Non-linear Mapping for Improved Identification of 1300+ Languages

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

Abstract

Non-linear mappings of the form $P(ngram)^\gamma$ and $\frac{log(1+\tau P(ngram))}{log(1+\tau)}$ are applied to the n-gram probabilities in five trainable open-source language identifiers. The first mapping reduces classification errors by 4.0% to 83.9% over a test set of more than one million 65-character strings in 1366 languages, and by 2.6% to 76.7% over a subset of 781 languages. The second mapping improves four of the five identifiers by 10.6% to 83.8% on the larger corpus and 14.4% to 76.7% on the smaller corpus. The subset corpus and the modified programs are made freely available for download at http://www.cs.cmu.edu/∼ralf/langid.html.

1 Introduction

Language identification, particularly of short strings, is a task which is becoming quite important as a preliminary step in much automated processing of online data streams such as microblogs (e.g. Twitter). In addition, an increasing number of languages are represented online, so it is desireable that performance remain high as more languages are added to the identifier.

In this paper, we stress-test five open-source n-gram-based language identifiers by presenting them with 65-character strings (about one printed line of text in a book) in up to 1366 languages. We then apply a simple modification to their scoring algorithms which improves the classification accuracy of all five of them, three quite dramatically.

2 Method

The selected modification to the scoring algorithm is to apply a non-linear mapping which spreads out the lower probability values while compacting the higher ones. This low-end spreading of values is the opposite of what one sees in a Zipfian distribution (Zipf, 1935), where the probabilities of the most common items are the most spread out while the less frequent items become ever more crowded as there are increasing numbers of them in ever-smaller ranges. The hypothesis is that regularizing the spacing between values will improve language-identification accuracy by avoiding over-weighting frequent items (from having higher probabilities in the training data and also occurring more frequently in the test string).

Two functions were selected for experiments:

$$x = P(ngram)$$

$$\text{gamma:} \quad y = x^\gamma$$

$$\text{loglike:} \quad y = \frac{log(1+10^\tau x)}{log(1+10^\tau)}$$

The first simply raises the n-gram probability to a non-unity power; this exponent is named "gamma" as in image processing (Poynton, 1998). The second mapping function is a normalized variant of the logarithm function; the normalization provides fixed points at 0 and 1, as is the case for *gamma*. Each of the functions *gamma* and *loglike* has one tunable parameter, γ and τ, respectively.

3 Related Work

Although n-gram statistics as a basis for language identification has been in use for two decades since Cavnar and Trenkle (1994) and Dunning (1994), little work has been done on trying to optimize the values used for those n-gram statistics. Where some form of frequency mapping is used, it is often implicit (as in Cavnar and Trenkle's use of ranks instead of frequencies) and generally goes unremarked as such.

Vogel and Tresner-Kirsch (2012) use the logarithm of the frequency for some experimental runs, reporting that it improved accuracy in some cases. Gebre *et al* (2013) used logarithmic term-frequency scaling of words in an English-language essay to classify the native language of the writer, reporting an improvement from 82.36% accuracy to 84.55% in conjunction with inverse document frequency (IDF) weighting, and from 79.18% accuracy to 80.82% without IDF.

4 Programs

4.1 LangDetect

LangDetect, version 2011-09-13 (Shuyo, 2014), uses the Naive Bayes approach. Inputs are split into a bag of character n-grams of length 1 through 3; each randomly-drawn n-gram's probability in each of the trained models is multiplied by the current score for that model. After 1000 n-grams, or when periodic renormalization into a probability distribution detects that one model has accumulated an overwhelming probability mass, the iteration is terminated. After averaging seven randomized iterations, each with a random gaussian offset (mean 5×10^{-6}, standard deviation 0.5×10^{-6}) that is added to each probability prior to multiplication (to avoid multiplication by zero), the highest-scoring model is declared to be the language of the input.

The mapping function is applied to the model's probability before adding the randomized offset. To work around the limitation of one model per language code, disambiguating digits are appended to the language code during training and removed from the output prior to scoring.

4.2 libtextcat

libtextcat, version 2.2-9 (Hugueney, 2011), is a C reimplementation of the Cavnar and Trenkle (1994) method. It compiles "fingerprints" containing a ranked list of the 400 (by default) most frequent 1- through 5-grams in the training data. An unknown text is classified by forming its fingerprint and comparing that fingerprint against the trained fingerprints. A penalty is assigned based on the number of positions by which each n-gram differs between the input and the trained model; n-grams which appear in only one of the two are assigned the maximum penalty, equal to the size of the fingerprints. The model with the lowest penalty score is selected as the language of the input.

For this work, the libtextcat source code was modified to remove the hard-coded fingerprint size of 400 n-grams. While adding the frequency mapping, the code was discovered to also hard-code the maximum distortion penalty at 400; this was corrected to set the maximum penalty equal to the maximum size of any loaded fingerprint.[1]

Score mapping was implemented by dividing each penalty value by the maximum penalty to produce a proportion, applying the mapping function, and then multiplying the result by the maximum penalty and rounding to an integer (to avoid other code changes). Because there are only a limited number of possible penalties, a lookup table is pre-computed, eliminating the impact on speed.

4.3 mguesser

mguesser, version 0.4 (Barkov, 2008), is part of the mnoGoSearch search engine. While its documentation indicates that it implements the Cavnar and Trenkle approach, its actual similarity computation is very different. Each training and test text is converted into a 4096-element hash table by extracting byte n-grams of length 6 (truncated at control characters and multiple consecutive blanks), hashing each n-gram using CRC-32, and incrementing the count for the corresponding hash entry. The hash table entries are then normalized to a mean of 0.0 and standard deviation of 1.0, and the similarity is computed as the inner (dot) product of the hash tables treated as vectors. The trained model receiving the highest similarity score against the input is declared the language of the input.

Nonlinear mapping was added by inserting a step just prior to the normalization of the hash table. The counts in the table are converted to probabilities by dividing by the sum of counts, the mapping is applied to that probability, and the result is converted back into a count by multiplying by the original sum of counts.

4.4 whatlang

whatlang, version 1.24 (Brown, 2014a), is the stand-alone identification program from LA-Strings (Brown, 2013). It performs identification by computing the inner product of byte trigrams through k-grams (k=6 by default and in this work) between the input and the trained models; for speed, the computation is performed incrementally, adding the length-weighted probabil-

[1]The behavior observed by (Brown, 2013) of performance rapidly degrading for fingerprints larger than 500 disappears with this correction. It was an artifact of an increasing proportion of n-grams present in the model receiving penalties greater than n-grams absent from the model.

ity of each n-gram as it is encountered in the input. Models are formed by finding the highest-frequency n-grams of the configured lengths, with some filtering as described in (Brown, 2012).

4.5 YALI

YALI (Yet Another Language Identifier) (Majlis, 2012) is an identifier written in Perl. It performs minor text normalization by collapsing multiple blanks into a single blank and removing leading and trailing blanks from lines. Thereafter, it uses a sliding window to generate byte n-grams of a (configurable) fixed length, and sums the probabilities for each n-gram in each trained model. As with whatlang, this effectively computes the inner products between the input and the models.

Mapping was added by applying the mapping function to the model probabilities as they are read in from disk. As with LangDetect, disambiguating digits were used to allow multiple models per language code.

5 Data

The data used for the experiments described in this paper comes predominantly from Bible translations, Wikipedia, and the Europarl corpus of European parliamentary proceedings (Koehn, 2005). The 1459 files of the training corpus generate 1483 models in 1368 languages. A number of training files generate models in both UTF-8 and ISO 8859-1, numerous languages have multiple training files in different writing systems, and several have multiple files for different regional variants (e.g. European and Brazilian Portugese).

The text for a language is split into training, test, and possibly a disjoint development set. The amount of text per language varies, with quartiles of 1.19/1.47/2.22 million bytes. In general, every thirtieth line of text is reserved for the test set; some smaller languages reserve a higher proportion. If more than 3.2 million bytes remain after reserving the test set, every thirtieth line of the remaining text is reserved as a development set. There are development sets for 220 languages. The unreserved test is used for model training.

The test data is word-wrapped to 65 characters or less, and wrapped lines shorter than 25 bytes are excluded. Up to the first 1000 lines of wrapped text are used for testing. One language with fewer than 50 test strings is excluded from the test set, as is the constructed language Klingon due to heavy pollution with English. In total, the test files contain 1,090,571 lines of text in 1366 languages.

Wikipedia text and many of the Bible translations are redistributable under Creative Commons licenses, and have been packaged into the LTI LangID Corpus (Brown, 2014b). This smaller corpus contains 781 languages, 119 of them with development sets, and a total of 649,589 lines in the test files. The languages are a strict subset of those in the larger corpus, but numerous languages have had Wikipedia text substituted for non-redistributable Bible translations.

6 Experiments

Using the data sets described in the previous section, we ran a sweep of different gamma and tau values for each language identifier to determine their optimal values on both development and test strings. Step sizes for γ were generally 0.1, while those for τ were 1.0, with smaller steps near the minima. Since it does not provide explicit control over model sizes, LangDetect was trained on a maximum of 1,000,000 bytes per model, as reported optimal in (Brown, 2013). The other programs were trained on a maximum of 2,500,000 bytes per model; libtextcat and whatlang used default model sizes of 400 and 3500, respectively, while mguesser was set to the previously-reported 1500 n-grams per model. After some experimentation, YALI was set to use 5-grams, with 3500 n-grams per model to match whatlang.

7 Results

Tables 1 and 2 show the absolute performance and relative percentage change in classification errors for the five programs using the two mapping functions, as well as the values of γ and τ at which the fewest errors were made on the development set. Overall, the smaller corpus performed worse due to the greater percentage of Wikipedia texts, which are polluted with words and phrases in other languages. In the test set, this occasionally causes a correct identification as another language to be scored as an error.

Figures 2 and 3 graph the classification error rates (number of incorrectly-labeled strings divided by total number of strings in the test set) in percent for different values of γ. A gamma of 1.0 is the baseline condition. The dramatic improvements in mguesser, whatlang and YALI are quite evident, while the smaller but non-trivial im-

		gamma mapping			*loglike* mapping		
Program	Error%	Error%	Δ%	γ	Error%	Δ%	τ
LangDet.	3.233	2.767	-14.4	0.80	2.889	-10.6	1.0
libtextcat	6.787	6.514	-4.0	2.20	–	–	–
mguesser	15.704	4.330	-72.4	0.39	4.177	-73.4	3.8
whatlang	13.309	2.136	-83.9	0.27	2.146	-83.8	4.5
YALI	9.883	2.313	-76.6	0.20	2.313	-76.6	8.0

Table 1: Language-identification accuracy on the 1366-language corpus. γ and τ were tuned on the 220-language development set; only marginally better results can be achieved by tuning on the test set.

		gamma mapping			*loglike* mapping		
Program	Error%	Error%	Δ%	γ	Error%	Δ%	τ
LangDet.	3.603	3.093	-14.2	0.68	3.083	-14.4	2.3
libtextcat	6.693	6.521	-2.6	1.70	–	–	–
mguesser	14.200	4.936	-65.2	0.40	4.779	-66.3	3.7
whatlang	11.879	2.770	-76.7	0.14	2.772	-76.7	5.6
YALI	8.726	2.972	-65.9	0.09	2.989	-65.7	9.0

Table 2: Language-identification accuracy on the 781-language corpus. γ and τ were tuned on the 119-language development set. `libtextcat` did not improve with the *loglike* mapping (see text).

provements in `libtextcat` are difficult to discern at this scale. Since `libtextcat` uses much smaller models than the others by default, Figure 1 gives a closer look at its performance for larger model sizes. As the models grow, the absolute baseline performance improves, but the change from gamma-correction decreases and the optimal value of γ also decreases toward 1.0. This hints that the implicit mapping of ranks either becomes closer to optimal, or that *gamma* becomes less effective at correcting it. At a model size of 3000 n-grams, the baseline error rate is 2.465% while the best performance is 2.457% at $\gamma = 1.10$.

That the best γ for `libtextcat` is greater than 1.0 was not entirely unexpected. The power-law distribution of n-gram frequencies implies that the conversion from frequencies to ranks is essentially logarithmic, and $\log n$ eventually becomes less than n^c for any $c > 0$. The implication of $\gamma > 1$ is simply that the conversion to ranks is too strong a correction, which must be partially undone by the *gamma* mapping.

Figures 4 and 5 graph the error rates for different values of τ. On the graph, zero is the baseline condition without mapping for comparison purposes; the mapping function is not the identity for $\tau = 0$. It can clearly be seen that `libtextcat` is hurt by the *loglike* mapping, which never reduces values, even with negative τ. Using the inverse of

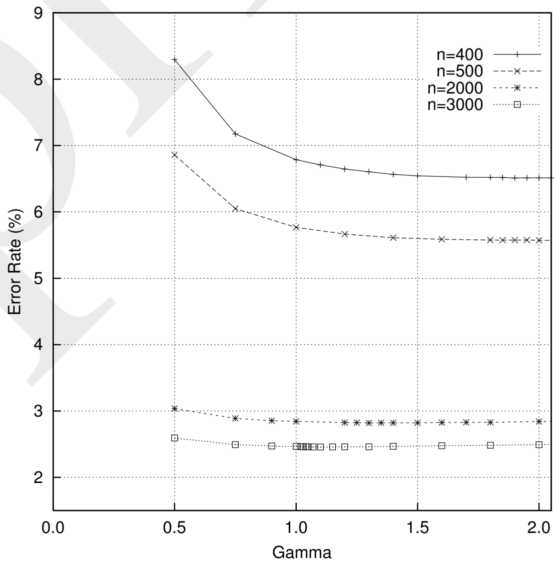

Figure 1: `libtextcat` performance at different fingerprint sizes. $\gamma = 1$ is the baseline.

the *loglike* mapping should improve performance, but has not yet been tried. The other programs show very similar behavior to their results with *gamma*.

8 Conclusions and Future Work

Non-linear mapping is shown to be effective at improving the accuracy of five different language

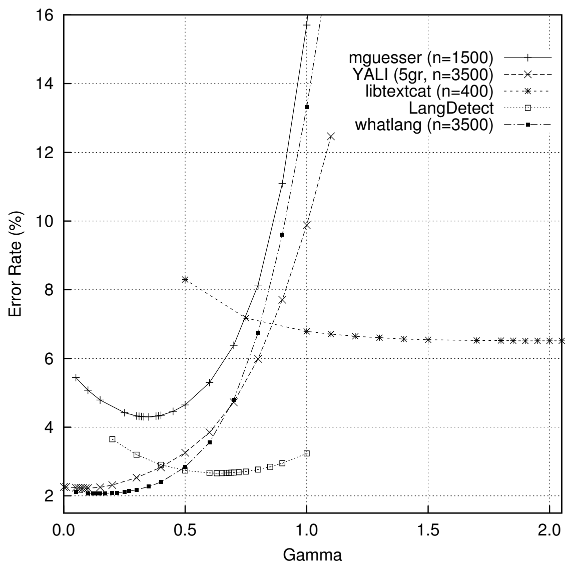

Figure 2: Performance of the identifiers on the 1366-language corpus using the *gamma* mapping.

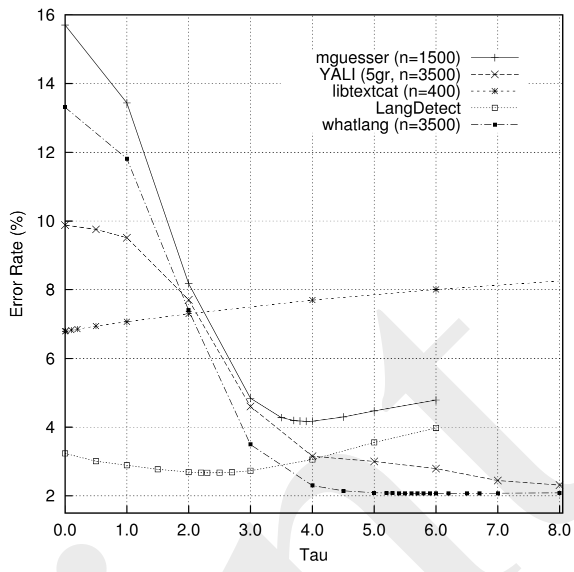

Figure 4: Performance of the identifiers on the 1366-language corpus using the *loglike* mapping.

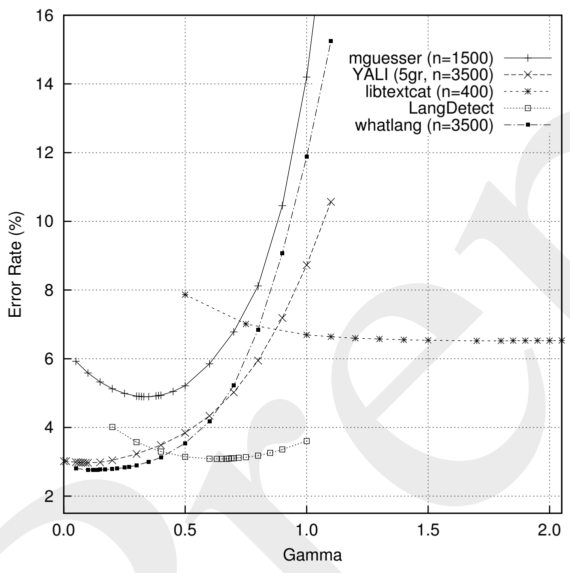

Figure 3: Performance of the identifiers on the 781-language corpus using the *gamma* mapping.

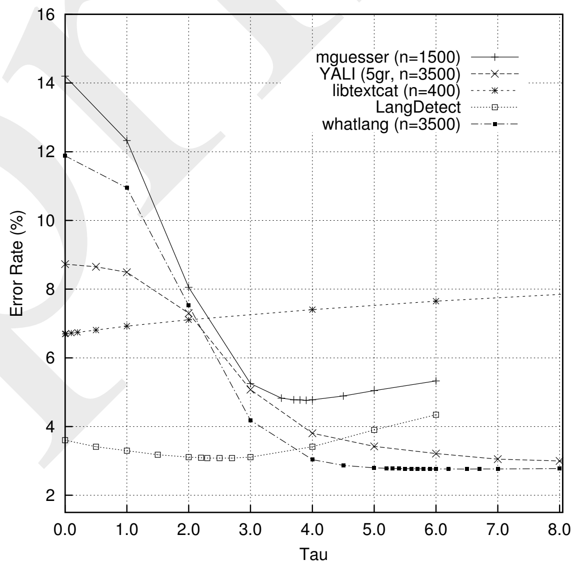

Figure 5: Performance of the identifiers on the 781-language corpus using the *loglike* mapping.

identifier using four highly-divergent algorithms for computing model scores from n-gram statistics. Improvements range from small – 2.6% reduction in classification errors – to dramatic for the three programs with the worst baselines – 65.2 to 76.7% reduction in errors on the smaller corpus and 72.4 to 83.9% on the larger. While both mappings have similar performance for four of the programs, `libtextcat` only benefits from the *gamma* mapping, as it can also reduce n-gram scores, unlike the *loglike* mapping.

Training data, source code, and supplementary information may be downloaded from http://www.cs.cmu.edu/∼ralf/langid.html.

Future work includes modifying additional language identifiers such as `langid.py` (Lui and Baldwin, 2012) and `VarClass` (Zampieri and Gebre, 2014), experimenting with other mapping functions, and investigating the method's efficacy on pluricentric languages like those `VarClass` is designed to identify.

References

Alexander Barkov. 2008. `mguesser` version 0.4. http://www.mnogosearch.org/guesser/-mguesser-0.4.tar.gz (accessed 2014-08-19).

Ralf D. Brown. 2012. Finding and Identifying Text in 900+ Languages. *Digital Investigation*, 9:S34–S43.

Ralf D. Brown. 2013. Selecting and Weighting N-Grams to Identify 1100 Languages. In *Proceedings of Text, Speech, and Discourse 2013*, September.

Ralf Brown. 2014a. Language-Aware String Extractor, August. https://sourceforge.net/projects/la-strings/ (accessed 2014-08-19).

Ralf D. Brown. 2014b. LTI LangID Corpus, Release 1. http://www.cs.cmu.edu/∼ralf/langid.html.

William B. Cavnar and John M. Trenkle. 1994. N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175. UNLV Publications/Reprographics, April.

Ted Dunning. 1994. Statistical Identification of Language. Technical Report MCCS 94-273, New Mexico State University.

Binyam Gebrekidan Gebre, Marcos Zampieri, Peter Wittenburg, and Tom Heskes. 2013. Improving Native Language Identification with TF-IDF Weighting. In *Proceedings of the 8th NAACL Workshop on Innovative Use of NLP for Building Educational Applications (BEA8)*.

Bernard Hugueney. 2011. `libtextcat` 2.2-9: Faster Unicode-focused C++ reimplementation of libtextcat. https://github.com/scientific-coder/-libtextcat (accessed 2014-08-19).

Philipp Koehn. 2005. Europarl: A Parallel Corpus for Statistical Machine Translation. In *Proceedings of the Tenth Machine Translation Summit (MT Summix X)*, pages 79–86.

Marco Lui and Timothy Baldwin. 2012. langid.py: An Off-the-shelf Language Identification Tool. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL-2012)*, pages 25–30, July.

Martin Majlis. 2012. Yet Another Language Identifier. In *Proceedings of the Student Research Workshop at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 46–54, Avignon, France, April. Association for Computational Linguistics.

Charles Poynton. 1998. The Rehabilitation of Gamma. In *Human Vision and Electronic Imaging III, Proceedings of SPIE/IS&T Conference 3299*, January. http://www.poynton.com/PDFs/-Rehabilitation_of_gamma.pdf.

Nakatani Shuyo. 2014. Language Detection Library for Java, March. http://code.google.com/p/-language-detection/ (accessed 2014-08-19).

John Vogel and David Tresner-Kirsch. 2012. Robust Language Identification in Short, Noisy Texts: Improvements to LIGA. In *Proceedings of the Third International Workshop on Mining Ubiquitous and Social Environments (MUSE 2012)*, pages 43–50, September.

Marcos Zampieri and Binyam Gebrekidan Gebre. 2014. VarClass: An Open Source Language Identification Tool for Language Varieties. In *Proceedings of the Ninth International Language Resources and Evaluation Conference (LREC 2014)*, Reykjavik, Iceland, May.

George Kingsley Zipf. 1935. *The Psycho-biology of Language: An Introduction to Dynamic Philology*. Houghton-Mifflin Co., Boston.