A Machine Learning Based Approach For Detecting Languages In Documents

Anindya Sundar Das

This article is the documentation of the method used to detect languages in text documents; It discusses scope of the method, datasets being used for training and evaluation and a thorough analysis of the results. This method can detect **54 natural languages** in text documents with an accuracy of **99.38%**.

1 Scope

This method can identify the following natural languages: **English, French, German, Spanish, Vietnamese, Catalan, Welsh, Czech, Danish, Estonian, Croatian, Finnish, Hungarian, Indonesian, Italian, Lithuanian, Latvian, Norweignian, Netherlandish, Portuguese, Polish, Somali, Slovenian, Slovak, Romanian, Albanian, Swedish, Swahili, Turkish, Tagalog, Afrikaans, Arabic, Persian, Urdu, Hindi, Marathi, Nepali, Ukranian, Russian, Bulgarian, Macedonian, Bengali, Greek, Gujarati, Punjabi, Hebrew, Kannada, Malayalam, Tamil, Telugu, Thai, Chinese, Japanese, Korean**. This approach assumes languages are written in their usual scripts with most of the words are spelled correctly i.e. a Hindi or Arabic paragraph written in Latin script, will yield wrong prediction.

Method

In this section, the method and algorithms for language classification is discussed in detail. This approach uses a two-step classification: First the languages in the text is classified into language group categories based on the usual scripts of the languages, in the next step the language groups are further classified into specific languages.

Language Group Classification

Based on the script of the written languages, the languages are classified into the following 17 groups (Refer Table 1). Among these language groups, four groups **(Latin, Arabic, Devanagari and Cyrillic)** need second stage classification, rest of the languages **(Bengali, Greek, Gujarati, Punjabi, Hebrew, Kannada, Malayalam, Tamil, Telugu, Thai, Chinese, Japanese, Korean)** can be detected in the first step of classification due to their unique written scripts. The posterior probability of a text belonging to language category is computed as follows:

$$P(C_i/t) = \frac{\sum_{j=0}^{N-1} W_j b_j}{nW_{max}} \tag{1}$$

Where N denotes total number of characters in text $t = (character_0 character_1 character_2 ... character_{N-1})$, $C_i \in \{C_1, C_2, C_3, ..., C_{17}\}$, is a language group, b_j is 1, if the character is found in script of language group C_i and is set to 0, if the character does not present in script for C_i. W_j is a scalar weight to be multiplied with b_j to emphasize the importance of certain characters in a text; example Chinese, Japanese characters represent words, unlike Latin, Devanagari or Arabic scripts; hence more weights are assigned to Chinese characters compared to Latin or Arabic characters in calculating the posterior probabilities for a text containing characters from multiple scripts. The denominator is multiplied with W_{max} i.e. the maximum possible character weight, to make sure individual probability scores are always less than 1. [1] Once the posterior probability of all language groups for a given text is computed, the language group for t is determined as $C = argmax_i P(C_i/t)$.

[1] Character based comparison and computations for UNICODEs are performed using python regex package.

Language Groups	Script	Languages
Latin	Latin	English, French, German, Spanish, Vietnamese, Catalan, Welsh, Czech, Danish, Estonian, Croatian, Finnish, Hungarian, Indonesian, Italian, Lithuanian, Latvian, Norweignian, Netherlandish, Portuguese, Polish, Somali, Slovenian, Slovak, Romanian, Albanian, Swedish, Swahili, Turkish, Tagalog, Afrikaans
Arabic	Arabic	Arabic, Persian, Urdu
Devanagari	Devanagari	Hindi, Marathi, Nepali
Cyrillic	Cyrillic	Ukranian, Russian, Bulgarian, Macedonian
Chinese	Han	Chinese
Japanese	Han, Hiragana, Katakana	Japanese
Korean	Hangul(mostly), Han	Korean
Bengali	Bengali	Bengali
Greek	Greek	Greek
Gujarati	Gujarati	Gujarati
Punjabi	Gurmukhi	Punjabi
Hebrew	Hebrew	Hebrew
Kannada	Kannada	Kannada
Malayalam	Malayalam	Malayalam
Tamil	Tamil	Tamil
Telugu	Telugu	Telugu
Thai	Thai	Thai

Table 1: Language Groups

Fine-grained Language Classification

Once the language groups are identified, the languages that uses same scripts is subjected to second stage classification to detect fine-grained language classes. These language groups are: **(Latin, Arabic, Devanagari and Cyrillic)**. In this stage a hash-map of language vocabularies are created for each language groups. The model is primarily hash-maps of vocabularies of language groups. Most Pre-trained word vector sets are sorted in descending order of word frequency. For this model, Fasttext [2] word vector sets which comprises of words and corresponding 300-dimensional word vectors are used for all the languages. The vocabulary map has word as key and a scoring function as value; for each languages the vocabulary hash-maps are built by extracting the most frequent 70,000 words from fasttext word vector sets. Hash map for each language group consists of vocabulary hash maps of all the member languages e.g. the hash-map for Devanagari group is composed of vocabulary hash-maps of Hindi, Nepali, Marathi. The scoring function for a word in a language (say c_l) vocabulary is calculated as follows:

$$Score^{l}_{word} = (1 + scalingfaction \times \frac{(Vocab\_size - word^{l}_{index})}{Vocab\_size}) \quad (2)$$

[2]https://fasttext.cc/docs/en/crawl-vectors.html

$Vocab\_size$ is vocabulary size of the language, as most frequent 70,000 words are chosen, so $Vocab\_size$ = 70,000. $word^{l}_{index}$ denotes the index of that particular word in Fasttext vector set of a language (say c_l). Scaling factor is set to 9 for this model. The formula ensures that words occurring more frequently in a language has more weights compared to less frequent words e.g. word with index 1 has a weight ≈ 10 whereas the word with index 70,000 has a weight of 1. The scoring function not only ensures simple vocabulary match but also it guarantees the frequency of occurrence of a particular word in a language is taken into consideration in language detection; such method is useful when same words are found in different languages. The hash-maps of scoring function are constructed for all four language groups and weights are stored as objects in pickle files.

For a given text $t = (word_0 word_1 word_2 ... word_{K-1})$ with K is number of tokens (words), respective class $\hat{c}$ is determined using equation 3.

$$\hat{c} = argmax_i P(c_i/t) = argmax_i \frac{\sum_{k=0}^{L-1} Score^{i}_{word_k} d_k}{(1 + scalingfactor) \times K} \tag{3}$$

Where d_k =0 if $word_k$ is out of vocabulary word in language c_i and d_k =1, if the word is present in respective vocabulary. The denominator has a factor $(1 + scalingfactor)$, so that the net scoring value is always less than 1 and $P(c_i/t)$ works as posterior probability.

Dataset

As discussed in section 1, for training/learning the parameters(hash-map scoring function) Fasttext word vector sets are used. The model is evaluated on WiLI-2018 dataset (Thoma, 2018). The original dataset contains 235,000 paragraphs of 235 different languages. From these 235,000 paragraphs 53,921 paragraphs of 54 languages (supported by the model under discussion) are extracted . As discussed in (Thoma, 2018), some of these paragraphs are wrongly annotated as some paragraphs are quotes in one language, in wikipedia of another language, e.g. **"He was a economics graduate from Elphinstone College, Mumbai. He was an industrialist in plastics business. He served as a president of Gujarat Chamber of Commerce and Industry in 1990s."** paragraph is marked as **Gujarati**, since it was extracted from wikipedia of Gujarati language, though the paragraph is written in **English**. So, the dataset is re-annotated. For **re-annotation** the python library **Langdetect** and python API for **google translator** are used. If the original annotation matches with Langdetect prediction, no changes are made, if there is contradiction between Langdetect prediction and actual annotation, then majority from all these three systems (Original Annotation, Langdetect, Google Translator detection) are used to re-annotate the paragraph, for a tie google translator detected language is selected for annotation. The final re-annotated dataset contains **53,921** paragraphs of **54** different languages.

2 Results and Analysis

The initial scoring function that has been used did not take account of the frequency of a word in a language, it was simple scoring function which assigns 1 for any word present in the vocabulary, ignoring their frequencies and position in Fasttext vector sets, i.e. $Scrore^{l}_{word}$ = 1 and the formula for language detection for this scoring function is obtained be setting $scalingfactor$=0 in Equation 3 . This leads to an accuracy of 99.2% on our dataset. After modifying the scoring function to incorporate frequency of the words in a language (Refer Eq. 2 and Eq. 3), the accuracy is improved to **99.38%**. The change of **0.18%** seems small but significant for difficult examples in which same words are present in different language vocabularies. The detailed report including language specific precision, recall and f-measure have been shown in Table 2. The model performs well in detecting most of the languages. Nevertheless, in case of English the precision and re-call both are lower (96% and 92% respectively). A thorough analysis depicted such wrongly classified examples are mostly collection of first names, places and mixture of foreign words (Table 3), which makes it difficult to categorize into a particular language.

Language	precision	recall	f1-score	support
Afrikaans	0.99	1	1	997
Albanian	1	1	1	976
Arabic	1	1	1	1003
Bengali	1	0.98	0.99	909
Bulgarian	1	1	1	965
Catalan	0.98	1	0.99	948
Chinese	1	0.98	0.99	997
Croatian	0.99	1	1	991
Czech	1	1	1	992
Danish	1	1	1	977
English	0.96	0.92	0.94	1690
Estonian	1	1	1	961
Finnish	1	1	1	996
French	1	0.99	0.99	1049
German	0.99	1	0.99	1098
Greek	1	0.99	1	991
Gujarati	1	0.99	1	951
Hebrew	1	1	1	998
Hindi	0.99	0.99	0.99	985
Hungarian	1	0.99	0.99	991
Indonesian	0.97	1	0.99	968
Italian	0.99	1	0.99	963
Japanese	0.99	1	0.99	993
Kannada	1	1	1	991
Korean	1	1	1	993
Latvian	1	1	1	987
Lithuanian	1	1	1	982
Macedonian	1	1	1	980
Malayalam	1	1	1	988
Marathi	1	1	1	966
Nepali	1	0.98	0.99	985
Netherlandish	0.99	1	0.99	979
Norweignian	0.99	1	1	990
Persian	1	1	1	999
Polish	1	1	1	1007
Portuguese	1	0.99	1	954
Punjabi	1	1	1	989
Romanian	1	1	1	988
Russian	1	0.99	1	1016
Slovak	1	1	1	992
Slovenian	0.99	1	1	976
Somali	0.99	1	1	954
Spanish	0.99	1	1	995
Swahili	0.93	1	0.96	952
Swedish	1	1	1	999
Tagalog	0.99	1	0.99	978
Tamil	1	0.99	1	994
Telugu	1	1	1	952
Thai	1	1	1	988

Language	precision	recall	f1-score	support
Turkish	1	0.99	1	1025
Ukranian	1	1	1	991
Urdu	1	0.99	0.99	963
Vietnamese	0.99	1	0.99	985
Welsh	0.99	1	0.99	994
micro avg	0.99	0.99	0.99	53921
macro avg	0.99	0.99	0.99	53921
weighted avg	0.99	0.99	0.99	53921

Table 2: Classification Report

Example	**Actual**	**Predicted**
Gatier, P., T. Sinclair, M. Ballance, R. Warner, R. Talbert, T. Elliott, S. Gillies. Places: 658606 (Symeon, Mon.). izdavač: Pleiades. pristupljeno 8. ožujka 2012.	English	Croatian
Bisby F.A., Roskov Y.R., Orrell T.M., Nicolson D., Paglinawan L.E., Bailly N., Kirk P.M., Bourgoin T., Baillargeon G., Ouvrard D. (red.) (2011). "Species 2000 & ITIS Catalogue of Life: 2011 Annual Checklist.". Species 2000: Reading, UK. Diakses pada 24 September 2012.	English	Indonesian

Table 3: Misclassified Examples

References

Martin Thoma. 2018. The wili benchmark dataset for written language identification. *arXiv preprint arXiv:1801.07779*.