CIS 573 Software Engineering
Fall 2015
Homework #1

Introduction

In this assignment, you are asked to implement a Java application that attempts to decrypt a document that has been encrypted using a substitution cipher. The algorithm that you will use is described below, and then your program will estimate the accuracy of this approach using a technique called cross-validation.

This should not be particularly challenging for you: by now you should feel comfortable writing reasonably large programs in Java, specifically designing classes, doing file I/O, using data structures, manipulating Strings, implementing basic algorithms, understanding whether your code works correctly, etc.

The purpose of this assignment is to get you thinking about some of the concepts we will cover in this course, particularly as they relate to your "personal software process" and "software quality." Anyone with even a little bit of Java experience should be able to write this program: you, however, are being asked to write a "good" program, and to justify why you think it's good.

Although you are free to discuss this assignment with your classmates, the work you submit must be your own: *you should not be sharing or discussing code with anyone else, nor should you be sharing answers to the writeup questions*. If you are uncertain of the policies, please ask a member of the instruction staff.

This assignment is due in Canvas on Friday, Sept 18, at 5:00pm; see submission instructions below. Late submissions will be subject to a penalty of 10% per day.

Background

As you probably already know, a document can be **encrypted** in order to hide its contents from unauthorized viewers.

A simple encryption algorithm is **substitution cipher**, in which each letter of the "plaintext" document is replaced by another letter to produce the "ciphertext," and all instances of that (plaintext) letter are always transformed to the same (ciphertext) letter. See http://en.wikipedia.org/wiki/Substitution_cipher

The ciphertext can then be **decrypted** by doing the reverse transformation, thus resulting in the original plaintext.

If the mapping/transformation of plaintext characters to ciphertext characters is not known, **frequency analysis** can be used to attempt to guess the mapping and decrypt the document. The intuition is that letters that appear frequently in the ciphertext are most likely mapped to letters that appear frequently in the language, whereas letters that appear frequently in the ciphertext are unlikely to be mapped to letters that *don't* appear frequently in the language. See: http://en.wikipedia.org/wiki/Frequency_analysis

To determine the frequency with which letters appear in the (plaintext) language, a corpus of documents can be analyzed and a model can be created, which ranks the letters according to their frequency. The model can then be applied during decryption in order to guess the mapping.

For instance, in general English the most frequently appearing letters are *E*, *T*, *A*, *O*, *I*, and *N*, in that order. Thus, using this approach, whichever letter appears most frequently in the ciphertext can be mapped to *E*, the next most frequently appearing letter can be mapped to *T*, and so on.

There are, of course, more complex approaches – for instance using bigram and trigram analysis, looking for most common words, etc. – but this is a simple approach that works reasonably well.

How well does it really work, though? A simple way of validating this approach would be to encrypt a document, use a corpus of other documents to create a model, apply frequency analysis to decrypt the ciphertext, and compare the decrypted ciphertext to the original plaintext. Then we could use some metric (such as the percent of letters correctly decrypted) to measure success.

To prevent bias from measuring the correctness for just one document, we could use **cross-validation** to measure each document in the corpus. However, to prevent overfitting, the document that we're measuring should be removed from the corpus when building the model. See: http://en.wikipedia.org/wiki/Cross-validation_(statistics)

That is, for each document T in the corpus:

1. Encrypt T to create ciphertext C
2. Remove T from the corpus and use the remaining documents to build a model
3. Apply the model to C to get the decrypted document D
4. Compare the original plaintext document T to the decrypted document D

The overall metric is then the percent of letters correctly decrypted across all documents in the corpus.

Specification

This section describes the specification that your program must follow. Some parts are intentionally under-specified; you are free to interpret those parts any way you like, within reason, but you should ask a member of the instruction staff if you have any questions.

Input File Specification
The documents in the corpus can be assumed to be plain-text files consisting of lowercase letters, uppercase letters, digits, punctuation, whitespace, etc. You cannot, however, assume that the documents in the corpus are always in English.

You can, of course, create your own corpus as you develop your program, but the correctness of your program will be evaluated using the documents in the corpus we provide you. This corpus is available in the Files section of Canvas under "Homework Files" and then "Homework #1." The corpus consists of Shakespeare's comedies.

Functional Specification
The program must be implemented in Java and must adhere to the following specification:

1. The class containing your program's "main" method must be called "Main" and must be in the "edu.upenn.cis573.hwk1" package. This convention is used to make the TAs' lives easier!

2. The runtime argument to the program should be the name of the directory containing the documents; the user should be able to specify either an absolute path or a relative path, and you can assume that you do not need to traverse into subdirectories to find files.

Do not prompt the user for this information! This should be specified when the program is run, e.g. using a Run Configuration in your Eclipse project.

The program should display an appropriate error message and immediately terminate upon any of the following conditions:

- the number of runtime arguments is not correct
- the specified directory does not exist
- the specified directory cannot be opened be opened for reading (e.g. because of file permissions)
- the specified directory is empty; *hint:* take a look at the java.io.File class if you don't know how to determine this

3. Once the program starts, it should use cross-validation on the files in the specified directory (as described above), to estimate the accuracy of the letter frequency analysis approach to decrypting text files.

Although you are free to organize your code however you see fit (though you will need to justify your design after implementing the program), keep in mind that your program needs to be able to:

- encrypt a plaintext document using a substitution cipher
- build a frequency model from a corpus of documents (minus the document you've encrypted) – note that you may *not* use the published distribution of letters in English, you *must* build the model from the corpus of text

- apply the frequency model to the ciphertext to attempt to decrypt it
- compare the decrypted ciphertext to the original plaintext
- repeat these steps for all documents in the corpus

For the encryption, you may not use an "identity cipher" that simply transforms a letter to itself.

Also, although the documents in the corpus may consist of uppercase and lowercase letters, encryption should be case-insensitive, so that 'A' and 'a' both get transformed to the same letter, e.g. 'k'. This means that your ciphertext should not have any uppercase letters.

Last, you do not need to encrypt punctuation, whitespace, digits, etc. and you should leave those "as-is" in the ciphertext. However, you should **not** include those characters when measuring the effectiveness of the approach, since they were never encrypted in the first place!

4. As your program performs cross-validation, it should print to the console (using System.out.println) the number of correctly and incorrectly decrypted letters for each document, followed by the total number and then finally the overall accuracy.

The output should look something like this:

```
apple.txt: 5372 correct, 2644 incorrect
banana.txt: 73021 correct, 44292 incorrect
carrot.txt: 8201 correct, 3322 incorrect

Total: 86594 correct, 50258 incorrect
Accuracy: 63.27%
```

Your program should **not** print out any debugging info, intermediate calculations, etc.

5. If any unexpected error occurs during the execution of the program, an appropriate error message should be shown and the program should terminate gracefully (no stack traces!).

But what about…?
You may have found a number of places in which this specification is ambiguous or under-specified. That is somewhat intentional as it simplifies things like error handling and corner cases (which we'll deal with later in the semester).

If there is something that appears to be unspecified or ambiguous, you may handle those situations any way you like, within reason. As long as your program adheres to what's written above, you should be fine. We're not trying to trick you, we promise! But if you're unsure, please ask a member of the instruction staff for help.

One last thought:
Please keep in mind that whether or not this is a "good" approach to solving the problem of decrypting text is not the point of the assignment. So although you may not agree with this approach or you may think of a better one, please adhere to this specification. Thank you. :-)

Before you begin

I know, I know, you want to start coding! But before you begin, think about the design of your program, as you will need to justify it after you're done implementing it.

In particular, think about all of the things that could potentially **change** in this program:

- the format of the input files (e.g. JSON instead of plain-text)
- the source of the input data (e.g. a database or web service instead of local text files)
- the appearance of the output (e.g. a Java Swing GUI instead of a command-line UI)
- the encryption algorithm that is used (e.g. something more complex than a substitution cipher)
- the decryption algorithm that is used (e.g. bigram frequency instead of letter frequency)

Then try to design your program in such a way that it would be easy for someone else to change the code if any of these things changed.

You may be thinking to yourself "yeah, but I *know* that no one else will ever read this code and that it will never need to change!" And, well, that's probably true.

But maybe it's not? Maybe you'll have to revise this code in a future assignment? Heh heh heh...

And even if you don't, in this class we're asking you to at least consider "what if?" you do need to reuse or change the code in the future (because this is certainly a concern in "the real world"), and to design/write it in such a way that makes that easier.

Analysis

Implementing the program is the easy part of this assignment. What we're more interested in is getting you to think about the way in which you develop software, and your own notions of what is "good" software.

In order for you to get full credit on this assignment, you must create a PDF document in which you answer all of the following questions. There are no "right" answers (yet), but you should answer these honestly and show that you have thought about what makes your code "good."

1. How did you design your application? That is, how did you choose what *classes* to create, what *fields* and *methods* each of them would have, and how they *relate* to each other?

2. Describe **four** specific and distinct ways in which the *design* of your classes is "good." In particular, why would it be easy to modify your code if any of the above things (format of data, format of input file, etc.) were to change?

3. Describe **four** specific and distinct ways (not related to design) in which the *style* of your code is "good." In particular, why is it easy to read or understand your code?

4. Describe **three** specific and distinct ways in which your code is *efficient* in terms of execution time and/or memory usage.

5. How do you know your code works correctly? Describe in detail the steps you took to ensure the correctness of your implementation.

Please be sure to include your name in the PDF as well!

Help! I'm stuck!

If you are having trouble with the *algorithm* that you're supposed to use (i.e., the steps that you need to go through in order to solve the problem), then we suggest doing the following (in this order) to try to figure it out:

1. Walk away for 30 minutes or so. Sometimes it's best to let your brain rest for a bit, and then come back to the problem somewhat refreshed.
2. If you're still stuck, then try to go see a member of the instruction staff during office hours, which are posted in Piazza.
3. If you can't make it to office hours, look on Piazza to see if anyone else has asked a similar question (and gotten a response). If not, post a question there. If you're afraid that asking your question would reveal the solution, then make it a "private question" that only the instruction staff can see.

Note that "look on Internet" is **not** on this list!!! And neither is "ask a friend"!!!

Although the purpose of this assignment is not primarily to measure your ability to think algorithmically (after all, we're giving you the algorithm!), you should be coming up with the solution on your own to whatever extent possible.

If you are having trouble with the *implementation* (e.g., a bug you can't fix, some code that won't compile, etc.), then we suggest doing the following (in this order) to try to figure it out:

1. Look in your notes from your previous Java courses. Pretty much all the Java you need to know should have been covered in a previous course.

2. If that's not helping, check out the online Java documentation at http://docs.oracle.com/javase/tutorial/. This is not the same as "go on stackoverflow"!! You should be striving for a mastery of Java, not just a solution to this one particular problem.
3. If you're still stuck, then try to go see a member of the instruction staff during office hours.
4. If you can't make it to office hours, look on Piazza and post your question if you don't see a similar one already there.

The moral of the story: try to figure it out by yourself first, then use reference material, then ask a member of the instruction staff or use Piazza. But do not go into "the wild" for help!

If you are really struggling with this assignment (e.g. it's taking more than 10-12 hours), particularly if you do not have much Java experience, please speak with the instructor.

One more thing about Piazza...

Although you are encouraged to post questions on Piazza if you need help or if you need clarification, please be careful about accidentally revealing solutions.

For instance, please do not post questions along the lines of "My program says that the accuracy is 51.4% for tamingoftheshrew.txt; is that right?" It's important that all students determine for themselves whether their program is working correctly.

If you think your question might accidentally reveal too much, please post it as a private question and we will redistribute it if appropriate to do so.

Academic Honesty and Collaboration

You must work on this assignment **alone**.

The point of the assignment is for you to think about your own "personal software process" and your own notions of "good" software, and you can't do that if someone else is writing the code for you!

You may discuss the assignment specification with other students, as well as your general implementation strategy and observations, but *you absolutely must **not** discuss or share code, or the overall results of the cross-validation analysis, or your answers to the writeup questions.* Please see the course policy on academic honesty (posted in Canvas on the "Syllabus" page) for more information.

Also, on this assignment you may **not** use third-party libraries other than the standard Java API. That is, although you are of course free to use classes in the java.lang, java.util, java.io, etc. packages, you may not use libraries from other packages, e.g. org.apache.commons, without permission from the instructor.

Although you can, of course, look online for help with the Java API and things like that, you should not be receiving any help from outside sources, including students not taking this course. Yes, there are implementations of this program already out there, but you are expected to implement this on your own and then contemplate your own notions of "good" software.

Failure to follow these policies will be considered academic dishonesty, and will be dealt with according to the course guidelines and School and University policies.

If you run into problems, please ask a member of the teaching staff for help before trying to find help online or asking your friends!

Submission

All deliverables are due by Friday, September 18, at 5:00pm. Late submissions will be penalized by 10% if 0-24 hours late, 20% for 24-48 hours late, and so on, up to one week, after which they will no longer be accepted.

To submit your assignment, put all of your code and your writeup into a *single* zip file. Please be sure that your source code is in a folder called "src", and that you follow the instructions regarding the package and class names as described above.

Aside from submitting the source code of your implementation of the program and the PDF that answers the questions posed above, please be sure to submit any other supporting code, scripts, etc. you may have written for this assignment.

Also, in your PDF, please include a section describing any known bugs, e.g. features that are not completely implemented, potentially incorrect outputs, or corner cases that were not addressed. The penalty for these bugs will be slightly reduced if you admit them in advance.

Submit the zip file into the "Homework #1" assignment in Canvas. You may submit as many times as you'd like; only the last version will be graded.

Grading

This assignment is worth a total of 100 points (which is the standard point total for most assignments in this course).

60% of the score is related to adhering to the specifications listed above. As mentioned, although the assignment is somewhat under-specified, we are not trying to trick you with weird corner cases or error handling. If your program matches what's written above, you should be fine, regardless of how you handle the unspecified parts.

The remaining 40% of the score is related to your answers to the analysis questions. Although there are no “right” answers at this point (after all, we haven’t taught you any of this yet!), the grade will mostly be determined based on the amount of insight and level of detail and specificity that you provide. The more you write, the better!

Updated: 25 Aug 2015, 12:24pm