Module 7: Frequency Analysis

JHU EP 606.206 - Introduction to Programming Using Python

Introduction

Frequency Analysis can be used to break various cryptographic ciphers. Consider the Caesar Cipher below, used by Julius Ceasar, which would shift the letters in an alphabet and then substitute them to form an encrypted message.

a	b	c	d	e	f	g	h	…	z
g	h	i	j	k	l	m	n	…	f

Using the above encoding, the phrase “attack at dawn” would become “gzzgiq gz jgct”. The weakness here, though, is that the same letter always encrypts to the same ciphertext character. In our ciphertext example above you probably noticed multiple g’s and z’s. This allows you to begin to guess at which character is which based on how frequently each ciphertext character occurs (just like how R, S, T, L, N, and E are the 5 letters provided to you in the final round of Wheel of Fortune). For a breakdown of how frequently each English letter occurs on average, check out this site.

Midterm Assignment

If you know a piece of ciphertext has been encrypted using a shift cipher, there are multiple possible approaches to break the encryption including:

1. Guess randomly like a monkey. Fun, but not likely to yield the correct answer before the end of the Module.
2. Try to compute the plaintext message manually by trying all 26 shift possibilities. That is, decrypt the message with an alphabet shift of 1. If that does not yield an intelligible message, use a shift of 2, and so on until you crack the code.
3. **Count the number of occurrences of each character in the ciphertext and use the frequency data in the attached file to figure out the corresponding plaintext characters and, subsequently, the secret message.**

For the Midterm Assignment you'll take what you've learned in class so far to break a shift cipher-encrypted secret message read in from a file **using frequency analysis** (Method #3 above). Your program, **freq_analysis.py**, should read data from 2 files:

1. *ciphertext.txt*: contains the encrypted secret message
2. *freq.txt*: contains the frequency count for each character as it appears in the **decrypted** message

There are only two, small requirements for this assignment:

1. Your code should contain 1 function named **`get_freq_counts`** that accepts the encrypted message (string) and returns a dictionary that maps each character to the number of times it occurs.
2. Your solution should only contain code written by-hand and should not use any external libraries. All of the concepts and built-in libraries we've learned about are in play.

Other Notes

- The frequency analysis we have provided is **not** the same as the one listed above for overall English. We have given you one specific to this ciphertext, so use the attached file.
- You'll notice that a few characters occur the same number of times: 'g' and 'd', for example. This means your solution will likely have a few characters that don't quite decrypt correctly, but a correct solution will still decrypt the message with about 95% accuracy. The only letters that'll be off are the ones that share a frequency count with another letter as mentioned above with 'g' and 'd'.
- The first line of the frequency analysis file doesn't have a character associated with it. Or does it? The line is correct as written, we have given you the count for the spaces as well.
- For the final part of the assignment, you're being asked to take the (mostly) decrypted output from your code and decrypt the rest by hand. To help you confirm your fully decrypted message, here is the MD5 hash of the decrypted secret message: **c674b76e1ada393c4274ada8490a888d**. Recall from some of the lectures in class that hashing will check exact match only, so be meticulous when verifying your hashes. You can visit **https://www.md5hashgenerator.com**, paste your fully decrypted message into the textbox, then click 'Generate' to see if the hash of your solution matches ours. You will provide your fully decrypted secret message as one of your deliverables.

Deliverables

readme.txt

So-called "read me" files are a common way for developers to leave high-level notes about their applications. Here's an example of a README file for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name**: Your name and JHED ID
2. **Module Info**: The Module name/number along with the title of the assignment and its due date
3. **Approach**: a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn't fully work.
4. **Known Bugs**: describe the areas, if any, where your code has any known bugs. If you're asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.

Please submit your **freq_analysis.*py*** source code file, your README.txt, along with a PDF file containing screenshots of your output in a PDF called JHEDID_mod7.pdf (ex: jkovba1_mod7.pdf).

Please submit your readme.txt, **freq_analysis.py**, and a PDF with a screenshot of your program's output under the Module 7 Exam Assignment on Blackboard. Good luck, have fun, and feel free to let us know if you have any questions.

Recap:

1. readme.txt (please include your fully, hand-decrypted secret message here)
2. freq_analysis.py
3. A PDF containing a screenshot of your (mostly) decrypted output