Language Identification System

CmpE493 Summer 2011

8/4/2011

Dilara Semerci & Erkan Solmaz
CONTENTS
1. INTRODUCTION .................................................................................................................. 2
2. PROGRAM INTERFACE and EXECUTION ........................................................................... 3
3. INPUT FORMAT .................................................................................................................... 9
4. EXPLANATIONS of THE FUNCTIONS .................................................................................. 10
   Button Functions: .................................................................................................................. 10
   - private void browse_Click(object sender, EventArgs e) .................................................. 10
   - private void show_Click(object sender, EventArgs e) ................................................... 10
   - private void fileSelection_Click(object sender, EventArgs e) ........................................ 10
   - private void buildDict_Click(object sender, EventArgs e) ............................................. 10
   Helper Functions .................................................................................................................. 11
   - public string[] RemoveDuplicates(string[] myList) ....................................................... 11
   - public string preprocess_input(string input) .................................................................. 11
   - public string[] chopped_words(string[] filenames) ..................................................... 11
   - public void write_to_dictionary(string[] sortedArr) .................................................... 11
   - public string read_File(string filename) ...................................................................... 11
   - private void inputText_TextChanged(object sender, EventArgs e) ............................... 11
   - public int fileContains(string filename, string cmpStr) ............................................... 12
5. IMPROVEMENTS and EXTENSIONS .................................................................................. 12
6. CONCLUSION ....................................................................................................................... 12
1. INTRODUCTION

In this project, we tried to implement a Language Identification System that helps the user to identify the language of the input text. There are some online systems where the user can find information about the language of their text or document directly; even Google Translator gives suggestions about the language of the text that we try to translate to another language.

We tried to implement an offline system where the user can learn the language of the text without any internet connection. We implemented our project in such a way that the user can add other languages that it does not include initially as there are so many languages with so many different alphabets on earth.

Most of the time, Latin-based languages such as French, Spanish, Italian can be mixed with each other unless the user is familiar with those languages. As they have many common words, sometimes with a small accent difference, it is hard to distinguish one from the other. In order not to be mistaken, we show the probabilities of the possible languages and suggest the most probable one which has the highest probability out of those.

In the project, a user-friendly interface which consists of buttons and textboxes are available to use it easily. The interface may warn the user in order to prevent him to enter invalid inputs.

The available built-in languages in the project are English, French, German and Spanish. We tried to add as many words as possible however if the user wants to contribute to the project, he/she can add new words to these languages by simply giving text files to the program and specifying the language name as the language which he/she wants to contribute. The program will automatically expand the dictionary of the language with the new data coming from the user.

After this brief introduction, let us explain our system more deeply by giving details on which methods run on behind and how the interface is used and executed. We also explained and gave details about methods we used in this project.
2. PROGRAM INTERFACE and EXECUTION

We have implemented our project as a Windows Application project in C#.NET in Visual Studio 2010. Here is the opening window of our project:

Let us explain each button and text box separately.

- Browse Button: This button basically opens a new dialog window where the user can select his/her document with an extension of “.txt” that he wants to learn the language of. After the document is selected, the contents of that document will be shown in the text box on the bottom-left. Here are the screen shots after clicking the Browse button (Figure 2) and after selecting the file (Figure 3).
Figure 2: After Browse Button

Figure 3: After the File is browsed
Show the Language Button: After entering the text directly to the text box or just browsing a text file from the computer, “Show the Language” button is enabled. When the user clicks on this button, the probable languages are shown on the middle-bottom text box with their probabilities and the most probable one will be shown on the upper-middle text box as the answer:

As it is shown above, the languages are given with the probabilities. These probabilities are rather hits/total words in the document, meaning that they do not sum up to 1 but they are interpreted independently. There might be ambiguities if the user enters a few common words but then our system will show 0.5-0.5 for the probabilities and show the last language as the result.
The current dictionaries included in the system are shown in the ComboBox (Figure 5).

![Figure 5: Current Dictionaries](image)

Sometimes, the user can enter a text in an unknown language. In this case, our program gives out a message box saying that the input is probably in other language (Figure 6). It suggests user to upload documents in that language and build his own dictionary out of the words in those documents so that next time the system will recognize that language.

In order to upload files to build a dictionary, the user just has to click on the “Select the Files to Build the Dictionary” button.

- **Select The Files to Build the Dictionary Button:** When the user clicks on this button, a browse dialog window will be opened and the user can select multiple files out of those documents. But the user must chose only files with .txt extension as C# only lets us to open and read text files. After selecting the files, the user must enter a name for that language. (Figure 7) It should be followed by clicking the “Build Your Dictionary” button. When the dictionary is created, the user is warned (Figure 8) and that language is automatically added to the language list in the combobox (Figure 9). In fact we hold a text file which can be seen as a .ini file that initializes the languages that are included so far.
Language Identification System

Figure 6: For the non-recognized Languages

![Image of software interface showing text and language selection]

Figure 7: After Selecting the Dictionary Files and selecting the name of the Language

![Image of software interface with selected language and list of files]

This is our best result but most probably this text is in another language, we suggest you to upload files in this language to build your dictionary for another use!
Language Identification System

Figure 8: The Dictionary is built

Figure 9: The Language is added to the Dictionary List
3. INPUT FORMAT

As we have shown above, the user can either enter his text directly to the textbox or he can upload a document from his personal computer. Either way, the input will be displayed in the textbox and will be available to be changed if the user wants. The program simply takes its input from the textbox.

The input is stored as type string in the program and after taking the input from the user via textbox, it’s preprocessed with the following methods:

- **Case-Folding:** All capital letters in the input string are converted into equivalent lowercase letters.
- **Normalization:** All letters with accents in languages such as Spanish, French etc. are converted into their equivalent letters. (For German, "ß" character is replaced by "ss" characters.)

After preprocess phase, the string is split by almost all punctuations and sent to a function called `RemoveDuplicates(string[] s)` to simply get rid of duplications in the string. Finally, the list of words is sorted by using Sort method of Array class.

The program gets an array of strings which consists of unique words with lowercase letters and without accents.

This process is applied similarly to the dictionary expansion and addition phase. All files are read to a string as input and the procedures above are applied to this string as well. In this phase, after getting an array of strings, each string (i.e. word) is compared with the words in the file in which the string will be placed to avoid duplicate entries in the file which will not affect our program but its performance badly.

Finally, when inserting a word to a file in the dictionary creation or expansion phase, the program simply checks the first letter of the word to create or open the file in this format:

**Ex:** for English: `english.<first_letter>.txt`

File operations are done by using StreamWriter and StreamReader classes of the C# language.
4. EXPLANATIONS of THE FUNCTIONS

**Button Functions:**

- `private void browse_Click(object sender, EventArgs e)`

  This function is called when the “Browse” button is clicked. Within this function, a dialog is opened where the user can choose the text file to be browsed. After selecting a text file, its contents will be shown in the textbox whose name is `inputText`. This function also has a warning if the user selects any document whichever has other extensions than “txt”.

- `private void show_Click(object sender, EventArgs e)`

  This function is called when the “Show the Language” button is clicked. In this function, the input string is preprocessed by `preprocess_input` function which is going to be explained soon. After that the input string is split into words. In order to get rid of the duplicate words, `RemoveDuplicates` function is called. The distinct words array is then sorted so that the search will be easier.

  Then the list of the dictionary names is retrieved from “dictionaries.txt” file which is used as an initialization file. After that, for every language, the corresponding letter’s dictionary file is opened for every word in the input string. If the word is found in that language’s dictionary, we increment the counter for that language. In the end, we show the language that has the maximum number of hits. We also show the probabilities of each language for the input string.

  We show a message box to the user if there is no language with a probability more than 0.5 and warn him that the input is most probably in some other language.

- `private void fileSelection_Click(object sender, EventArgs e)`

  This function is called when the “Select The Files to Build the Dictionary” button is clicked. It opens a file dialog where multiple text file choices can be done. The chosen files are written to the textbox below the button.

- `private void buildDict_Click(object sender, EventArgs e)`

  This function is called when the “Build Your Dictionary” button is clicked. It takes the names of the files that are browsed and the filenames are sent to the `chopped_words` function which will be explained soon and the resulting array is sorted and sent to the `write_to_dictionary` function.
## Helper Functions

- **public string[] RemoveDuplicates(string[] myList)**

  This function helps to remove the duplicate entries in an array. It takes myList string array as input and copies each string to a new list without repetition and returns that list in string array format.

- **public string preprocess_input(string input)**

  This function helps to replace characters in the input string with different accents with regular characters so that the search will be easier. It just changes every character to lowercase and get rid of the accents of the letters.

- **public string[] chopped_words(string[] filenames)**

  This function takes a string array and for each of the elements it calls the read function with that element name which is basically the filename. After that it reads the contents of that file and removes the duplicates and puts the distinct words to a main array. All the files’ distinct contents are in that main array in the final. That main array is returned.

- **public void write_to_dictionary(string[] sortedArr)**

  This function takes an array of words. It creates a directory named the language that is specified by the user. Then for each word in that array, a text file is created with the first letter of the word (e.g.: english.a.txt) and the words are written to the corresponding dictionary files. The “dictionary.txt” file is also updated after the dictionary addition operation.

- **public string read_File(string filename)**

  This function opens a file named “filename” and it returns a string that contains the contents of that file.

- **private void inputText_TextChanged(object sender, EventArgs e)**

  This function just enables the “Show the Language” button when the inputText has something to be checked.
• public int fileContains(string filename, string cmpStr)

This function simply checks if a given string is contained in a file whose name is also given. It’s used to avoid duplicates in the dictionaries.txt and language files when contributing a new language or improving an existing one.

5. IMPROVEMENTS and EXTENSIONS

In our current system, both the browsed input text and the dictionary creation files should be in “txt” format. Due to the fact that file operations can be done with only text files in C# programming, we had to make a reduction in our project concept. For now, the user should browse only text files. For the future work, we can extend this feature by finding ways to deal with Word and Pdf documents.

6. CONCLUSION

In this project, we created a simple Language Identification System for the term project of CMPE493: Introduction to Information Retrieval Summer 2011. When implementing the project, we reviewed the concepts that we learned and discussed in our lectures and we tried to make use of them as much as possible. In input preprocessing and processing phases, we applied some of those concepts to our program.

Overall performance of the program may vary due to the factors such as the vocabulary size of a language dictionary; correct spelling of the words etc. But in our test cases, we observed that the program works fast and performs well enough. In this version of the program, some Asian languages such as Japanese, Korean, Chinese and Thai etc. are not supported due to character encoding. Therefore, it may not be reasonable to try to add these languages as new language to the program from the user perspective. This feature may be added in the future works.