language-detector

By Amit Kanfer and Punam Mahale

This is the documentation for the project.

An overview of the function of the code (i.e., what it does and what it can be used for)

The language-detector application detects the language of the text that is entered in the Input Text area. As you type the letters, the app in the background, runs an algorithm to detect the probabilities of
the string/text typed. Based on the best probability achieved, it concludes the language in the format "locale - language name", for e.g. "en - English". It also displays the percentage of probability for the detected text.

The idea behind this application was based on personal experience of one of our Author *Amit Kanfer*. Amit's native language being Hebrew, he mostly set his keyboard on Hebrew in order to write emails to friends/family. And many a times, when he is writing his office emails, halfway through he is on Hebrew and realizes that he should have being warned and changed his keyboard language. So, this is one of the use case for our application.

Along with the above, this application can be integrated with some of the below existing tools/application for language detection:

a. Text editor
b. Email composing editor
c. Chat applications

Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement

This application is build using Node Js and various NPM packages like n-gram, bluebird, etc.

Application is maintaining a resource for each of the supported language. Here it is supporting 71 languages. Refer to path *resources/languages*. For each language, this resource is basically the frequency of each letter in that language.

At run time, the application builds a profile for the languages based on the resource file available. To do this, it creates N-grams from the texts. This is stored in the profiles.

We are using NPM package `n-gram` to generate the N-grams. The support is for unigram, bigram and trigram.

When the user inputs the text, we go in exactly the same process. Based on the input text, create n-grams for it. And compare the relative frequency of them and find the language that matches the best frequency.

Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

Follow the below steps to run the application demo

1. clone the repo using https://github.com/amitkanfer/language-detector.git or by clicking on the Clone or download button and copying the GIT link.
2. run `npm install`
3. run `node main.js`
   Be sure to check if this throws error for PORT already in use.
   If you get an error for this, open `main.js` and look for line `const PORT = 80;`
   Change the port number to a desired one.
4. Change PORT number in PORT URL here and browse to `http://localhost:PORT/` using your favorite web browser.

Brief description of contribution of each team member in case of a multi-person team

Amit Kanfer worked on the algorithm in node and created the git repo to package this application
Punam Mahale worked on language name integration, UI styling and the documentation.

Challenges/Limitation

If the input text is short(less than 50 characters) or is unclean such as tweets, the application may give varying locale as the probability is calculated.

The input text if composed of various languages, the application detects the language that is the most dominant. The algorithm may detect wrong language. However, the suggestion is to split the text in paragrams or sentences and detect in parts.

Since, this application is supporting only 71 languages right now whose profiles gets build at runtime, if a language is not supported, the application may give unexpected results. One of the improvements to this application can be this scenario to warn that language is not supported.

Language Supported by our application

1. af Afrikaans
2. an Aragonese
3. ar Arabic
4. ast Asturian
5. be Belarusian
6. br Breton
7. ca Catalan
8. bg Bulgarian
9. bn Bengali
10. cs Czech
11. cy Welsh
12. da Danish
13. de German
14. el Greek
15. en English
16. es Spanish
17. et Estonian
18. eu Basque
19. fa Persian
20. fi Finnish
21. fr French
22. ga Irish
23. gl Galician
24. gu Gujarati
25. he Hebrew
26. hi Hindi
27. hr Croatian
28. ht Haitian
29. hu Hungarian
30. id Indonesian
31. is Icelandic
32. it Italian
33. ja Japanese
34. km Khmer
35. kn Kannada
36. ko Korean
37. lt Lithuanian
38. lv Latvian
39. mk Macedonian
40. ml Malayalam
41. mr Marathi
42. ms Malay

43. mt Maltese
44. ne Nepali
45. nl Dutch
46. no Norwegian
47. oc Occitan
48. pa Punjabi
49. pl Polish
50. pt Portuguese
51. ro Romanian
52. ru Russian
53. sk Slovak
54. sl Slovene
55. so Somali
56. sq Albanian
57. sr Serbian
58. sv Swedish
59. sw Swahili
60. ta Tamil
61. te Telugu
62. th Thai
63. tl Tagalog
64. tr Turkish
65. uk Ukrainian
66. ur Urdu
67. vi Vietnamese
68. wa Walloon
69. yi Yiddish
70. zh-cn Simplified Chinese
71. zh-tw Traditional Chinese

Authors

Amit Kanfer and Punam Mahale

References

https://www.npmjs.com/package/n-gram
https://www.npmjs.com/package/bluebird
http://bluebirdjs.com/docs/api-reference.html
https://www.npmjs.com/package/random-normal
https://www.npmjs.com/package/express
https://github.com/optimaize/language-detector
https://blog.xrds.acm.org/2017/10/introduction-n-grams-need/
https://en.wikipedia.org/wiki/Frequency_analysis