CMPUT 331, Fall 2020, Assignment 10 (Version 1.0)

All assignment submissions must conform to the Assignment Submission Specifications *posted on eClass. Ensure that your submission follows these specifications before submitting your work.*

You will produce a total of three files for this assignment: "a10.py" for problems 1 and 2, "a10.pdf" or "a10.txt" for problems 3 and 4, and a README file (also in either PDF or plain text). **If your code requires any external modules or other files to run, include them in your submission as well. Your submission should be self-contained. Modules from the textbook must not be edited – if you wish to modify code from the textbook, put it in a module with a different name. Attribute any code you use, even if it is from the textbook.**

Introduction

Earlier in the course, you watched a pre-recorded lecture on the automated decipherment of monoalphabetic substitution ciphers. One of the problems considered in this lecture was that of *ciphertext language identification* (CLI), the task of determining the language of the plaintext. Three methods for solving this problem were presented:

1. Return the language with the closest sorted symbol distribution.
2. Return the language with the closest decomposition pattern distribution.
3. Return the language which gives the best trial decipherment.

In this assignment, you will implement simplified versions of *the first two* of these methods, and test them on a set of ciphers in ten languages: English, French, Italian, Spanish, German, Dutch, Polish, Russian, Bulgarian, and Greek. Since some of the files you use in this assignment will necessarily contain unicode symbols, remember to open the files using `encoding="utf8"`.

Problem 1

Given a text, its *sorted symbol distribution* (SSD) is a frequency distribution over the set of symbol frequency ranks. The most frequent symbol in the text has rank 1, and its relative frequency is the probability of rank 1 in the SSD. In general, $P_{SSD}(i)$ is the relative frequency of the i^{th} most frequent symbol in the text, where i ranges from 1 to the size of the alphabet (excluding whitespace and punctuation). For example, in a typical English text, $P_{SSD}(3) = 0.0817$ (using the textbook's relative letter frequencies), because 'A' is the third most frequent letter, and its relative frequency is 0.0817.

The key insight here is that SSD is invariant under monoalphabetic substitution. A monoalphabetic substitution may change what the i^{th} most frequent symbol is, but it will not change its relative frequency, and so $P_{SSD}(i)$ will not change. As different languages have different SSDs, we should be able to use this to identify the language of a ciphertext. First, compute the SSDs for various languages from sample texts. Second, compute the SSDs of the ciphertext(s). Third, determine which language has the closest SSD to the SSD of the ciphertext; return that language.

For the purposes of this assignment, you will compare frequency distributions using the sum-of-squared-differences method. So, given two SSDs P_1 and P_2, the distance between them

is:

$$\sum_{i=1}^{|alphabet|} (P_1(i) - P_2(i))^2$$

It may be that the two texts have a different number of symbol types. For example, a short English ciphertext may contain only twenty-four symbols, but a lonter English sample text may contain all twenty-six English letters. In such cases, you should take the larger of two alphabet sizes, and assign a probability of zero to "unseen" ranks. (In other words, undefined probabilities are zeros.)

Write a function *cliSSD* in your a10.py module. It should take two arguments, *ciphertext*, a string containing the ciphertext, and *files*, a list of strings, with each string being a path to a file containing a sample text in a single language. It should return dictionary, where each key is a string which is one of the elements of *files*. The value of each key should be the distance between the SSD of the text in the file, and the SSD of the ciphertext.

Problem 2

Given a word, its *decomposition pattern* is a k-ary tuple, where k is the number of distinct letter types in the word. Each element of the tuple is a positive integer which corresponds to one type of letter, and its value is the number of times that letter appears in the word. The elements of the tuple are in numerically sorted order, from highest to lowest. It is implicit in this specification that the sum of the elements of the decomposition patterns should be equal to the number of letters in the word. Here are some examples:

- "SEEMS" $\rightarrow (2, 2, 1)$.
- "BEAMS" $\rightarrow (1, 1, 1, 1, 1)$.
- "WERE" $\rightarrow (2, 1, 1)$.

Decomposition patterns are resistant to monoalphabetic encipherment: a word with five distinct symbols, such as "BEAMS" will always encipher to a word with five distinct symbols.

Given a text, its *decomposition pattern distribution* (DPD) is the probability distribution over the decomposition patterns of the words in the text. For example, if 0.19% of all word tokens in a text have pattern (2,2,1), then $P_{DPD}((2, 2, 1)) = 0.0019$. We can use DPDs to perform CLI just as we did with SSDs: compute the DPD of the ciphertext and some samples of various languages, and decide which of the samples has the closest DPD to that of the ciphertext. Again, use sum-of-squared-differences to compare DPDs.

Write a function *cliDPD* in your a10.py module. It should take two arguments, *ciphertext*, a string containing the ciphertext, and *files*, a list of strings, with each string being a path to a file containing a sample text in a single language. It should return dictionary, where each key is a string which is one of the elements of *files*. The value of each key should be the distance between the DPD of the text in the file, and the DPD of the ciphertext.

Problem 3

Included with this assignment are 100 ciphers, 10 each from 10 different languages. The file names indicate the correct ciphertext languages, e.g. *ctext_english_1.txt* is an English ciphertext. Apply each of the three methods you presented to these 100 ciphers, and compute the accuracy

of each: for how many ciphers, out of 100, did each method correctly identify the ciphertext language?

Report the accuracy of each of the two methods you implemented in percent. That is, for how many ciphers out of 100 did each method correctly identify the plaintext language? Briefly report your observations about the relative performance of the methods.

Problem 4

Create a *confusion matrix* for each of your methods. Each confusion matrix should be a a 10-by-10 array where each row and column corresponds to a language, with the languages in alphabetical order (e.g. the third row and the third column both correspond to English; the first and second are Bulgarian and Dutch). The value at row i, column j should be the number of ciphers which were actually of language i, but were identified as being of language j. So, if one method identified 7 English ciphers as English, 2 as German, and 1 as Russian, the row corresponding to English should have 7 in the English column, 2 in the German column, 1 in the Russian column, and zeros in all other columns.

Do you notice any patterns in your matrices? Do the methods tend to make similar errors? Are some languages more likely to be confused, or are the errors mostly evenly distributed? Speculate on why this might be.