Selecting and Weighting N-Grams to Identify 1100 Languages

Ralf D. Brown

Carnegie Mellon University Language Technologies Institute
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
ralf@cs.cmu.edu

Abstract. This paper presents a language identification algorithm using cosine similarity against a filtered and weighted subset of the most frequent n-grams in training data with optional inter-string score smoothing, and its implementation in an open-source program. When applied to a collection of strings in 1100 languages containing at most 65 characters each, an average classification accuracy of over 99.2% is achieved with smoothing and 98.2% without. Compared to three other open-source language identification programs, the new program is both much more accurate and much faster at classifying short strings given such a large collection of languages.

Keywords: language identification, discriminative training, n-grams

1 Introduction

Language identification (and the related task of encoding identification) is a useful first step in many natural language processing applications. Increasing numbers of languages have significant online presence, but most prior work on language identification focuses on only a handful of (usually European) languages. As in many language identification techniques, the method presented here relies on n-gram statistics trained from text, but it gains much of its accuracy from *which* n-grams are selected for inclusion in a language model and how their statistics are converted into weights.

2 Method

The method used by our `whatlang` program is a k-nearest neighbor approach using cosine similarity (normalized inner product) of weighted byte n-grams as the metric, which permits each n-gram to be individually weighted according to the strength of the evidence it provides for – or *against* – the hypothesis that the input is in a particular language. One or more language models are trained for each combination of language and character encoding one wishes to identify. The cosine similarity scores are computed incrementally by adding the weighted score of each n-gram match between the input and the models to the overall score for each model, and the languages (and optionally character encodings) corresponding to a user-specified maximum of k highest-scoring models (provided $score \geq 0.85 \times highest$) are output as guesses.

Observing that successive strings are more likely to be in the same language in most texts, the scores for the current input string may be smoothed by adding in a portion of the scores from the immediately prior string(s). The smoothed score is an adaptive linear interpolation between the current string's score vector and an exponentially-decaying sum of all previous strings' score vectors. The interpolation takes into account the current string's length and the maximal score of any model; the higher either of these two factors is, the less weight is given to the previous scores. To avoid excessive smoothing, the inter-string score smoothing weights were tuned on held-out data such that the error rate matched the un-smoothed error rate when the language changed after every fourth string, and was better if the same language occurred at least five times consecutively.

Given the task of selecting the most useful n-grams to populate a language model of limited fixed size, one should focus on the most frequent n-grams. However, even among the top n-grams, there are those which are clearly not indicative of a language (e.g. strings of digits or whitespace), and some which are less informative if certain other n-grams are already included in the model.

`whatlang` treats the input as an untokenized stream of bytes, allowing n-grams to capture multi-word phenomena. The baseline language model extracts from the training data the K highest-frequency three- through N-grams which do not span a linebreak, start with a tab character, two blanks, three digits, or three identical puncation marks. N is set to 6, 10, or 12 for language/encoding pairs which are predominantly one, two, or three bytes per character. Previous experiments [1] showed that 6 is the optimum fixed length limit for K of 3000 and 9000, and the longer limits for multi-byte situations account for the reduced information content of all but the final byte of a character where the script for a language fits into a 256-codepoint block.

The first modification to the baseline model is to filter out n-grams which are substrings of other n-grams also in the model but which contribute little additional information about the language. For example, if the 8-gram "withhold" is in the language model, the 7-gram "withhol" and six-gram "ithhol" would not be informative as all three strings occur the same number of times in the data. The substring need not occur exactly the same number of times in the training data for its removal to be useful, so the removal threshold is a tunable parameter. A threshold of 0.62 (the optimum found for the held-out development set described in Section 4) means that "withhold" would have to occur at least 62% as often as "ithhol" for the latter to be excluded. Removing such uninformative n-grams frees space in the fixed-size model for alternative n-grams.

The second modification to the baseline model is to use discriminative training to add n-grams which provide negative evidence (what we will call "stopgrams"). To compute the appropriate stopgrams for a language model, we first train baseline models for all languages, then select those baseline models whose cosine similarity relative to the baseline model for the language being trained is above some threshold, typically in the range of 0.4 to 0.6. The union of the n-grams in those selected models is formed, and the training text is scanned for occurrences; any which never appear in the training data are added to the baseline model as stopgrams, with an appropriate weight as discussed in the next section. Although this adds n-grams to individual models, the global set of n-grams remains unchanged since the additions were already present in another model.

Weighting the n-grams in a language model is as important as selecting them. A simple uniform weight would not allow an n-gram shared between multiple models to distinguish between them, even though that n-gram is stronger evidence for a language in which it occurs more frequently. On the other hand, using the actual probability of occurrence may over-state differences resulting largely from the relative small amounts of training data. Another factor to be examined in weighting an n-gram is its length, since longer n-grams will typically occur less frequently but are expected to be individually more informative as they are less likely to be encountered by chance.

Empirically, the best weight for an n-gram G was determined to be

$$probability(G)^{0.27} \times length(G)^{0.09}$$

where the probability is simply the frequency of occurrence in the training data divided by the training data's size. These parameters were tuned using the held-out development set but their values proved not to be critical. Varying the exponent for $length(G)$ over the range 0.0 to 1.0 resulted in baseline error rates ranging from 2.414 to 2.421%, a relative change of less than 0.3%. Error rates varied less than 0.5% (relative) for $probability(G)$ exponents ranging from 0.20 to 0.50.

When using discriminative training, stopgrams need to be weighted separately from baseline n-grams; three additional factors contribute to a stopgram's weight. Within the union set of n-grams used as candidate stopgrams, each is weighted by the maximum cosine similarity of any of the individual language models containing it times the maximum baseline score within those models. However, when there is only a small amount of training data available for a language, an n-gram may fail to occur in the training data not because it is not permitted by the language, but simply due to lack of data. Thus, for training sizes less than 2,000,000 bytes, the weight of each stopgram is discounted by a factor of

$$\left(\frac{\max(0, ||t|| - 15000)}{2000000 - 15000}\right)^{0.7}$$

The exponent of 0.7 compensates for the power-law distribution of the most frequent n-grams by making the weight increase more rapidly for small training sizes than a simple linear discount. Finally, an overall weight is given to stopgrams. Their occurrence in the input is strong negative evidence; empirical tuning on the held-out set confirms this with an optimal global weight of -9.0.

3 Related Work

One of the earliest uses of n-grams for language identification, Cavnar and Trenkle's [2] rank-order statistics of the (usually 400) most frequent 1- through 5-grams in the training and test data, has become quite popular; numerous implementations are available, typically including models for between 70 and 100 language/encoding pairs. On a 14-language collection, they reported 98.6% accuracy for documents of 300 or fewer bytes using models with 300 n-grams, trained on 20K to 120K of text.

Ljubesic *et al* [3] use a form of discriminative training to distinguish among Croatian, Slovenian, and Slovak. The discriminative training consisted of stop-word lists for Croatian and Serbian containing words which appear at least five times in the training data for one language but never in the training data for the other. The occurrence of *any* word on a stop-word list in a document tentatively classified as that language

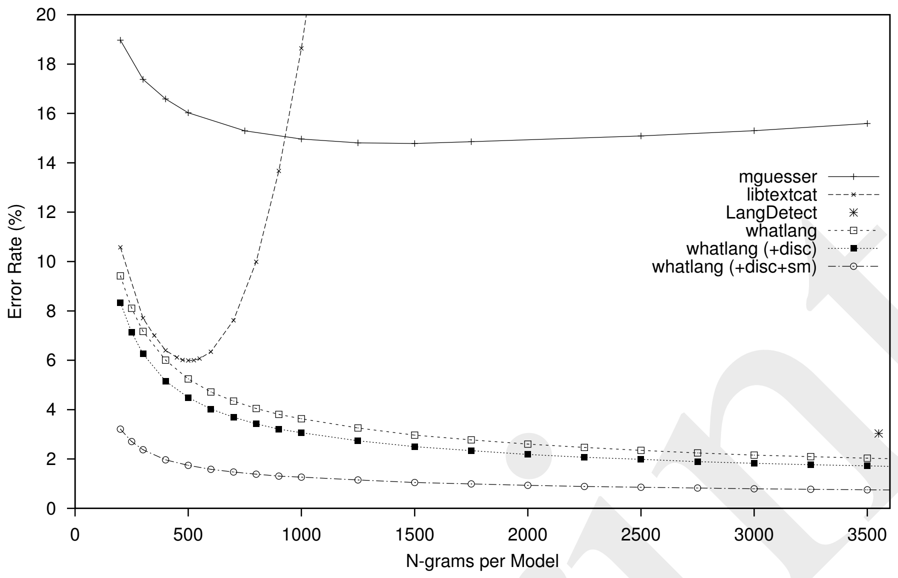

Fig. 1. Average accuracy over 1100 languages when using 200 to 3500 n-grams per language model. `whatlang` was tested in a baseline configuration and with discriminative training and inter-string smoothing enabled. LangDetect averaged 5642 n-grams per model.

would switch the classification if no stop-words for the other language were present. The authors report a final document-level accuracy among the three languages above 99%.

Ahmed *et al* [4] use an incremental inner product. For 50-byte test strings over a twelve-language collection, accuracy was 88.66% for a Naive Bayes classifier, 96.56% for rank-order statistics, and 97.59% for inner product.

Carter *et al* [5] use semi-supervised priors on Twitter messages to bias language identification based on the assumption that a particular user will only post in a limited number of languages, that conversations will remain in the same language and that pages linked from tweets will be in the same language as the tweet. On a five-language collection, they found that applying these priors improved overall accuracy across the five languages from 91.0% to 93.2%.

Very few published results for language identification involve more than 20 languages. Damashek [6] reported an experiment visualizing the similarities of 31 languages, and Shuyo [7] reported 99.8% average accuracy at the document level for news articles in 49 languages. Xia *et al* [8] used information from the containing document (primarily the occurrence of the language name) to help classify Interlinear Glossed Text examples in 638 languages with an accuracy of 84.7 to 85.1%. Some closed-source offerings now provide identification for large numbers of languages; the most comprehensive we have found to date is Likasoft's Polyglot 3000 [9], which claims to support more than 400 languages.

4 Training and Test Data

The greatest difficulty in performing language identification for more than 1000 languages is in actually obtaining text in that many languages, properly labeled by language. Wikipedia is available in 285 language versions as of this writing, of which some 200 have sufficient text to be useful for our purposes. Translations of the full Christian Bible have been made into at least 475 languages and the New Testament has been translated into a further 1240 [10]; hundreds of them have become available as e-texts since early 2010. The New Testament is large enough at around one million characters to be useful as training data. Bible translations were obtained from web sites such as bible.is, ScriptureEarth.org, PNGScriptures.org, and GospelGo.com; they include languages from all continents except Antarctica, ranging from millions of speakers to nearly extinct. Scripts used by these languages include, among others, Arabic, Cyrillic, Canadian Aboriginal Syllabics, Devanagari, Han, Khmer, Lao, Sinhala, Tamil, Telugu, Thai, and Tibetan.

Wikipedia data was obtained by downloading pages linked via the

{langcode}.wikipedia.org/Special:PrefixIndex

search page and extracting the main entry from each such page. The resulting lines of text were sorted, removing duplicates and lines which were unambigously English, French, or German as well as long sequences of ASCII in non-Latin scripts, and given some manual cleaning such as eliminating most occurrences of templates like "X is a city in Y with a population of Z" to avoid skewing the statistics. Despite this cleaning, there is still a fair amount of pollution from other languages in each language's data.

The available data for each language was split into training and test sets in one of two ways. For Bible translations organized as one file per chapter, the first verse of each chapter was held out as test data and the remainder used for training. In all other cases, a uniform sample of every 30th line was held out as test data. The held-out text was converted into test strings by word-wrapping the lines to 65 characters or less and then filtering out any wrapped lines containing fewer than 25 bytes as well as any resulting lines in excess of 1000 (using a New Testament as training data results in approximately 700 test strings).

For the 153 languages for which the above train/test split produced more than 4 million bytes of training data, the training file was split again, reserving a uniform sample of every 30th line as a development set for parameter tuning.

The final collection of languages for `whatlang` contains a total of 1119 languages. Nineteen of the languages (kept as potential confounders) do not have enough data to form a useful test set of at least 100 test strings, leaving 1100 languages for testing. Because a number of languages use multiple scripts, there were 1129 test files containing a total of 824,171 lines.

5 Experiments and Results

Multiple sets of language models were trained with differing numbers of n-grams per model, differing amounts of training data per model, varying substring filtering thresholds, and varying stopgram weights. We evaluated `whatlang` by identifying the language of each of the test strings in each of the 1100 languages using each set of models,

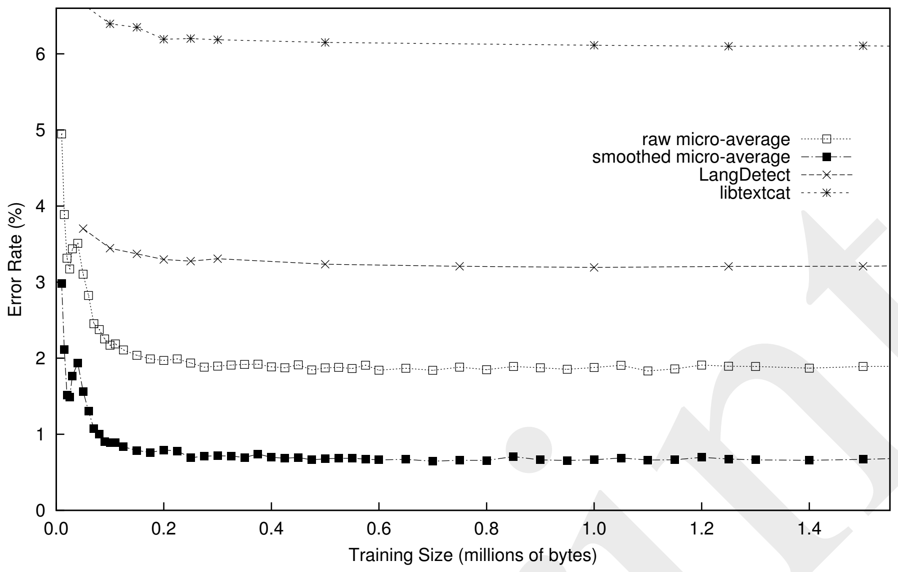

Fig. 2. Average accuracy over the 184 languages with at least 3 million bytes of training data when trained with 15,000 to 1,500,000 bytes of text per language model.

and computing micro-average and macro-average classification error rates (total errors divided by total classifications, and average of per-language error rates, respectively) under multiple conditions, including with and without substring filtering applied to the language models and with and without inter-string score smoothing.

Three other open-source language identification programs were trained and run on the same data. `libtextcat`, version 2.2-9 [11], is a C reimplementation of the Cavnar and Trenkle approach. `mguesser` version 0.4 [12] is part of the mnoGoSearch search engine; its similarity computation is an inner product between 4096-element hash arrays for the input and each trained language, where the elements are normalized to a mean of zero and standard deviation of 1.0. `LangDetect` [13] version 2011-09-13 is a Java library using the Naive Bayes approach. All three packages' identification programs were modified to optionally provide language identification on each line of their input rather than on the entire file, and `libtextcat` and `LangDetect` were modified to process the entire input rather than a small initial segment in whole-file mode to match `whatlang`. `LangDetect`'s limitation of one model per language code was worked around by adding disambiguating digits to the language code during training and removing them from its output prior to scoring.

Figure 1 compares the accuracies of the programs as the size of the language models is varied from 200 n-grams per model to 3500; `libtextcat` was not evaluated beyond 1100, and `LangDetect` does not provide control over the model size (average 5642 n-grams per model for 1 million bytes training data, with 23 models containing more than 15,000 n-grams). The weighted cosine-similarity approach clearly dominates

Program	Model Size	Training Time	Error Rate	Evaluation by line	Evaluation by file	Disk Space	Memory
whatlang	500	763s	1.743%	38s (25s)	35s (21s)	16 MB	16 MB
whatlang	3500	1622s	0.754%	67s (51s)	63s (47s)	97 MB	94 MB
libtextcat	500	583s	5.950%	2138s (1975s)	125s (4s)	4.9 MB	18 MB
mguesser	1500	166s	14.783%	15482s (15218s)	268s (26s)	19 MB	76 MB
LangDetect	5642	699s	3.035%	9491s (1158s)	8275s (7s)	40 MB	8250 MB

Table 1. Comparison of run times in seconds and memory requirements for the programs. Times in parentheses exclude program startup (one invocation per test file) and scoring overhead.

at a given model size even without the addition of discriminative training and inter-string smoothing. As it appears to asymptotically approach its optimal performance [1], the choice of model size is a compromise between accuracy and size/speed.

Figure 2 shows how the classification error rate varies with the amount of training data. Only the 184 languages for which models could be built from at least 3 million bytes were used. Accuracy improves rapidly up to 300,000 bytes; the slight "jitter" in error rates for larger training sizes is likely due to the selection of different lines of text when subsampling. The lowest error rate of 1.832/0.662% (raw/smoothed micro-average) is achieved at 1.1 million bytes, compared to `libtextcat`'s 6.065% error rate using 500-element models trained on 2.0 million bytes of data and `LangDetect`'s 3.193% on 1.0 million bytes. Substring filtering reduces the micro-average raw error rate on the development set from 2.459% to 2.330% (-5.2% relative), and the micro-average error rate with smoothing is reduced from 1.231% to 1.141% (-7.3%), both at a threshold of 0.62. On the full 1100-language set, discriminative training reduces the micro-average raw error rate from 2.035% to 1.722% (-15.3%) compared to the baseline, while the error rate with inter-string smoothing is reduced to 0.754% (-62.9/-56.2% relative to baseline/discriminative).

Table 1 compares the four programs for training time, evaluation time, and memory requirements on a hex-core Intel i7 processor at 4.3 GHz. Due to the incremental scoring used by `whatlang`, it runs much faster than `libtextcat` and `mguesser` on small inputs such as the 65-character lines used in these experiments; decreasing model sizes further increases its speed. For large inputs, such as the entire test file for each language (e.g. 1129 language identifications instead of 824,171), there is much less disparity in overall speed. Times are compared both with and without startup and scoring overhead, since `LangDetect` has a very long startup time (over seven seconds), but it evaluates large inputs very quickly as a result of randomly sampling a fixed number of n-grams from the input. The given training times are for single-threaded training and include the discriminative training second pass for `whatlang`. Since individual language models can be built independently of each other, training is highly parallelizable.

6 Conclusions

We have shown that a cosine-similarity approach to language identification scales to large numbers of languages and outperforms the popular rank-order method of Cavnar and Trenkle as well as two other programs in both accuracy and speed on short strings. Removing redundant high-frequency n-grams from the language models reduces overall classification error by 5.2 to 7.3%. Adding negative n-grams to the language models results in a further error rate reduction of 15.3% over the basic filtered model, and smoothing model scores reduces errors by more than a factor of two. The resulting error rate of 0.754% across 1100 languages is an accuracy of over 99.2%, and could be further improved (at the cost of greater resource use) by increasing the model size.

The `whatlang` program is a separately-compilable module within the Language-Aware String Extractor package, available under the terms of the GNU General Public License at `http://la-strings.sourceforge.net/`. Training data for approximately half of the languages used is redistributable under Creative Commons licenses and may be downloaded from the above URL.

The author thanks the reviewers for their feedback, which improved this paper.

References

1. Brown, R.D.: Finding and Identifying Text in 900+ Languages. Digital Investigation **9** (2012) S34–S43
2. Cavnar, W.B., Trenkle, J.M.: N-Gram-Based Text Categorization. In: Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, UNLV Publications/Reprographics (April 1994) 161–175
3. Ljubešić, N., Mikelić, N., Boras, D.: Language identification: How to distinguish similar languages. In Lužar-Stifter, V., Hljuz Dobrić, V., eds.: Proceedings of the 29th International Conference on Information Technology Interfaces, Zagreb, SRCE University Computing Centre (2007) 541–546
4. Ahmed, B., Cha, S.H., Tappert, C.: Language Identification from Text Using N-gram Based Cumulative Frequency Addition. In: Proceedings of Student/Faculty Research Day, CSIS, Pace University. (May 2004)
5. Carter, S., Tsagkias, M., Weerkamp, W.: Semi-Supervised Priors for Microblog Language Identification. In: Proceedings of the Dutch-Belgian Information Retrieval workshop (DIR-2011), Amsterdam (February 2011)
6. Damashek, M.: Gauging Similarity with n-grams: Language Independent Categorization of Text. Science **267**(5199) (1995) 843–848
7. Shuyo, N.: Language Detection Library - 99% over precision for 49 languages (December 2010) `http://www.slideshare.net/shuyo/-language-detection-library-for-java` Accessed 2013-05-30.
8. Xia, F., Lewis, W.D., Poon, H.: Language ID in the Context of Harvesting Language Data off the Web. In: Proceedings of EACL 2009. (2009) 870–878
9. Likasoft: Polyglot 3000 `http://polyglot3000.com` Accessed 2013-05-30.
10. United Bible Societies: Scripture Language Report 2011 (2011) `http://-www.unitedbiblesocieties.org/wp-content/uploads/2012/04/-report-TABLE-I-2011-Rec-March-28.doc` Accessed 2013-05-30.
11. Hugueney, B.: `libtextcat` 2.2-9: Faster Unicode-focused C++ reimplementation of libtextcat (2011) `https://github.com/scientific-coder/libtextcat` Accessed 2013-05-30.
12. Barkov, A.: `mguesser` version 0.4 (2008) `http://www.mnogosearch.org/-guesser/mguesser-0.4.tar.gz` Accessed 2013-05-30.
13. Shuyo, N.: Language Detection Library for Java `http://code.google.com/p/-language-detection/` Accessed 2013-05-30.