CMPUT 331, Winter 2025, Assignment 5

Before beginning work on this assignment, carefully read the Assignment Submission Specifications posted on Canvas.

Introduction

Because mono-alphabetic substitution ciphers can be enciphered by more than 10^{20} possible keys, a brute-force approach will not work to decode a ciphertext. In this assignment, you will perform frequency analysis to decode mono-alphabetic substitution ciphers, and then evaluate the accuracy of a decoded plaintext.

All encipherment and decipherment in this assignment uses the simple substitution cipher. **Lowercase and uppercase letters are to be treated as the same. Punctuation, special characters, and spaces are not substituted by the mono-alphabetic substitution cipher.**

You will produce a total of four files for this assignment:

- **a5p1.py, a5p2.py, a5p3.py:** python solutions to problem 1 and 2 respectively
- a README file (**README.txt** or **README.pdf**)

Submit your files in a zip file named 331-as-5.zip. Lowercase and uppercase letters are to be treated as the same.

Problem 1

Modify the provided skeleton file named *a5p1.py*, which contains functions *freqDict(ciphertext)* and *freqDecrypt(mapping, ciphertext)*.

When invoked with a text enciphered using the substitution cipher, the *freqDict(text)* function should perform frequency analysis on the entire text, using the frequency statistics methods from the textbook, and return a dictionary whose keys are the cipher characters, with the value for each key being the plaintext character it is assigned to using frequency analysis.

For English texts that have been deciphered using this *frequency analysis* method, the value of the most frequent cipher character should be 'E', and the value of the least frequent cipher character should be 'Z'. **If two or more letters occur the same number of times in the ciphertext, the letters that occur earlier in the alphabet should be considered to have a higher frequency.**

The *freqDecrypt(mapping, ciphertext)* function simply accepts a dictionary mapping of characters (in the format returned by "freqDict") and returns the resulting text after applying the mapping to each character in the ciphertext.

Note: This method of cracking the substitution cipher is far from perfect and it is unlikely to return the correct key.

We provide the letters in the alphabet sorted by the frequency in English (from Wikipedia's article on Letter Frequency):

```
ETAOIN = "ETAOINSHRDLCUMWFGYPBVKJXQZ"
```

```
>> freqDict("AABBA")['B']
"T"

>> freqDict("good morning")['O']
"E"

>> freqDict("-: AB CD AH")['A']
"E"

>> freqDict("MKLAKAALK")
{'A': 'E', 'K': 'T', 'L': 'A', 'M': 'O'}

>> freqDecrypt({"A":"E","Z":"L","T":"H","F":"O","U":"W","I":"R","Q":"D"},
                "TAZZF UFIZQ!")
HELLO WORLD!
```

Problem 2

There are two primary ways of evaluating a solution to a simple substitution cipher: *key accuracy* and *decipherment accuracy.*

Key accuracy is the proportion of cipher character types in the alphabet which are mapped to their correct plaintext characters. Decipherment accuracy is the proportion of cipher character tokens in the ciphertext which are mapped to their correct plaintext characters.

Modify the provided skeleton file named *a5p2.py* that contains a function named *evalDecipherment(text1, text2)* where text1 is a plaintext and text2 is an attempted decipherment of a ciphertext created by enciphering text1. The evalDecipherment function should compare the two files, and return a list containing two fields that correspond to the key accuracy and decipherment accuracy of text2 w.r.t the plaintext, text1.

For example, if text1 contains the plaintext "this is an example", text2 might contain "tsih ih an ezample" – the text in text1 was enciphered, and text2 contains an imperfect attempt to decipher it. This decipherment has a key accuracy of 8/11, since there are 11 character types, and three of them, 'h', 's', and 'x', were deciphered incorrectly; it also has a decipherment accuracy of 11/15, since it is 15 characters long, but only 11 character tokens in the decipherment are correct. Thus, the function should return the list [0.7272727272727273, 0.7333333333333333]. **Your program should only count alphabetical characters in its decipherment evaluation and it should be case-insensitive.**

```
>> evalDecipherment("this is an example", "TSIH IH AN EZAMPLE")
[0.7272727272727273, 0.7333333333333333]

>> evalDecipherment("the most beautiful course is 331!",
        "tpq munt bqautiful cuurnq in 331!")
[0.7142857142857143, 0.625]
```

Problem 3

One problem with frequency analysis is that some cipher symbols will have the same frequency. In Problem 1, you used alphabetical order to break ties. This results in keys being inaccurate. While frequency analysis alone is not enough to fully decipher a text, permuting letters with the same frequency may increase the accuracy.

Modify the provided skeleton file named “a5p3.py”. You are tasked with adjusting your implementation from Problem 1 in the function “bestFreqDict(text1, text2)””, where text1 is a plaintext and text2 is the same text enchiphered with an unknown key. Rather than breaking ties by choosing the letters in the alphabetical order, try all different permutations of ties to find the best decipherment, evaluated using evalDecipherment from Problem 2.

You are provided with a plaintext and an encipherment using an unknown key. Your program should take these texts and find the mapping that provides the best **key accuracy** by permuting letters in the mapping with tied frequencies.

An example of an expected output, compared with the original accuracy. Your bestFreqDict should output the map providing the best score.

```
>> text1 = "If a man is offered a fact which goes against his instincts, he will
            scrutinize it closely, and unless the evidence is overwhelming, he
            will refuse to believe it. If, on the other hand, he is offered something
            which affordz a reason for acting in accordance to his instincts, he will
            accept it even on the slightest evidence. The origin of myths is explained
            in this way. -Bertrand Russell"
>> text2 = "RU Z NZM RH LUUVIVW Z UZXG DSRXS TLVH ZTZRMHG SRH RMHGRMXGH, SV DROO
            HXIFGRMRAV RG XOLHVOB, ZMW FMOVHH GSV VERWVMXV RH LEVIDSVONRMT, SV DROO
            IVUFHV GL YVORVEV RG. RU, LM GSV LGSVI SZMW, SV RH LUUVIVW HLNVGSRMT DSRXS
            ZUULIWA Z IVZHLM ULI ZXGRMT RM ZXXLIWZMXV GL SRH RMHGRMXGH, SV DROO ZXXVKG
            RG VEVM LM GSV HORTSGVHG VERWVMXV. GSV LIRTRM LU NBGSH RH VCKOZRMVW RM GSRH
            DZB. -YVIGIZMW IFHHVOO"
>> mapping = a5p1.freqDict(text2)
>> decrypted_text = a5p1.freqDecrypt(mapping, text2)

>> a5p2.evalDecipherment(text1, decrypted_text)
[0.043478260869565216, 0.13183279742765272]

>> mapping = bestFreqDict(text1, text2)
{’V’: ’E’, ’R’: ’T’, ’M’: ’A’, ’H’: ’O’, ’G’: ’I’, ’S’: ’N’, ’Z’: ’S’, ’L’: ’H’,
’X’: ’R’, ’O’: ’D’, ’I’: ’L’, ’U’: ’C’, ’W’: ’U’, ’D’: ’W’, ’T’: ’M’, ’E’: ’F’,
’N’: ’G’, ’F’: ’Y’, ’B’: ’P’, ’A’: ’V’, ’Y’: ’B’, ’K’: ’K’, ’C’: ’J’}
>> checkEvalDecipherment(mapping, text1, text2)
[0.13043478260869565, 0.1607717041800643]
```

Notice that the key accuracy and decipherment accuracy have both slightly improved by switching letters with tied frequencies.