civil Engineering
University of California, Berkeley
Paris, France
elias_bougrine@berkeley.edu

Loo Marc-Antoine
Student, Dept. of Civil Engineering
University of California, Berkeley
Paris, France
marcantoine.loo@berkeley.edu

Peng Xin
GSI
University of California, Berkeley
Berkeley, CA

***Abstract*—Controlling social media content to prevent extreme content to be spread or to avoid insults to be posted could have a really positive impact in our society. This is why for our project, we decided to focus on this subject. In this report we will discuss about our program design, its implementation in python, and its complexity. Our program, which can control a text file, could be used in order to control posts on Facebook, Twitter, forums, or other types of website.**

I. INTRODUCTION (*HEADING 1*)

A. Controlling bad content

Nowadays, social medias are used by everyone and people can spend the main part of their day on them, to communicate, to learn, to watch videos and photos, and so on. Social medias are more than platforms where people can interact with each other, they are platforms where people can learn together, share feelings and develop their way of thinking and beliefs. Hence, social medias have a growing influence on everyone, and therefore, it is important to be able to control them. Controlling social media content is of the most importance in order to avoid extreme ideas to be shared to everyone. In our project, we will focus on bad content such as : racist, sexual, violent, extremist, terrorist and so forth.

Being able to control such content will lead to more secure social medias, and more balanced individuals. Therefore, we chose these following questions for the subject of our project: how to control bad content on social media? And how to create a program that can detect every inappropriate content posted?

B. Being able to solve this question has become critical

This question of controlling social medias is critical in our society, where extreme organizations are growing. In fact, most of the extreme organizations such as terrorist organizations or racist ones are developing their network and are enrolling people thanks to the use of social medias. In fact, lots of users, especially young users (12-18 years old) are very likely to be influenced by social medias, and are therefore the main targets of extreme organizations.

Furthermore, some researches have showed that people who are involved in extreme organizations on social media (racist group for example) are likely to become involved in them in real life. In fact, since extreme beliefs are very controlled in our society (but not in social medias), people are less likely to talk about them in public places, and thus social medias are the best places to expose their extreme ideas and feelings.

Therefore, controlling the content of social media can be very useful to reduce extreme beliefs and to fight against extreme organizations, especially terrorism. In addition, this control of information can be helpful to find who is likely to be involved in such organizations, and can be used to track terrorists, racists, and so forth.

To summarize, Controlling the content of social medias can have a real positive impact on our world, by dealing with issues that includes, but is not limited to:

- Fight against extreme organizations (especially terrorism)
- Finding who is likely to be involved in such organizations (via controlling social posts)
- Tracking terrorists, racists, and so forth (through the analysis of comments, private message, post, pictures, linked shared, likes,...)
- Prevent extreme beliefs to grow (by deleting harmful contents posted such as comments, posts, video, pictures, linked, or by preventing them to be posted)
- Prevent attacks (through message encrypted analysis)
- To create more balanced individuals (especially teenagers)
- Prevent extreme organizations to grow and develop their network by enrolling people through the use of social medias.

To solve these important questions we will create a program that will enable us to detect inappropriate content in order to make social media more secure.

In this project we will focus on two kinds of content.

First, we will focus on explicit content, not hidden, and using clear words. To do so, we will scan all the content on the

social media (posts/ articles/ videos) and we will analyze this content. By looking for specific words that are very likely to be associated with extreme beliefs and ideas, we will be able to control bad content.

Then, we will also focus on a second type of content: hidden content, not explicit, using encrypted messages (cryptography).

Cryptography allows people to store confidential information or transmit it over unsecured networks (such as social media) so that, no one other than the recipient can read it. A cryptographic algorithm is a mathematical function used during the encryption and decryption process. This algorithm is associated with a specific key, used to encrypt the text. The security of encrypted messages relies on two elements: the invulnerability of the cryptographic algorithm and the confidentiality of the key. In fact, lots of extreme organizations are interacting using encrypted messages, and thus, being able to decrypt them is of the most importance for security purpose.

For our project, we will focus on common cryptographic algorithms. We will analyze all encrypted messages with each of the most common algorithms, in order to find the real meaning of such contents. Furthermore, in order to have a real impact, we will not only try to control English contents, but also foreign ones.

II. Complexity

The complexity is different for the two programs.

A. *Explicit content*

The program for the explicit content is by far the fastest with the lowest complexity. In fact, since the program will just check if the text file contains some bad contents, without having to translate it first.

The computation time for this program depends on the length of the original message: the longer the text file is, the longer it will take to be analyzed. Since our program processes word by word, each one at a time to see if the words are in the different CSV that contain potential "bad" words, its complexity runtime is equal to: T(n) = O(n). The time grows linearly as input size increases.

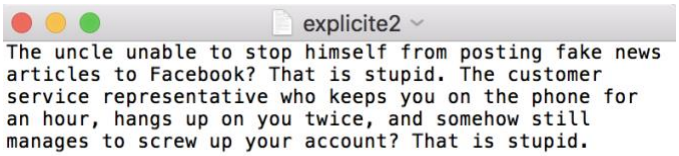
This program runs in linear time, because its time execution is directly proportional to the input size (our text file in our case).

Please, find below an example of our Input and Output.

As we can see in the shell returned, we focused on two different content:

- Insults
- Potential dangerous content (e.g. terrorism related)

Input text file:

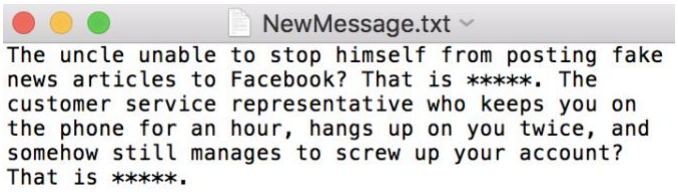


Shell returned by the program:

```
>>> (executing file "290 project.py")
The txt file does not contain a potential extreme content from the dictionnary.
('The txt file contains an insult:', ['stupid', 'stupid'])
We will remove this insult
('The computation time is:', 0.0008120536804199219)
```

We can see that the computation time for this part is very good.

New text file created by the program:



B. *Encrypted messages*

The complexity of the program for encrypted messages is bigger than for explicit content. We can divide the program for the encrypted messages into two part : the first one being the decryption using Caesar Cipher algorithm, the second one being the decryption using Vigenere Cipher.

a) Caesar Cipher

The complexity for this program is equal to 26. In fact, the algorithm is built in order to test all shifts possible until we found a decrypted message in English. Therefore, since there are 26 letters in the alphabet, the shift is within 0 and 26. The computation time depends here on the level of the shift. The higher the shift is, the longer it will take to solve the program, because the program will first try all the smaller shift by increasing until finding the good one.

Please, find below an example of our input and our output.

Below is the input, which is a text file, with an encrypted message. This can of message could be used in private message on Facebook for example.

Caesar2

```
sx polbekbi, wb. tkdd, grsvo sx zyvsmo mecdyni,
zobceknon yppsmobc drkd ro gkc csmu kxn xoonon dkusxq dy
k qyfobxwoxd ryczsdkv. wsxedoc kpdob ro kbbsfon, dgy
wsvsdkxdc kvbokni zycsdsyxon sx dro ryczsdkv cryd kxn
usvvon rsc qekbnc gsdr k qex, kxn wb. tkdd czon kgki yx
k wydybmimvo. sd gkc yxo yp dro wycd nbkwkdsm tksv
lbokuc sx bomoxd ukcrwsbs wowybi.
```

Below is the output obtained (from the shell). The program will separate all the words of the initial message after decrypting it, in order to find if it contains any words with potential dangerous content (listed in one of the two CSV file, the other one containing insults).

```
>>> (executing file "290 project fin.py")
('This message has been crypted with the Caesar method. Here
is the key:', 16)
('The message is:', ['in', 'february', 'mr', 'jatt', 'while',
'in', 'police', 'custody', 'persuaded', 'officers', 'that', '
he', 'was', 'sick', 'and', 'needed', 'taking', 'to', 'a', 'go
vernment', 'hospital', 'minutes', 'after', 'he', 'arrived', '
two', 'militants', 'already', 'positioned', 'in', 'the', 'hos
pital', 'shot', 'and', 'killed', 'his', 'guards', 'with', 'a'
, 'gun', 'and', 'mr', 'jatt', 'sped', 'away', 'on', 'a', 'mot
orcycle', 'it', 'was', 'one', 'of', 'the', 'most', 'dramatic'
, 'jail', 'breaks', 'in', 'recent', 'kashmiri', 'memory'])
('The txt file contains a potential extreme content. Here are
the words potentially dangerous:', ['gun'])
('The computation time is:', 4.7145061492919921)
```

After decrypting and separating the words, the program will process as it will do for explicit message : find if it contains dangerous content. After analyzing the decrypted message, the program returns if yes or not the text file contains any potential dangerous content.

Here, the computation time is approximately 4.7 seconds, which shows well that decrypted messages take way more time to be controlled.

b) Vigenere Cipher

The complexity for this program depends on the composition of the key, and on its length. The length of the key is defined as the number of letters in the key. In fact, due to time limitation for our project, we only focused on a key of length up to 3: 1, 2 or 3.

When the key length is equal to 1, the algorithm Vigenere Cipher is equivalent to the algorithm of Ceasar Cipher. Hence, there is no need to do the case where length is equalt to 1 since it is already covered by Caesar Cipher. In addition, the number of loops for this algorithm is 26.

In other cases, with n=length of the key, we have a complexity fit Vigenere Cipher of : 26^n, so therefore we have an exponential complexity runtime, which is is equal to: T(n) = O(26^n). The computation time grows linearly as input size increases and as the length of the key increases.

In fact, in order to find the best decryption of our input, the program will run every cases, and see which output is “the more” English.

The computation time depends not only on the length of the input, but also on the letters composing the key. In fact, the program will run every case in order to find the solution, by trying the list on the top right.

```
>>> (executing file "290 project fin.py")
['aa', 'aaa', 'aab', 'aac', 'aad', 'aae', 'aaf', 'aag', 'aah', 'aai', 'aaj', 'aak',
'aal', 'aam', 'aan', 'aao', 'aap', 'aaq', 'aar', 'aas', 'aat', 'aau', 'aav', 'aaw',
'aax', 'aay', 'aaz', 'ab', 'aba', 'abb', 'abc', 'abd', 'abe', 'abf', 'abg', 'abh', '
abi', 'abj', 'abk', 'abl', 'abm', 'abn', 'abo', 'abp', 'abq', 'abr', 'abs', 'abt', '
abu', 'abv', 'abw', 'abx', 'aby', 'abz', 'ac', 'aca', 'acb', 'acc', 'acd', 'ace'
, 'acf', 'acg', 'ach', 'aci', 'acj', 'ack', 'acl', 'acm', 'acn', 'aco', 'acp', '
acq', 'acr', 'acs', 'act', 'acu', 'acv', 'acw', 'acx', 'acy', 'acz', 'ad', 'ada'
, 'adb', 'adc', 'add', 'ade', 'adf', 'adg', 'adh', 'adi', 'adj', 'adk', 'adl', '
adm', 'adn', 'ado', 'adp', 'adq', 'adr', 'ads', 'adt', 'adu', 'adv', 'adw', 'adx
', 'ady', 'adz', 'ae', 'aea', 'aeb', 'aec', 'aed', 'aee', 'aef', 'aeg', 'aeh', '
aei', 'aej', 'aek', 'ael', 'aem', 'aen', 'aeo', 'aep', 'aeq', 'aer', 'aes', 'aet
', 'aeu', 'aev', 'aew', 'aex', 'aey', 'aez', 'af', 'afa', 'afb', 'afc', 'afd', '
afe', 'aff', 'afg', 'afh', 'afi', 'afj', 'afk', 'afl', 'afm', 'afn', 'afo', 'afp
', 'afq', 'afr', 'afs', 'aft', 'afu', 'afv', 'afw', 'afx', 'afy', 'afz', 'ag', '
aga', 'agb', 'agc', 'agd', 'age', 'agf', 'agg', 'agh', 'agi', 'agj', 'agk', 'agl
', 'agm', 'agn', 'ago', 'agp', 'agq', 'agr', 'ags', 'agt', 'agu', 'agv', 'agw',
'agx', 'agy', 'agz', 'ah', 'aha', 'ahb', 'ahc', 'ahd', 'ahe', 'ahf', 'ahg', 'ahh
', 'ahi', 'ahj', 'ahk', 'ahl', 'ahm', 'ahn', 'aho', 'ahp', 'ahq', 'ahr', 'ahs',
'aht', 'ahu', 'ahv', 'ahw', 'ahx', 'ahy', 'ahz', 'ai', 'aia', 'aib', 'aic', 'aid
', 'aie', 'aif', 'aig', 'aih', 'aii', 'aij', 'aik', 'ail', 'aim', 'ain', 'aio',
'aip', 'aiq', 'air', 'ais', 'ait', 'aiu', 'aiv', 'aiw', 'aix', 'aiy', 'aiz', 'aj
', 'aja', 'ajb', 'ajc', 'ajd', 'aje', 'ajf', 'ajg', 'ajh', 'aji', 'ajj', 'ajk',
'ajl', 'ajm', 'ajn', 'ajo', 'ajp', 'ajq', 'ajr', 'ajs', 'ajt', 'aju', 'ajv', 'aj
w', 'ajx', 'ajy', 'ajz', 'ak', 'aka', 'akb', 'akc', 'akd', 'ake', 'akf', 'akg',
'akh', 'aki', 'akj', 'akk', 'akl', 'akm', 'akn', 'ako', 'akp', 'akq', 'akr', 'ak
s', 'akt', 'aku', 'akv', 'akw', 'akx', 'aky', 'akz', 'al', 'ala', 'alb', 'alc',
'ald', 'ale', 'alf', 'alg', 'alh', 'ali', 'alj', 'alk', 'all', 'alm', 'aln', 'al
o', 'alp', 'alq', 'alr', 'als', 'alt', 'alu', 'alv', 'alw', 'alx', 'aly', 'alz',
'am', 'ama', 'amb', 'amc', 'amd', 'ame', 'amf', 'amg', 'amh', 'ami', 'amj', 'am
```

Hence, thanks to this list, we can understand that if the key is “aaa”, the solution will be found faster than if the key is “ze” or “abc”, because for “aaa”, the algorithm has less combination to try before finding the good key.

Please, find below an example of our input and our output.

Below is the input, which is a text file, with an encrypted message using Vigenere Cipher

Vigenere3

```
Kty mwi eyybnh yd rwmqsh
kty emsgqh tfzq itvw ytvq
ejvutyeqc uswfjep ak
bqeknrs inxt ktyd szr
```

Below is the output returned by the shell:

```
>>> (executing file "290 project fin.py")
('This message has been crypted with the Vigenere method.
Here is the key:', 'mfe')
('The message is:', ['you', 'are', 'stupid', 'my', 'friend
', 'you', 'should', 'have', 'work', 'more', 'seriously', '
instead', 'of', 'playing', 'with', 'your', 'gun'])
('The txt file contains a potential extreme content. Here
are the words potentially dangerous:', ['gun'])
('The computation time is:', 559.3391621112823)
```

As we can see, the output will first give us by which Cryptographic method the input text file has been encoded, here: Vigenere method. In addition, we also obtain the key and therefore its letter combination and its length.

Then the message is split into the different words in order to see if there are words that are classified as potentially dangerous (words listed by ourselves in one of the two CSV files).

Here, the computation time is approximately 560 seconds (approximately 9-10 minutes), which is very long, and shows how the length of the input, the length of the key and its combination affect the computation time of the program.

III. Limitations & improvement

In this program, there will always be one major problem: The complexity of the human language. In fact, one word can have different meaning, and even a same sentence can be interpreted differently depending on the context. Since our program does not consider the context of the words, we cannot detect inference and we can therefore wonder about the reliability or this controlling method. However, even if out program cannot be perfect, there are some ways to improve it.
The points or improvement include, but are not limited to:

- Implement this algorithm for different languages (by changing the dictionary).
- Use more cryptography algorithms rather than just two.
- Implement our program to be able to decrypt Vigenere Cypher message with longer key length.
- Add words/ sentences/ combination of words/ context in the two different CSV files "BAD" and "dangerous". The more our CSV files are complete, the more our program is reliable. In fact, we did not use any library for this project, we decided to built everything from scratch.
- To improve the computation time (especially for encrypted messages such as Vigenere), instead of decrypting all the message with every key, we could decrypt only the first 10 words of the message, and when the first 10 words decrypted contains a number of English words greater that a specific number arbitrary chosen, we decrypt the rest of the message. Otherwise, if the number of English words in the 10 first words decrypted is below the specified number desired, the program will not decrypt the rest, since the key used is very likely to not be the good one. By doing so, we could significantly decrease the number of decryptions done by the program, saving us a lot of time.

IV. Conclusion

Controlling social media content is a very difficult task due to the complexity of human language, and also the important amount of messages posted every day on social media. Even if our program does not enable us to control the content in real-time (due to the computation time of vigenere decryption), it enable us to easily control explicit content, which is the most common on social media, and which does not need a complex understanding of the context of the message since an insults is always a bad word, no matter its context. Therefore, regarding the control of bad words, we can say that our program is reliable and efficient (computation time very low). However, regarding the control of encrypted messages (especially Vigenere cipher and dangerous content), our program is less reliable and efficient, because it does not consider the context of messages, cannot infer meaning of sentences and its computation time is not optimal.

References

[1] "Vigenere Cipher" *Wikipedia*: https://en.wikipedia.org/wiki/Vigenère_cipher

[2] "Caesar Cipher" *Wikipedia*. https://en.wikipedia.org/wiki/Caesar_cipher

[3] "The Vigenere Cipher Encryption and Decryption": https://pages.mtu.edu/~shene/NSF-4/Tutorial/VIG/Vig-Base.html