CMPUT 331, Fall 2023, Assignment 10
Version 1, Nov 14, 2023

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

Introduction

You will produce three files for this assignment:

- python solutions to problems 1, 2, and 3 (**a10.py**)
- written responses to problem 4 (**a10.txt**)
- a README file (**README.txt** or **README.pdf**)

Submit your files in a zip file named **331-as-10.zip**.

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, please remember to open the files using `encoding="utf8"` and to convert all text into uppercase for consistency in processing. Please also refer to the *Assignment Submission Specifications* for the run-time limit.

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 can 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 longer one may contain all twenty-six 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(ciphertext, files)` in the a10.py module, taking 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 a dictionary, where each key is a string corresponding to an item in `files`. The value of each key is a float 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.

It is often the case that the decomposition pattern in the ciphertext does not match exactly with that of a sample text. In such scenarios, it is important to include all patterns from both the ciphertext and the sample in the comparison and assign a probability of zero to the patterns not observed for each side. (In other words, undefined probabilities are zeros.)

Write a function `cliDPD(ciphertext, files)` in the a10.py module, taking 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 a dictionary, where each key is a string corresponding to an item in `files`. The value of each key is a float distance between the DPD of the text in the file, and the DPD of the ciphertext.

Problem 3

Included with this assignment are 90 ciphers, 9 each from 10 different languages. The filenames reveal the correct ciphertext languages. For example, the file "ciphertext_en_1.txt" contains an English ciphertext. Apply the two methods presented to these 90 ciphers. You can evaluate them based on the accuracy: determine how many ciphers, out of 90, the method can correctly identify the ciphertext language.

First, write a function `cliSSDTest(ciphertext_files, sampletext_files)` in the a10.py module to implement the SSD method.

Second, write a function `cliDPDTest(ciphertext_files, sampletext_files)` in the a10.py module to implement the DPD method.

Both functions should take two arguments: `ciphertext_files`, a list of strings, with each string being a path to a file containing a ciphertext in a single language (e.g., "ciphertext_en_1.txt"), and `sampletext_files`, a list of strings, with each string being a path to a file containing a sample text in a single language (e.g., "sample_en.txt"). It should return a dictionary, where each key is a string corresponding to an item in `ciphertext_files`, and the value of each key is a string corresponding to an item in `sampletext_files`. The value should represent the sample file that has the closest distance to the key which is the ciphertext file, indicating the identified ciphertext language.

An example output is as follows.

```
{"ciphertext_el_1.txt": "sample_el.txt",
 "ciphertext_el_2.txt": "sample_el.txt",
 ...
 "ciphertext_nl_8.txt": "sample_nl.txt",
 "ciphertext_nl_9.txt": "sample_nl.txt"}
```

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 6 English ciphers as English, 2 as German, and 1 as Russian, the row corresponding to English should have 6 in the English column, 2 in the German column, 1 in the Russian column, and zeros in all other columns.

Examine the two matrices for any noticeable patterns. For example, are some languages more likely to be confused, or are the errors mostly evenly distributed? If you find any, please provide a brief description of these patterns in a few sentences limited to at most 50 words.

Your two confusion matrices and the description should be saved as a10.txt for submission.