Homework 1 - Substitution Cipher
CIS 573 Software Engineering

Siddharth Srivatsa

September 18, 2015

Introduction

A Substitution Cipher is a method of encrypting documents where simple plain text is replaced with some code or cipher text. These codes can be shifts or substitutions in letters, pairs of letters, triplets or mixtures of the above. The receiver is aware of the substitution and performs the inverse operation for decryption and obtains the original text.

One of the ways of deciphering an encoded cipher, if the substitution is not known, is to use frequency analysis. Frequency analysis is based on the fact that in any given language, the occurrence of certain letters and certain combinations of letters is significantly more and the occurrence of all letter is quite varied over a large amount of data. This fact can be leveraged to build a model to decrypt an encoded message (encoded using a simple substitution cipher).

In this assignment, frequency analysis was used to decrypt an encrypted document and cross validation was used to estimate the accuracy of the model that was built. A corpus of documents was provided, from which each documented was extracted and encrypted using a simple substitution cipher. The rest of the documents in the corpus were analyzed and a model was built based on the letter count using frequency analysis. The encoded document was deciphered based on the model and the number correct and incorrect decryptions were noted. This was repeated for each file in the corpus and an overall accuracy score was obtained. The whole project was implemented on Java.

Analysis

1. Being new to Java, I am still trying to wrap my head around classes and objects and am not sure how to use them effectively. I used a single class with different methods in it. My opinion was that classes were to be used for representing different groups of items or things etc and didn't quite seem to fit here.
   On a high level, I had four main methods in my class - the main, encrypt, formModel, and decrypt. Apart from these I had a few other small 'helper' methods to make the code easier to read and understand. Each method was named based on the function it performed.
   I limited the number of fields (to just two - the correct and incorrect decryptions)and mostly used local variables. This was done to prevent mixing of data, unnecessary change in variables, etc.

2. A few reasons why I would consider my code to be designed well are -

   - Components only from Java.IO and Java.util used. No external libraries used. Hence the code can be easily run by anyone with additional effort.
   - Since the program checks if the file is valid, and omits all files in the corpus that are not valid txt files, changing the file format does not affect the code.
   - If the content (Language, etc) of the file changes, the program will be able to handle to the files and read them fine. However since I check for English alphabets in my document, using a different language will not result in a successful model though the code will work and run.
   - Handles most exceptions.
   - Code is not very long (about 280 lines).

3. A few reasons why I would consider my code to be easily read and understood well are -

   - Well commented code.
   - Practically named methods and variables which make sense and help understand what they are and what they do.

- Program follows Java conventions and ensures familiarity.
- Code is not very long (about 280 lines) which makes it easier to read.
- Code is formatted and indented well with appropriate punctuation marks.

4. A few ways in which your code is efficient in terms of execution time and/or memory usage are -
   - The program takes approximately 4 seconds (measured using System.nanoTime()) to run each iteration of encryption, model forming and decryption for a which is around 140 kB. The number of characters that were encoded were around 80,000.
   - The size of the program itself is just 7kB which makes it very small.
   - Usage of *treemaps*, to store the frequency counts of letters is optimized. Runtime is log(n).
   - Usage of *Character* instead of *String*, for the model to store letters.
   - Usage of BufferReader instead of Scanner to read in the documents. (Buffer reader has more memory and can handle larger documents as Strings).
5. The way I ensured my code was implemented correctly was through testing. I was not very confident in using the Java test bench and was unsure about it's usage, so I wrote a sample file (With just a few words) and tested my implementation with this file. Since each method in the program is independent of the other, I was able to test each method on the file and ensure that my implementation was right. A better solution though would have been to write unit tests for the methods implemented.