DIPF application process

Language recognition software

User guide and tests

Anil Narassiguin

Monday, December 9, 2013

Table of contents

The goal of this programming exercise is to recognize the language in which a document is written. First I'll fix the hypothesis: What kind of document? Which languages? Then I'll describe the method I used to implement my solution. The third part is a short user guide to understand how the final application works. Finally I'll test my program on different set of texts and I'll try to show the limit of this method.

I - Hypothesis

The first formulation of the problem was **"...implement a method to identify the language a document is written in"**. This problem is vague and hypothesis must be fixed. Regarding the document type, should I make a program that reads text files, PDF files, HTML files, ect… ? This means that I would have to write or call many functions to remove useless content. So in a first approach, we'll consider that the program takes as input only **plain text** (.txt files).
Considering the language, I first restricted the hypothesis to languages that use the Latin alphabet. A character recognizing system would have recognized other alphabets (Cyrillic, Arabic, Devanagari…) and I assume that the method I use, based on letters frequencies, could work to distinguish the languages written in one of those alphabets (Arabic and Persian for the Arabic alphabet, Russian and Ukrainian for the Cyrillic alphabet, Hindi and Marathi for Devanagari, for example). To make my work faster and because the method is still not very efficient for very related languages (Catalan and Spanish for example), I decided to make a solution that recognizes **7 European languages**: English, French, German, Italian, Spanish, Dutch and Polish.

II – Method used

The two main ideas that came to my mind were to use either a method based on the articles that are very different from languages to languages ("the" in English, "le" and "la" in French for example), either a method based on letters frequencies in texts. I decided to use the second method because I found it more general. Then I thought that I should use machine learning algorithms to find patterns in letter distributions but it would have taken a lot of time.
Statistics made on thousands of dictionaries definitions and encyclopedias articles allow us to know very accurate letter frequencies for different languages. The Wikipedia article "Letter frequency" gives us these statistics for 14 languages [1].
My solution is very simple and explicit: given those statistics and a text file, the program can compute the letters distribution for the given text and then compare their relatedness by computing the square error of the two distributions:

$$Error = \sum_{\alpha \in Latin\ alphabet} \frac{|English(\alpha) - Text(\alpha)|^2}{||English||_2}$$

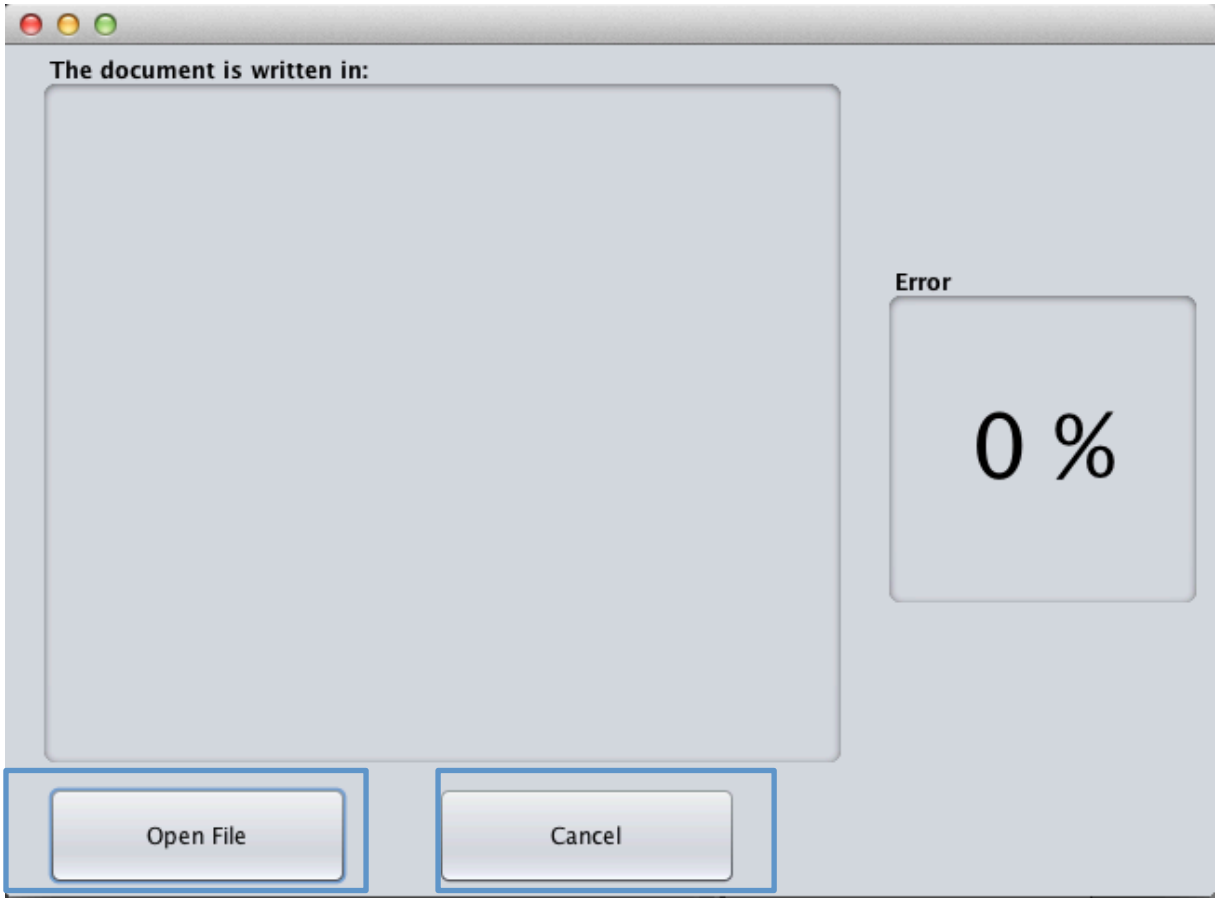
English(α) and Text(α) are respectively α frequency in the designated language (here English) and α in the text, knowing that α is a letter from "a" to "z". The diacritical marks and some specific letters such as the German β are voluntary ignored. Even if we lose information by doing this, the results that we obtain afterwards are pretty good (It's maybe not a good method if we want to compare Slavic languages like Czech, Polish and Slovak because they use much more diacritical marks than other European languages).
$||English||_2$ corresponds to the distribution's norm 2 :

$$||English||_2 = \sum_{\alpha \in Latin\ alphabet} English(\alpha)^2$$

Finally by using a sorting algorithm, the program finds the smallest error.

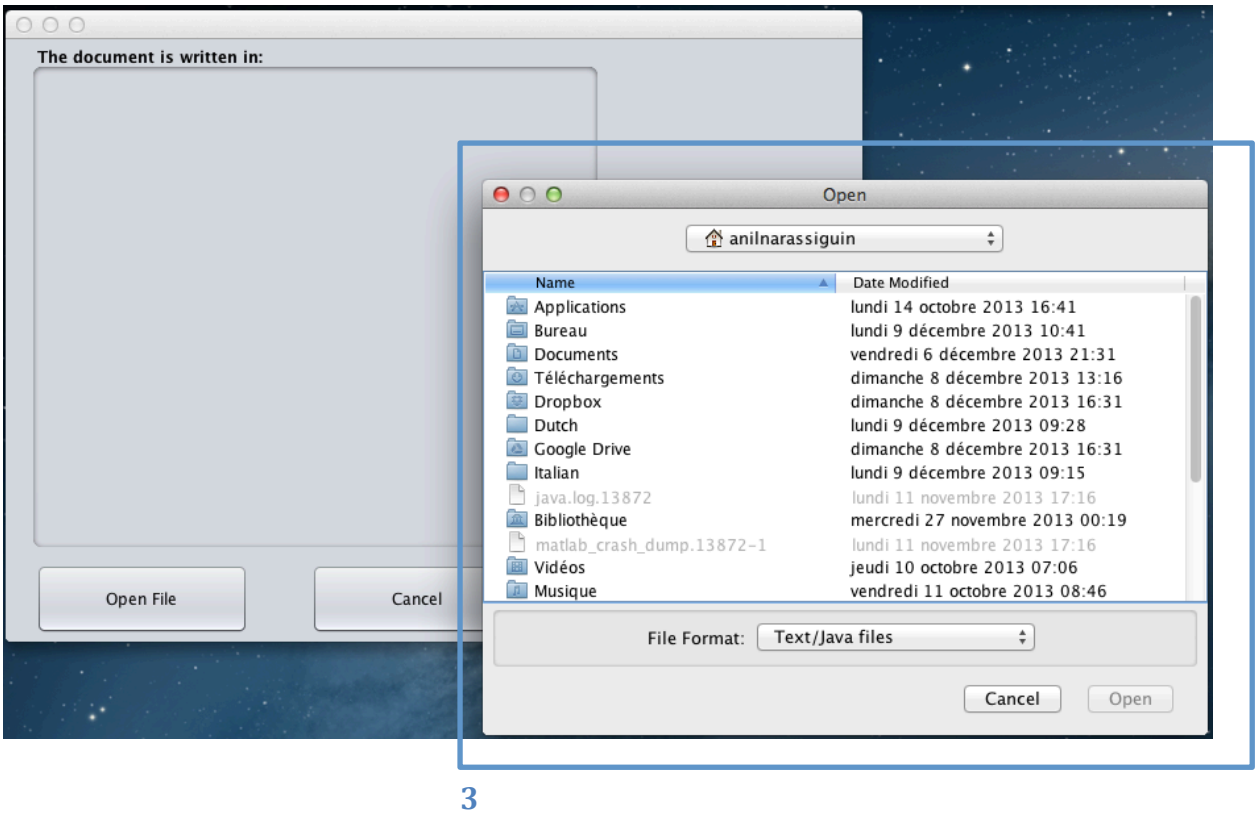
III – User guide

This Java application was developed on a Mac (OS X Mountain Lion) with NetBeans. The class Find_language makes all the computation and the class Display_result is charge to build the interface. They are given in a folder with flags images and other tests files.
Once you run the code on your IDE, a Frame will open:



1 2

If you click on Cancel (2), the Frame will close. Click on Open File (1) and a File Chooser will be opened (3).



If you click on cancel, it'll close the File Chooser. You can select online text files. Once you selected the text file you want to analyze, new information will be displayed on the main Frame.

The program guessed the language (4) ! To make the applet more attractive, the European flag corresponding to the language is also displayed (5). As an indication, you can also see the error defined above (6).

IV - Test

First to evaluate the effectiveness of the method, I test the program on different set of texts. Those texts are automatically generated thank to the website Random Text Generator [2]. This website generates random texts without any specific meaning, so some other tests should be done also on novels, articles, ect… The files generated by Random Text Generator are given in the attached files. Here are the results:

English:
On 10 generated texts, 10 are recognized as English (errors between 17% and 19%)

French:
On 10 generated texts, 10 are recognized as French (errors between 10% and 14%).

German:
On 10 generated texts, 10 are recognized as German (errors between 12% and 17%).

Italian:
On 10 generated texts, 10 are recognized as Italian (errors between 11% and 15%).

Spanish:
On 10 generated texts, 10 are recognized as Spanish (errors between 14% and 19%).

Dutch:
On 10 generated texts, 10 are recognized as Dutch (errors between 9% and 13%).

Polish:
On 10 generated texts, 10 are recognized as Polish (errors between 18% and 23%).

Those results for large text sets are pretty good. It would be useless to test this program on short sentences, because the method is purely statistic and according to law of large numbers the experiment set must be big enough to converge towards theoretical data.
In practice, the program will work well for texts with more than 100 characters.
Even so, a pure statistics method has a lot of disadvantages, as we'll see afterwards.

<u>The limits of the program</u>:

Since the method is only based on letters frequency, this software has some issues to identify dialects and cousin languages.

French Creoles:
I selected 3 texts from Haitian, Mauritian and Louisiana Creole (spoken respectively in Haiti, Mauritius and the American State of Louisiana).
The Louisiana Creole is recognized as Italian with an error of 38% (63% for French).
The Mauritian Creole is recognized as Italian with an error of 39% (53% for French).

The Haitian Creole is recognized as Italian with an error of 42% (50% for French).

This result is due to the fact that those creoles use more “a” and “o” vowels (like in Italian language) and even if most of the words derive from French, the program algorithm won’t be able to recognize it. Maybe a method based on k-mer frequencies would fix this issue.

German dialect:
Alemannic German is a German dialect spoken in some part of Germany, Austria, Switzerland, France (Alsace) and Italy (Aosta Valley). Two texts I found on Wikipedia were recognized as German.

Other Slavic languages:
This program is very bad to link Czech and Slovak with Polish. A Czech text and a Slovak text were recognized as a Spanish text (respectively 42% and 44% of error). Included the diacritical marks in the analysis might be a good idea to distinguish those three languages.

References

[1] Wikipedia article about letter frequencies:
http://en.wikipedia.org/wiki/Letter_frequency

[2] Random Text Generator website: http://randomtextgenerator.com/