CS641 - Modern CryptoGraphy
Assignment 1

Shashvat Singham
(200922)

On entering the sequence of commands: Climb, enter and read, we reached the cipher-text for the level.

The cipher-text consisted of alphabets of the English language. We did a frequency analysis and found that the frequency of the letters in the cipher-text is quite similar to the frequency of letters in the English language. This fact along with the classic cipher types discussed in the class led us to conclude that it must be a Caesar cipher, substitution cipher or something similar to these. We first tried the Caesar cipher and then the substitution cipher. While Caesar failed, the given cipher was found to be substitution. We wrote a python script to decode the cipher, the details of which we describe below.

1 Caesar Cipher

Our code initially checks if the cipher is Caesar cipher or not. It does so by trying all the possible values of shift and generating possible plain-texts for each shift value. While shifting, the non-alphabetic characters are not changed. The generated plain-text is checked if it is a legible English paragraph. This is done via a dictionary matching. We use the dictionary already available in Linux. For each word, we check if it is present in the dictionary or not. Then we consider the fraction of words found in dictionary. If most of the words (in our code more than 90%) are present in the dictionary, we output the plain-text and the code terminates. We tried all 26 shifts and none of them gave legible English paragraph.

2 Substitution Cipher

On finding that the cipher was not produced using Caesar cipher, our code moves on to try substitution cipher. We start with an initial mapping of characters. The initial mapping is generated using frequency analysis of the code. First we find the most frequent letter in the cipher-text and map it to 'e'. The we find the most frequent 3-letter word and map it to 'the'. For example, if 'xyz' is the most frequent word 'x', 'y' are mapped to 't', 'h' respectively. We don't map 'z' to 'e' as it was determined earlier using frequency analysis. The we find single-letter words. If there is only one single-letter word, it is mapped to 'a', otherwise if there are two, the letter whose frequency is more is mapped to 'a' and the other to 'i'. The rest of the mapping is generated randomly.

We define quadgram score of a string as the sum of score of each quadgrams in the string. Example, "abcde" has two quadgrams "abcd" and "bcde". The score of a quadgram is proportional to its frequency in the language. We took the quadgram score file from here. The file was generated by considering large volume of English text replacing all non-alphabetic characters including whitespace and counting the number of times, each quadgram occur. For each quadgram we used to file from above link to calculate quadgram score as:

$$Score\ of\ a\ quadgram\ =\ log(\frac{Number\ of\ times\ this\ quadgram\ occurs}{Total\ Number\ of\ times\ that\ all\ quadgram\ occurs})$$

We took log to avoid numerical overflows as the numbers were quite large.

Using the quadgram score above, the initial mapping is improved. At each step, we took two elements from the mapping and swap them. If the decryption from the new mapping has better score we keep it, else we revert to the previous mapping. Our algorithm may be stuck and not loop indefinitely. To prevent this we introduce timeouts. Whenever the algorithm times out it aborts the current search, generates new initial mapping as described above and restarts. To determine if a correct decryption has been reached, we use dictionary matching same way as we used for Caesar cipher. For the given cipher-text, our code returned the plain-text in less than 2 minutes everytime we ran it.

The password we found was wuNy85Vodd. Also the message mentioned that the digits have been shifted by 4. Since, the digit within the message must also have been shifted, we concluded that the digits have actually been shifted by 2 and the correct password is: wuNy63Vodd.

Our algorithm is an instance of Hill-Climbing, a heuristic search algorithm for optimizations, where we used quadgram score as heuristic for improvement and dictionary matching as terminating condition.

3 Code

The submission package has the code we wrote.

decrypt.py : Substitution cipher and Caesar cipher decoder.

All the codes are well commented and implements the algorithms described above.