Substitution Cipher

Partha Protim Paul

September 2019

1 Introduction

The Simple substitution cipher is one of the simplest ciphers, simple enough that it can usually be broken with pen and paper in a few minutes. On this report we will focus on automatic cryptanalysis of substitution ciphers, i.e. writing programs to solve these ciphers for us.

2 Process

We will be using a 'hill-climbing' algorithm to find the correct key. For this approach, we need a way of determining how similar a piece of text is to english text. This is called rating the 'fitness' of the text. A piece of text very similar to english will get a high score (a high fitness), while a jumble of random characters will get a low score (a low fitness). For this we will use a fitness measure based on quadgram statistics. This method works by first determining the statistics of english text, then calculating the probability that the ciphertext comes from the same distribution. An incorrectly deciphered (i.e. using the wrong key) message will probably contain sequences e.g. 'QKPC' which are very rare in normal english. In this way we can rank different decryption keys, the decryption key we want is the one that produces deciphered text with the highest likelyhood.

3 Hill-Climbing Algorithm

1. Generate a random key, called the 'parent', decipher the ciphertext using this key. Rate the fitness of the deciphered text, store the result.

2. Change the key slightly (swap two characters in the key at random), measure the fitness of the deciphered text using the new key.

3. If the fitness is higher with the modified key, discard our old parent and store the modified key as the new parent.

4. Go back to 2, unless no improvement in fitness occurred in the last 1000 iterations.

As this cycle proceeds, the deciphered text gets fitter and fitter, the key becomes better until either the solution appears, or, the solution is not found. In this case the run has failed and must be repeated with a different starting key. This means the hill-climbing algorithm is stuck in a 'local maximum', where there are no simple changes that can be made to the key to improve fitness, and yet it is not at the true solution. If this happens we can run the algorithm again with a different parent in the hope it may reach the true solution this time.

4 Example

We have the following ciphertext:

SOWFBRKAWFCZFSBSCSBQITBKOWL
BFXTBKOWLSOXSOXFZWWIBICFWUQLRXINO
CIJLWJFQUNWXLFBSZXFBTX
AANTQIFBFSFQUFCZFSBSCSB
IMWHWLNKAXBISWGSTOXLXTSWLUQ
LXJBUUWLWISTBKOWLSWG
STOXLXTSWLBSJBUUWLFULQR
TXWFXLTBKOWLBISOXSSOWTB
KOWLXAKOXZWSBFIQSFBRKANSOW
XAKOXZWSFOBUSWJBSBFTQRKAWSWANECRZAWJ

To begin the algorithm, we generate a random key, e.g.
plain alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
cipher alphabet: YBXONGSWKCPZFMTDHRQUJVELIA

and decipher the ciphertext using this key to get:
GDHMBRIZHMJLMGBGJGBSYOBIDH
XBMCOBIDHXGDCGDCMLHHYBYJMH
TSXRCYEDJYUXHUMSTEHCXMBGLCMBO
CZZEOSYMBMGMSTMJLMGBGJGBYN
HQHXEIZCBYGHFGODCXCOGHXTSXCU
BTTHXHYGOBIDHXGHFGODCXCOGHX
BGUBTTHXMTXSROCHMCXO
BIDHXBYGDCGGDHOBIDHXCZID
CLHGBMYSGMBRIZEGDHCZI
DCLHGMDBTGHUBGBM OSRIZHGHZEWJRLZHU

The fitness of our first plaintext attempt is -2304.04, something we should be able to improve on. By swapping 'Y' and 'B' in the key, and deciphering again we get a fitness of -2200.78, an improvement! But the text is still a long way off being readable. We must continue with this procedure until there are no two letters we can swap in the key that will result in an improvement of the fitness. After many iterations of this approach, the final key that was found was

XZTJWUMOBEPARIQKDLFSCHYGNV

resulting in the quite readable plaintext:
THESIMPLESUBSTITUTIONCIPHERISACIPHERTHATHASBEENINUSEFORMANYHUNDREDSOFYE
ALLYCONSISTSOFSUBSTITUTINGEVERYPLAINTEXTCHARACTERFORADIFFERENTCIPHERTEXTCHARACTER ITDIFFERSFROMCAESARCIPHERINTHATTHECIPHERALPHABETISNOTSIMPLYTHEALPHABETSHIFTEDITIS COMPLETELYJUMBLED

5 Input Text

aceah toz puvg vcdl omj puvg yudqecov, omj loj auum klu thmjuv hs klu zlcvu shv zcbkg guovz, upuv zcmdu lcz vuwovroaeu jczoyyuovomdu omj qmubyudkuj vukqvm. klu vcdluz lu loj avhqnlk aodr svhw lcz kvopuez loj mht audhwu o ehdoe eunumj, omj ck toz yhyqeoveg auecupuj, tlokupuv klu hej sher wcnlk zog, klok klu lcee ok aon umj toz sqee hs kqmmuez zkqssuj tckl kvuozqvu. omj cs klok toz mhk umhqnl shv sowu, kluvu toz oezh lcz yvhehmnuj pcnhqv kh wovpue ok. kcwu thvu hm, aqk ck zuuwuj kh lopu eckkeu ussudk hm wv. aonncmz. ok mcmukg lu toz wqdl klu zowu oz ok scskg. ok mcmukg-mcmu klug aunom kh doee lcw tuee-yvuzuvpuj; aqk qmdlomnuj thqej lopu auum muovuv klu wovr. kluvu tuvu zhwu klok zlhhr klucv luojz omj klhqnlk klcz toz khh wqdl hs o nhhj klcmn; ck zuuwuj qmsocv klok omghmu zlhqej yhzzuzz (oyyovumkeg) yuvyukqoe ghqkl oz tuee oz (vuyqkujeg) cmubloqzkcaeu tuoekl. ck tcee lopu kh au yocj shv, klug zocj. ck czm’k mokqvoe, omj kvhqaeu tcee dhwu hs ck! aqk zh sov kvhqaeu loj mhk dhwu; omj oz wv. aonncmz toz numuvhqz tckl lcz whmug, whzk yuhyeu tuvu tceecmn kh shvncpu lcw lcz hjjckcuz omj lcz nhhj shvkqmu. lu vuwocmuj hm pczckcmn kuvwz tckl lcz vueokcpuz (ubduyk, hs dhqvzu, klu zodrpceeuaonncmzuz), omj lu loj womg juphkuj ojwcvuvz owhmn klu lhaackz hs yhhv omj qmcwyhvkomk sowcecuz. aqk lu loj mh dehzu svcumjz, qmkce zhwu hs lcz ghqmnuv dhqzcmz aunom kh nvht qy. klu uejuzk hs kluzu, omj aceah’z sophqvcku, toz ghqmn svhjh aonncmz. tlum aceah toz mcmukg-mcmu lu ojhykuj svhjh oz lcz lucv, omj avhqnlk lcw kh ecpu ok aon umj; omj klu lhyuz hs klu zodrpceeu- aonncmzuz tuvu scmoeeg jozluj. aceah omj svhjh loyyumuj kh lopu klu zowu acvkljog, zuykuwauv 22mj. ghq loj aukkuv dhwu omj ecpu luvu, svhjh wg eoj, zocj aceah hmu jog; omj klum tu dom dueuavoku hqv acvkljog-yovkcuz dhwshvkoaeg khnukluv. ok klok kcwu svhjh toz zkcee cm lcz ktuumz, oz klu lhaackz doeeuj klu cvvuzyhmzcaeu ktumkcuz auktuum dlcejlhhj omj dhwcmn hs onu ok klcvkg-klvuu

6 Output Text

bilbo was very rich and very peculiar, and had been the wonder of the shire for sixty years, ever since his remarkable disappearance and unexpected return. the riches he had brought back from his travels had now become a local legend, and it was popularly believed, whatever the old folk might say, that the hill at bag end was full of tunnels stuffed with treasure. and if that was not enough for fame, there was also his prolonged vigour to marvel at. time wore on, but it seemed to have little effect on mr. baggins. at ninety he was much the same as at fifty. at ninety-nine they began to call him well-preserved; but unchanged would have been nearer the mark. there were some that shook their heads and thought this was too much of a good thing; it seemed unfair that anyone should possess (apparently) perpetual youth as well as (reputedly) inexhaustible wealth. it will have to be paid for, they said. it isn't natural, and trouble will come of it! but so far trouble had not come; and as mr. baggins was generous with his money, most people were willing to forgive him his oddities and his good fortune. he remained on visiting terms with his relatives (except, of course, the sackvillebagginses), and he had many devoted admirers among the hobbits of poor and unimportant families. but he had no close friends, until some of his younger cousins began to grow up. the eldest of these, and bilbo's favourite, was young frodo baggins. when bilbo was ninety-nine he adopted frodo as his heir, and brought him to live at bag end; and the hopes of the sackville- bagginses were finally dashed. bilbo and frodo happened to have the same birthday, september 22nd. you had better come and live here, frodo my lad, said bilbo one day; and then we can celebrate our birthday-parties comfortably together. at that time frodo was still in his tweens, as the hobbits called the irresponsible twenties between childhood and coming of age at thirty-three

7 Python Code

7.1 Ngram Score

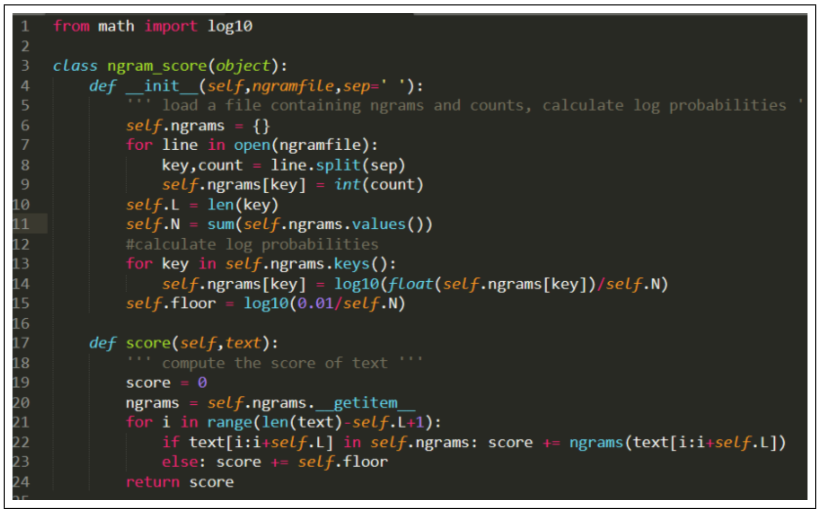
7.2 Decrypter

```
from pycipher import SimpleSubstitution as SimpleSub
import random
import re
from ngram_score import ngram_score
fitness = ngram_score('quadgrams.txt')  load our quadgram statistics
```

8 Reference

1. Quadgram Statistics as a Fitness Measure

2. Simple Substitution Cipher

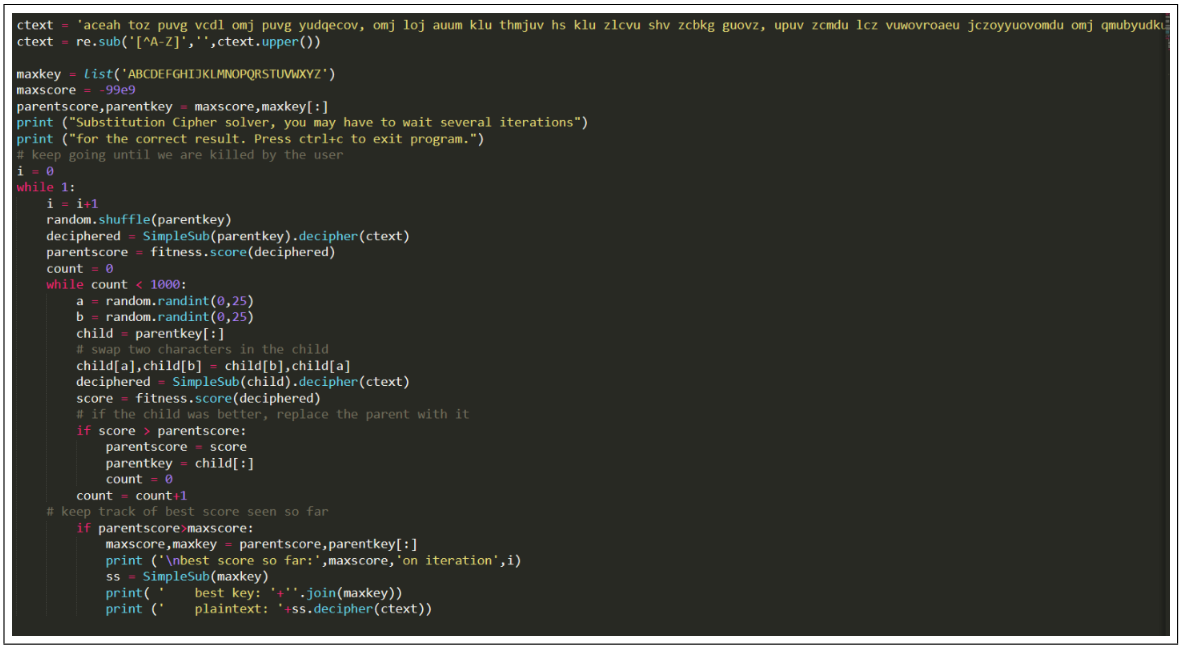


```python
from math import log10

class ngram_score(object):
    def __init__(self,ngramfile,sep=' '):
        ''' load a file containing ngrams and counts, calculate log probabilities '
        self.ngrams = {}
        for line in open(ngramfile):
            key,count = line.split(sep)
            self.ngrams[key] = int(count)
        self.L = len(key)
        self.N = sum(self.ngrams.values())
        #calculate log probabilities
        for key in self.ngrams.keys():
            self.ngrams[key] = log10(float(self.ngrams[key])/self.N)
        self.floor = log10(0.01/self.N)

    def score(self,text):
        ''' compute the score of text '''
        score = 0
        ngrams = self.ngrams.__getitem__
        for i in range(len(text)-self.L+1):
            if text[i:i+self.L] in self.ngrams: score += ngrams(text[i:i+self.L])
            else: score += self.floor
        return score
```

Figure 1: ngram-score.py



```python
ctext = 'aceah toz puvg vcdl omj puvg yudqecov, omj loj auum klu thmjuv hs klu zlcvu shv zcbkg guovz, upuv zcmdu lcz vuwovroaeu jczoyyuovomdu omj qmubyudku
ctext = re.sub('[^A-Z]','',ctext.upper())

maxkey = list('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
maxscore = -99e9
parentscore,parentkey = maxscore,maxkey[:]
print ("Substitution Cipher solver, you may have to wait several iterations")
print ("for the correct result. Press ctrl+c to exit program.")
# keep going until we are killed by the user
i = 0
while 1:
    i = i+1
    random.shuffle(parentkey)
    deciphered = SimpleSub(parentkey).decipher(ctext)
    parentscore = fitness.score(deciphered)
    count = 0
    while count < 1000:
        a = random.randint(0,25)
        b = random.randint(0,25)
        child = parentkey[:]
        # swap two characters in the child
        child[a],child[b] = child[b],child[a]
        deciphered = SimpleSub(child).decipher(ctext)
        score = fitness.score(deciphered)
        # if the child was better, replace the parent with it
        if score > parentscore:
            parentscore = score
            parentkey = child[:]
            count = 0
        count = count+1
    # keep track of best score seen so far
        if parentscore>maxscore:
            maxscore,maxkey = parentscore,parentkey[:]
            print ('\nbest score so far:',maxscore,'on iteration',i)
            ss = SimpleSub(maxkey)
            print( '    best key: '+''.join(maxkey))
            print ('    plaintext: '+ss.decipher(ctext))
```

Figure 2: decrypter.py