Finding and Identifying Text in 900+ Languages

By

Ralf Brown

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2012 USA

Washington, DC (Aug 6th - 8th)

Digital Investigation 9 (2012) S34–S43

Contents lists available at SciVerse ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

Finding and identifying text in 900+ languages

Ralf D. Brown

Carnegie Mellon University, Language Technologies Institute, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Keywords:
Text extraction
Language identification
n-Gram language models
Smoothing
Discriminative training

A B S T R A C T

This paper presents a trainable open-source utility to extract text from arbitrary data files and disk images which uses language models to automatically detect character encodings prior to extracting strings and for automatic language identification and filtering of non-textual strings after extraction. With a test set containing 923 languages, consisting of strings of at most 65 characters, an overall language identification error rate of less than 0.4% is achieved. False-alarm rates on random data are 0.34% when filtering thresholds are set for high recall and 0.012% when set for high precision, with corresponding miss rates of 0.002% and 0.009% in running text.

1. Introduction

Extracting strings of text from among non-text data is an important fall-back capability when analyzing corrupted files or searching for hidden messages. While simple tools to extract sequences of printable characters have been available for decades (e.g. the '`strings`' command which has been a standard tool on Unix almost from its inception), they are very limited in the supported character encodings[1] and do not use language knowledge to guide the extraction or filter out erroneous candidate strings.

In this paper, we present an *n*-gram-based language identification method which also identifies character encodings, and combine the latter with string extraction to determine *which* strings to extract. We then evaluate the language identification accuracy and string extraction accuracy using a database of over 950 languages. Finally, we compare this method with other work in the literature.

2. Language identification

2.1. Training

A language is modeled using the frequencies of the k highest-frequency byte n-grams (for $3 \leq n \leq N$) in training data for that language. Unigrams and bigrams are ignored as insufficiently informative for language identification and too prone to false alarms in character-set identification (bigrams were used initially, but proved to have minimal – and sometimes even negative – effect on classification accuracy). To permit efficient extraction of the most common n-grams for arbitrarily large n from potentially large amounts of training data, multiple passes over the training data are performed.

In the first pass, trigram counts are collected in a simple 256 by 256 by 256 array (using 64 MB of RAM). At the end of this pass, certain non-informative trigrams have their counts zeroed, and the array is filtered to remove all but the $1.5k$ highest-frequency entries. The remaining nonzero trigrams are added to a trie to form the basis for subsequent passes. The trigrams which have their counts zeroed are those beginning with multiple blanks or digits, or consisting of tripled punctuation marks for eight-bit encodings (none of which help to identify the language), as well as those starting with a newline or carriage return character, as those are not considered valid characters for the purposes of string extraction and would merely pollute the model with n-grams that are never seen in extracted strings.

In the second and any further passes, n-grams are added to the trie if they are an extension of one of the filtered n-grams remaining after the prior pass. Each pass will extend the maximum length of an n-gram by up to six bytes, depending on factors such as the size of the training data, the value of k, the previous maximum length, and whether starting positions are forced to offsets which are multiples

E-mail address: ralf@cs.cmu.edu.

[1] Typically, ASCII and ASCII encoded in 16 bits per character; less frequently ASCII encoded in 32 bits per character or ISO 8859-1.

1742-2876/$ – see front matter © 2012 R.D. Brown. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2012.05.004

greater than one. The increment increases in later passes as the data collection is more focused. At the end of each pass, the elements in the trie are enumerated and filtered to eliminate certain less-informative n-grams, and then those whose frequencies are not among the top k (final pass) or $1.5k$ (non-final pass) are removed. A new trie is built from the selected n-grams if there will be further passes. The 50% oversampling on each non-final pass allows for skewed distributions where the k highest-frequency $(n+m)$-grams are not all extensions of the k highest-frequency n-grams.

The filtering at the end of a pass defines "less-informative" n-grams as those which are prefixes of some longer $(n+m)$-gram whose frequency is at least 90% of the n-gram's frequency. The rationale is that the $(n+m)$-gram, with its much higher weight, will almost always be seen whenever the n-gram is seen, and thus the shorter string will not help to distinguish between models. That this filtering is indeed beneficial was proven in early experiments when a small but noticeable (about 0.1% absolute) jump in error rate occurred at the point where the combination of training data size and k resulted in a decrease in the amount by which each pass through the data could extend n-gram lengths, making the filtering less effective since n-grams from a previous pass are not removed by the filtering. To avoid this artifact in the results, the switch-over point was increased such that all experiments reported in this paper used identical training regimens regardless of other parameter settings.

After the final pass, frequency counts are normalized by the size of the input and the relative frequency information is written as the language model.

The above describes the training of baseline language models. Because many languages are quite similar to other languages, we also apply discriminative training in a manner similar to Ljubešić et al. (2007): n-grams which never occur in the training data for language L are considered to be evidence *against* that language if they are frequent in another sufficiently-similar language L'. For the purposes of discriminative training, "frequent" means that the n-gram appears in the baseline model of L', and "sufficiently-similar" means that the cosine similarity between the baseline models of L and L', interpreted as term vectors of the n-grams in the model, is above some threshold. The thresholds should be set high enough that only languages which experience substantial confusability are considered "sufficiently similar", as setting them lower unnecessarily bloats the language models. For UTF-8 and eight-bit encodings, the threshold is usually 0.72 and occasionally 0.80; for UTF-16BE and UTF-16LE, the threshold is raised to 0.85 or 0.90 since their byte sequences inherently have higher cosine similarities.

Discriminative training for a language L proceeds by collecting the union of n-grams of length 3 or greater in the baseline models for all languages L' with similarity above the threshold, and weighting each n-gram by

$$\max_{L'}(cos(L,L') \times f(n,L'))$$

where $f(n,L')$ is the relative frequency of the n-gram in the baseline model for L'. The training text is then scanned for matches against this union, and any members of the union which are not found in the input are added to the baseline model for L as "stop-grams".

2.2. Scoring

Each n-gram in the input which matches the corresponding n-gram in a language model is considered evidence either *for* the hypothesis that the string or block of bytes is written in the particular language (in the case of regular n-grams), or *against* that hypothesis (in the case of stop-grams). Thus, the score for the text being identified relative to an individual language model is simply the sum of the weights associated with each n-gram match between the input and the model. The weight of an n-gram match M is

$$\frac{f(M)^{0.25}\|M\|^{1.25}}{\|input\|}$$

where $f(M)$ is the relative frequency of occurrence in the training data and $\|M\|$ is the length of the n-gram. This exponential combination of the two factors provides a means of setting their relative importance; an exponent of less than 1.0 on the frequency results in smoothing to counteract the often very small absolute frequency in the training data, while an exponent greater than 1.0 on the length counteracts the overall lower frequencies of longer n-grams. The values of the exponents were determined empirically. Division by the length of the input ensures normalized scores; stop-grams frequencies are already discounted by the inter-language similarity score as described in Section 2.1 and are additionally multiplied by the empirically determined penalty factor of -20.

The above summation is simply an incremental method for computing the inner product (dot product) between the language model, interpreted as a frequency-weighted term vector of n-grams, and the test string, interpreted in the same way, since

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{N} u_i v_i$$

where the term $u_i v_i$ is accumulated incrementally by adding u_i for each of the v_i occurrences of the n-gram in the test string. This is closely related to the cosine similarity metric popular in information retrieval[2]

$$\cos(\vec{u},\vec{v}) = \frac{\vec{u} \cdot \vec{v}}{\sqrt{\vec{u} \cdot \vec{u}}\sqrt{\vec{v} \cdot \vec{v}}}$$

The denominator may be omitted while still selecting the same model as highest-scoring because the input string remains constant across language models and the individual language models have already been normalized (thus, $\sqrt{\vec{u} \cdot \vec{u}} = 1$ for every model).

Matches against a language model may be restricted to begin at a multiple of 2 or 4 bytes from the beginning of the input. This is useful for encodings such as UTF-16 in which characters take up a uniform number of bytes greater than one. Without such a restriction, it can be quite

[2] Damashek (1995) used it for topic categorization, but also reported an experiment using cosine similarity of trigrams to visualize the similarity of 31 languages.

difficult to distinguish between big-endian and little-endian versions of an encoding. For example, the hexadecimal byte sequence

00 20 04 10 04 11 04 20 00

could represent either UTF-16LE

0420 0410 0411 0020 “РАБ ”

preceded by a NUL byte, or UTF-16BE

0020 0410 0411 0420 “ АБР”

followed by a NUL. By restricting matches to begin only on every other byte, and training the models with the same restriction, only big-endian Cyrillic UTF-16 models will have matches in this example, while the little-endian versions will not.

The vector of scores with respect to each language model forms a fingerprint of the input. The element with the highest score is considered to indicate the language and character encoding of the input, and its associated language name is extracted. In some cases, there is no clear-cut highest score, so up to m (default 2) language names may be output provided their scores are at least 0.85 times the highest score.

2.3. Smoothing

Since it is unusual for the language to change with every successive string,[3] identification accuracy can be improved by biasing the scores toward those of the preceding lines of text. This becomes particularly relevant for the shortest strings, which have few matches against the language model and thus receive low and/or unreliable scores.

A simple exponential decay combined with adaptive interpolation proves to work quite well. The smoothing vector $\overrightarrow{S_i}$ for the ith string is

$$\overrightarrow{S_i} = \sum_{j=1}^{i-1} \frac{\overrightarrow{R}_{i-j}\left(1 + \frac{ln\|R_{i-j}\|}{8}\right)}{4^j}$$

where $\overrightarrow{R}_i$ is the vector of raw scores of the ith string with respect to the individual models and $\|R_i\|$ is the length of that string in bytes. The length-based factor gives additional weight to longer strings, whose scores are more reliable (recall that the raw scores are length-normalized). $\overrightarrow{S}_i$ is computed incrementally by simply dividing the previous smoothing vector by 4 and adding a weighted copy of the current raw-score vector.

The interpolation weight λ is computed based on the maximum score among all language models and the cube root of the string length, and the final score vector for a string is then

$$\lambda\overrightarrow{R}_i + (1-\lambda)\overrightarrow{S}_i$$

The smoothing parameters, particularly those used to compute λ, have been tuned to avoid excessive smoothing, such that the overall error rate is reduced even when the language switches after every fifth string. Although the smoothing cuts the error rate in half on long monolingual blocks (see Figs. 1and 3), it does occasionally induce an error in the first string following a language switch, where the scores of the prior context are sufficiently high to override the score of the new text and delay reporting a switch in languages by one line. It can also increase the false-alarm rate for non-text data following text since the smoothing will increase the score of the otherwise very low-scoring candidate and allow it to pass the filtering threshold.

3. String extraction

Our string extraction uses the usual notion of S or more consecutive valid characters as forming a textual string, but extends it with knowledge of a large number of character sets. This knowledge allows using not merely byte ranges, but actual codepoint values in the character set to determine whether a byte sequence is valid for the purposes of deciding whether a string should be extracted. For example, the byte sequence E2 81 A5 20 is a well-formed UTF-8 string, but the first three bytes represent the codepoint U + 2065, which is not valid as of Unicode 6.1 (The Unicode Consortium, 2012).

On its own, the ability to specify any of a large number of encodings is useful but limited, since there may be multiple encodings in use within a file (e.g. Microsoft Powerpoint .ppt files may contain both ASCII and ASCII encoded in 16 bits per character). By applying the language models described previously to a segment of the file at a time, the proper encoding to use for extraction can be determined segment by segment. Identification is applied to 320-byte blocks starting every 256-bytes to balance responsiveness and accuracy – smaller blocks allow better detection of changes in encoding, but provide fewer opportunities for matches against the models. The 64-byte overlap ensures that strings which cross a 256-byte boundary are not missed as a result of having their n-gram matches split between blocks and the reduced score for either the first or both blocks failing to exceed the encoding-detection threshold.

Blocks of the file are scored as for language identification, except that stop-grams are ignored (as they may unnecessarily lower the score to the point that a string may be missed). Then the *encoding* associated with each model is examined, rather than the language name, and the highest-scoring encodings (those with scores of at least 0.3 times the maximal score) are used to attempt string extraction. ASCII strings are always extracted, and UTF-8 strings are extracted if there are multiple valid multi-byte UTF-8 codepoints in the block; either may subsequently be filtered due to low confidence scores and not be displayed.

Each extracted string is given a confidence score which is a combination of a factor based on its length (longer strings are less likely to be noise), some heuristics such a proportion of alphabetic characters and number of transitions between alphanumeric and punctuation/whitespace characters, and the language identification score. Strings whose scores fall below a specified threshold are not displayed. The user may select either a pre-defined high-recall threshold (the

[3] The one common exception is localization data, where all of the translations of a program message may be stored consecutively.

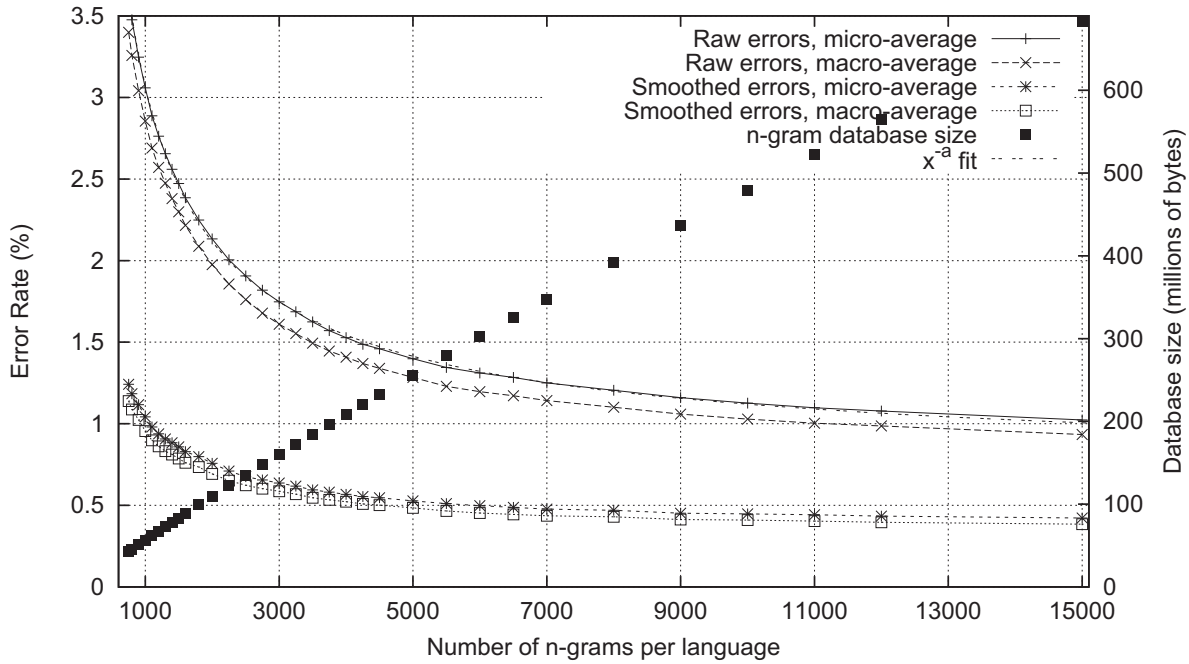

Fig. 1. Comparison of error rates and database size as the number of modeled *n*-grams per language increases, holding *n*-gram lengths and amount of training data constant.

default), a pre-defined high-precision threshold, or any arbitrary threshold. As most non-text strings receive scores less than 7 or 8, the pre-defined threshold values are 7 for high recall and 10 for high precision.

The combination of language identification and string extraction is implemented in the `LA-Strings` (Language-Aware String Extractor) program, which is available from http://la-strings.sourceforge.net/ under the terms of the GNU General Public License version 3. Pre-built models for all of the languages used in the experiments described in this paper are included with the source code. The training data for approximately half of the languages on which `LA-Strings` has been trained is redistributable under Creative Commons licenses, and has been made available in the SourceForge project area.

`LA-Strings` also provides transcoding and romanization support, such that all extracted strings may optionally be transcoded into the UTF-8 encoding and, where appropriate, a second line is displayed containing a romanized version of the extracted string.

4. Experiments and results: language classification

`LA-Strings` was trained on a large number of languages from multiple sources. For the majority of the languages for which models were generated, the training data consisted of a translation of the Christian Bible, most often the New Testament and less frequently both Old and New Testaments or just one or two books of the New Testament. Other sources of language training data include European parliamentary proceedings from the Europarl corpus, version 5 (Koehn, 2005) and non-English versions of Wikipedia.

Models were trained for multiple encodings of each language, typically either ASCII or UTF-8 and both UTF-16 big-endian and UTF-16 little-endian. Where appropriate, additional eight-bit encodings including CP-862, Latin-1 (ISO 8859-1), Latin-2, Latin-3, Latin-5, TSCII, VISCII, Windows-1251, and Windows-1256 were trained as well. Various languages exist in multiple scripts (e.g. Ladino may be written in either Hebrew or Latin characters, and Turkmen in either Cyrillic or Latin characters), resulting in additional models for the language. The final language identification database contained a total of more than 3100 language models. Training required between 13 and 45 min on a quad-core AMD Phenom II CPU at 3.2 GHz, depending on explicit limits to training data and the number and length of *n*-grams in the models.

To test the performance of the language identification, a portion of the training data was held out. In cases where Bible text was used and the electronic text was organized as one file per chapter, the first verse of each chapter in each book of the Bible was held out as testing data. Otherwise, a uniform sample of every thirtieth line or verse of the training material was held out. In a few cases, a slightly higher proportion was held out to obtain at least 100 test strings.

The held-out data was word-wrapped to at most 65 characters using the GNU 'fold' program and then filtered to remove resulting lines consisting of fewer than 25 bytes with the commands

```
fold -s -w 65|LC_ALL = C grep -E '^.{25}'
```

and up to the first 1000 filtered lines became the test set for our experiments. Sixteen languages for which the preceding construction resulted in fewer than 100 test strings were

omitted from testing, but remained in the language identification database. Languages trained on just the New Testament typically resulted in 500–700 test strings.

In all, 941 test files for 923 languages (thirteen languages included two differing scripts each, one included four variants in three scripts, and two others included two files from differing sources) were used to characterize the performance of the language identification method. Classification accuracy was tested by performing language identification with and without inter-string score smoothing and counting the number of lines whose top-scoring language did not match the language of the containing test file's strings. The micro-average error rate was computed by dividing the total number of mis-classified strings by the total number of test strings (685,272), while the macro-average error rate was computed as the mean of the per-language error rates. The latter places equal weight on performance in every language, while the former places greater weight on the languages with more test strings, which are also the languages with more training data.

Because many of the languages included in the full collection have very few speakers, some of the experiments were repeated on the frequent-language subset. The subset used 98 test files for 93 languages; as with the full set, some languages include multiple scripts.

As can be seen in Fig. 1, language identification accuracy improves smoothly and asymptotically with an increase in the number of most common *n*-grams modeled per language. For the smaller sets of languages in Fig. 2, the progression is not quite as smooth, but still exhibits an asymptotic behavior. It is not the case, as has been reported with similar *n*-gram-based language identification methods, that performance peaks at a relatively small number of *n*-grams (e.g. 300–400 as reported in). As a result of the asymptotic performance, the decision on number of *n*-grams to use is a matter of deciding on a trade-off between resource usage (primarily disk and RAM) and incremental improvements in accuracy.

An interesting point to note in Fig. 1 is that the raw error-rate curve is a very close fit to $ax^{-b} + c$, i.e. performance is proportional an inverse power of the number of *n*-grams in the model plus some residual error rate. Curve fitting of the micro-average raw error rate results in $a = 369.982$, $b = 0.729944$, and $c = 0.675123\%$.

Performance also improves smoothly with increasing amounts of training data, as shown in Fig. 3. While there is a marginal increase in error rate above 2,000,000 bytes per language (where e.g. Ljubešić et al. (2007) reported a definite peak in performance at around 350,000 bytes for second-order Markov models), i.e. from a micro-average raw error rate of 1.590% at two million bytes to 1.617% at three million bytes, this is likely a result of the training data for several languages containing both Bible texts and less reliable data such as that collected from Wikipedia, with the latter placed second in the training file. In addition, the limited amount of training data available for a majority of the languages being tested (the first through third quartile total training data sizes are 1,028,302, 1,394,385, and 2,136,758 bytes), undoubtedly flattens the performance curve for training sizes greater than one million bytes.

Fig. 3 also demonstrates the effectiveness of discriminative training. The top-most curve (for training data of 200,000 or more bytes) is the micro-averaged raw error rate without using discriminative training, while the curve immediately below it is the error rate when discriminative training is applied. For the very smallest amounts of training data,

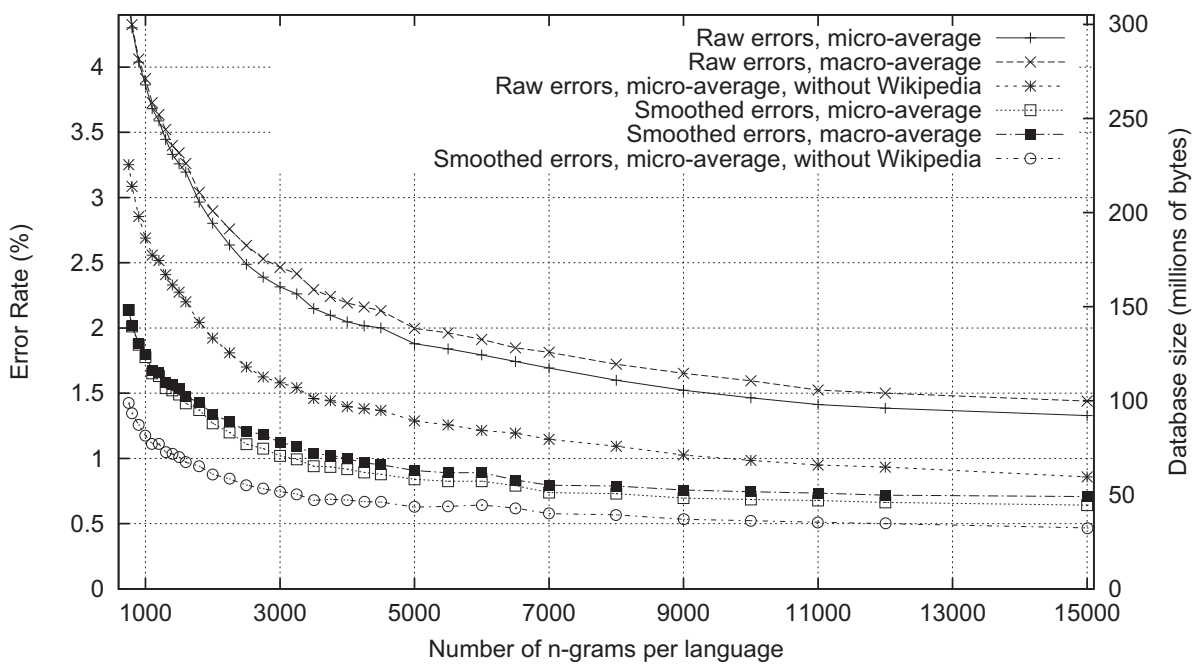

Fig. 2. Comparison of error rates versus the number of modeled *n*-grams per language for subsets of 93 and 79 ("without Wikipedia") of the most-frequently spoken languages.

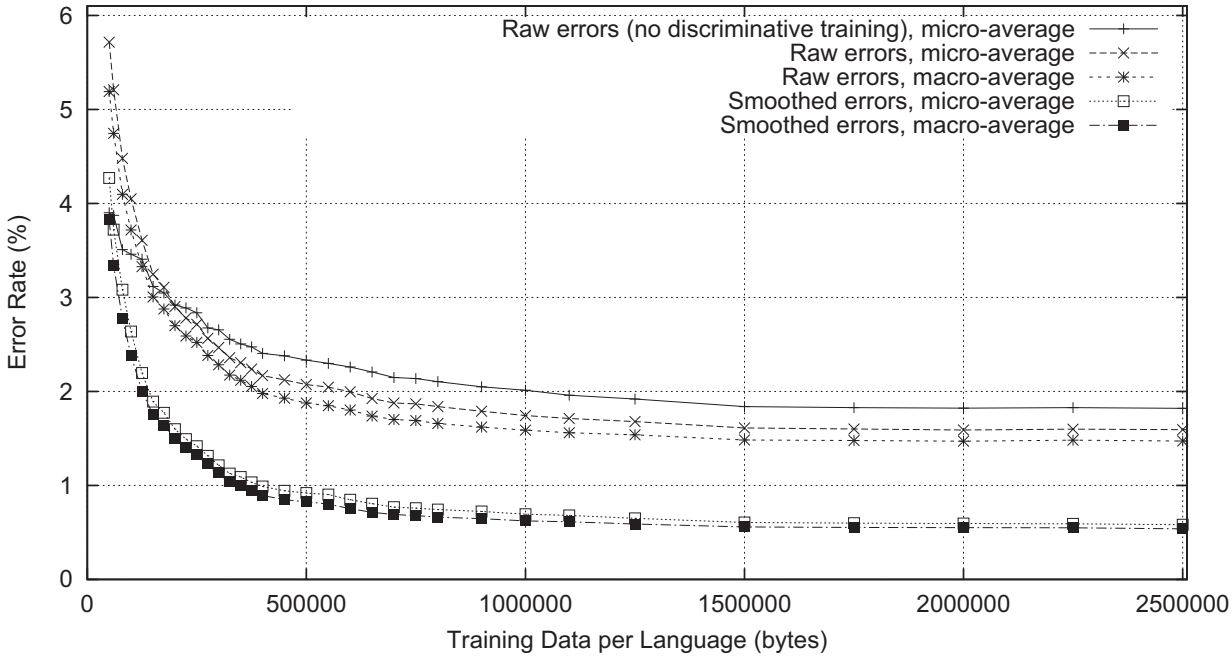

Fig. 3. Comparison of error rates as the available training data per language increases, holding *n*-gram counts and lengths constant.

discriminative training *with its current settings* actually increases the error rate because *n*-grams which are in fact valid for a language are learned as stop-grams because they did not occur in the tiny sample used for training. The other three error-rate conditions (not displayed in the figure) show corresponding improvements.

As shown in the top half of Fig. 4, performance begins to degrade again if the maximum *n*-gram length is increased beyond six bytes when using 3000 *n*-grams per language. This is likely due to using a fixed number of *n*-grams per language, as longer *n*-grams displace less-frequent but still informative shorter *n*-grams. The same experiment using 9000 *n*-grams per language (bottom half of Fig. 4) supports this hypothesis, as the optimal length for smoothed error rate increases to 7-grams and the increase in error rate above the optimal point is less pronounced. It is possible that increasing the number of *n*-grams further will shift the optimum length even higher, but at the cost of a dramatic increase in the size of the language models – the 6-gram model database for 3000 *n*-grams per language is 119 million bytes, while the 7-gram models for 9000 *n*-grams per language total 426 million bytes.

The best result was achieved using 15,000 *n*-grams per model, with a maximum length of six bytes for language/encoding pairs which are predominantly one byte per character and eight bytes for language/encoding pairs which are predominantly multiple bytes per character, limiting training data to at most 2,500,000 bytes per model (the right-most point in Fig. 1). The raw error rate was 1.023% micro-average and 0.934% macro-average, while the smoothed error rate was 0.422% micro-average and 0.385% macro-average.

In all three figures for the full collection of languages, the micro-average error rate is noticeably higher than the macro-average error rate. This implies that, on average, the languages with smaller test sets were more accurate than the languages with larger test sets. Since the test set sizes are proportional to the training data size, one is led to conclude that languages with more training data perform worse overall. This is in fact the case, but is an artifact of the availability of training data – most of the largest training sets are derived from Wikipedia pages, which contain a substantial amount of text in languages other than the primary language of the page, be it parenthetical presentation of a source language version of an expression or untranslated text from the equivalent page in a majority-language version of Wikipedia. Even after semi-automated cleaning, the training sets (and thus also the test sets) are polluted with other languages, which reduces accuracy both by distorting *n*-gram frequencies in the model and by providing an erroneous answer key for some of the test strings. For example, only 68 of the 941 test files are derived from Wikipedia data, yet 22 of the 42 test files with raw error rates of more than 10% in the length = 6, $k = 3000$ case are from Wikipedia. Most of the remaining high-error-rate files are from very closely-related language pairs such as Indonesian/Malay and Bosnian/Croatian.[4]

Fig. 2 shows the effect of excluding the 16 files containing Wikipedia data from the 98-file common-language subset, leaving 82 files in 79 languages. Computed error rates drop by about one-fourth, from a best micro-averaged smoothed error rate of 0.642–0.467%.

[4] Bosnian and Croatian are so closely related that Ljubešić et al. did not attempt to make the distinction in their language identification work because "it is hard even for native speakers to notice the difference between the two of them at a glance" (Ljubešić et al., 2007).

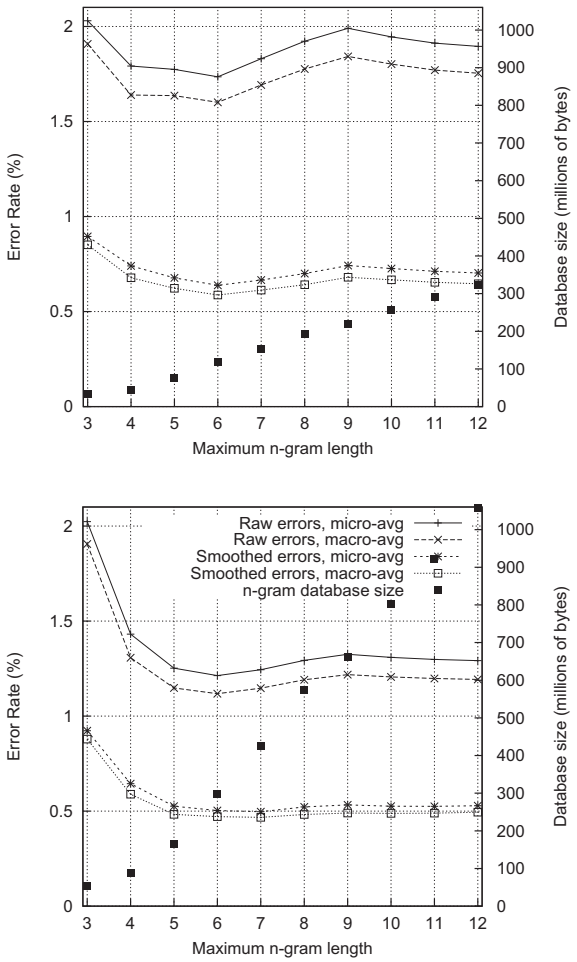

Fig. 4. Comparison of error rates and database size versus the length of the longest modeled *n*-gram, using 3000 *n*-grams per language (top) or 9000 *n*-grams per language (bottom).

By the very nature of the *n*-gram matching approach used, erroneous classifications will tend to be for linguistically similar languages such as the pairs mentioned in the preceding paragraph. Table 1 shows the confusion matrix for a set of Indo-Iranian languages which effectively form a dialect continuum, four members of the Chinese macrolanguage, and several languages of the former Republic of Yugoslavia. Most of the languages in the table are also among the languages with the highest error rates.

A consequence of the tendency for mis-classified results to be for similar languages is that the error rate can be further reduced if one is willing to accept classification as another language which is mostly intelligible to speakers of the actual language of the test input as a correct classification. To simulate such fuzzy scoring, 54 languages were mapped into eleven equivalence classes prior to scoring (for example, Egyptian Arabic and Modern Standard Arabic were considered equivalent, as were Indonesian and Standard Malay). Additional equivalences could have been added, but these capture the majority of the high-error-rate languages. The result is approximately a one-third reduction in error rate; for $k = 15{,}000$, the micro-averaged raw error rate is reduced from 1.023% to 0.678% and the micro-averaged smoothed error rate is reduced from 0.422% to 0.271%.

Table 1
Confusion matrices for three sets of closely-related languages. Top: Indo-Iranian languages. Center: members of the Chinese macrolanguage. Bottom: languages of the former Yugoslavia.

Language	Identified as:			
	glk	pes	prs	Other
Gilaki (glk)	**532**	62	13	17
Western Farsi (pes)	9	**802**	187	3
E. Farsi/Dari (prs)	1	40	**435**	–

Language	Identified as:				
	cmn	yue	gan	wuu	Other
Mandarin (cmn)	**957**	1	–	42	–
Cantonese (yue)	8	**892**	82	18	–
Gan (gan)	9	22	**187**	6	1
Wu (wuu)	98	38	19	**330**	2

Language	Identified as:					
	bs	hr	sk	sl	sr	Other
Bosnian (bs)	**932**	32	–	2	21	13
Croatian (hr)	167	**807**	–	–	6	20
Slovak (sk)	1	–	**487**	1	–	3
Slovenian (sl)	3	2	–	**716**	1	7

Bold-faced numbers indicate correct language identification.

5. Experiments and results: string extraction

Determining the accuracy of string extraction is less straightforward than determining the accuracy of language identification. In preliminary experiments to estimate extraction accuracy, we used files of random data to examine false alarms and counted the number of missed lines in the 941 text files used for language identification experiments. The false-alarm rate was computed as the number of bytes extracted from a 10,000,000-byte file of random bytes created with

```
head-10000000c /dev/urandom
```

averaged over twenty runs with different random data for each run.

We also performed some preliminary investigations of extraction accuracy for a few of the test files when large amounts of random bytes are inserted between each line to simulate extraction of isolated strings. As we have not yet had an opportunity to write a proper scoring program for this situation, the results presented for this situation below are indicative of general, rather than exact, performance.

Table 2 compares `LA-Strings` extraction accuracy to a popular implementation of the Unix '`strings`' utility,

Table 2
False-alarm rates on random data (string length ≥ four characters) and miss rates due to language-model filtering. `LA-strings` is extracting multiple encodings, including Latin-1 and UTF-16.

Tool	False alarms	Miss rate
GNU strings (ASCII)	5.68%	N/A
GNU strings (8-bit)	80.58%	N/A
GNU strings (16-bit ASCII)	0.00%	N/A
`LA-Strings`, high recall	0.338%	0.002%
`LA-Strings`, high precision	0.012%	0.009%

available as part of the GNU Binary Utilities collection (Free Software Foundation, 2012). By default, GNU strings extracts sequences of four or more printable ASCII bytes (or tab characters) terminated by an invalid byte. When given the command line flag '`-eS`', the definition of valid bytes for a string is extended to include all bytes with their high bit set, which permits extractions of strings encoded in ISO 8859-X, EUC, or UTF-8. As can be seen in the table, this renders `strings` effectively useless for any file which is not composed predominantly of NUL bytes. When given the command line flag '`-el`', the definition of valid character is changed to a little-endian 16-bit value containing an ISO 8859-1 (Latin-1) character in the low-order byte and a zero in the high-order byte, i.e. the Latin-1 subset of UTF-16LE, which is commonly encountered in Microsoft Windows executables and Microsoft-based data files. Due to the restriction that every other byte of the candidate string be zero, the false-alarm rate on random data is miniscule: one in 2^{8N}, where N is the minimum length of the string to be extracted. The false-alarm rate will be higher on arrays of 16-bit quantities, but that case has not been tested.

Fig. 5 shows the trade-off between false alarms and loss of the shortest strings as the minimum string length is increased. Eight-bit extraction for GNU strings is not included in the plot, as its false-alarm rate is still 55.9% for a minimum length of ten characters. The figure also shows the effect of the language-model database on false-alarm rates by comparing performance with the full database using the high-recall threshold against performance with the subset of 99 frequent languages. The smaller number of languages provides fewer opportunities for *n*-gram matches to produce language scores above the threshold and thus a lower false-alarm rate, but a ten-fold increase in number of languages in the full collection only resulted in a one-third increase in the false-alarm rate.

In contrast to GNU `strings`, `LA-Strings` uses its language models to determine the encoding used in each 320-byte block of the file so that it can perform targeted extraction using the appropriate definition of valid byte sequences for the detected encoding. The extracted candidates are then further filtered based on a score threshold to produce the extremely low false-alarm rates shown in Table 2. The reported miss rate is the proportion of lines in the language identification test corpus which were not extracted; the miss rate for short, isolated strings will be higher as there may not be enough matching trigrams in the containing 320-byte block to trigger extraction with the appropriate character encoding.

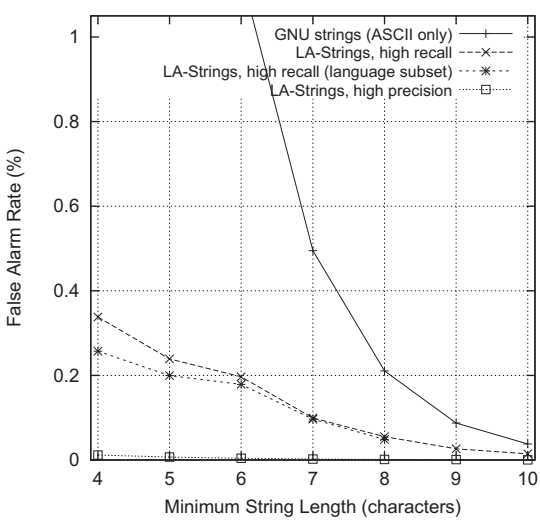

Fig. 5. Comparison of false-alarm rates as the minimum string length is increased.

When extracting isolated strings embedded in large amounts of random data, preliminary investigation indicates that `LA-Strings` extracts a false-alarm string following about one in six true strings in high-precision mode (using the pre-defined high-precision string score threshold to filter extracted strings) and an average of slightly more than one extraneous string per true string in high-recall mode (using the default string score threshold). As these erroneous strings are quite short, the false-alarm rate as a percentage of extracted bytes is much lower.

Classifying languages in the collection of test files (a total of 47 MB) takes 5–10 min, single-threaded, depending on the training parameters. Extraction from random data processes around 1600 KB per second, again using a single CPU core. The disparity in speeds is due to the fixed overhead of scoring the large number of language models; on random data, larger blocks are processed. Since many of the languages are of little practical interest, the `LA-Strings` source code includes an option to build a smaller database of the most-spoken languages according to The Ethnologue (Paul Lewis). This smaller database of some 440 models for 99 languages runs approximately four times as fast as the full database – the 7.8 MB of test text in the subset of languages included in the smaller database requires 12–20 s to classify.

6. Related work

Language identification has been widely researched for decades, with an initial burst of activity on statistics-based classification methods in the mid-1990s. Popular approaches involve lists of common words, unique letter combinations, *n*-gram statistics, or some combination thereof, using classification methods such as Naive Bayes, rank-order statistics, vector space models, and *k*-nearest neighbors.

Cavnar and Trenkle (1994) use rank-order statistics on the most frequent *n*-grams in the training and test data. After stripping digits and most punctuation and tokenizing the input into words padded with blanks at beginning and end, all possible 1-grams through 5-grams are counted. The *n*-grams are sorted by count and the counts are discarded, retaining only the rank ordering, and only the top k *n*-grams are retained to form a "profile" of the training data or the test input. To identify a language, the profile of the input is compared against each of the language profiles by assigning to each *n*-gram a score equal to the number of rank positions by which it differs between the two profiles, or k if it does not exist in the other profile. The language whose profile receives the lowest sum of differences is declared the language of the input.

On a 14-language collection, their method achieved 98.6% accuracy for documents of 300 or fewer bytes and 99.8% accuracy for documents of more than 300 bytes using models with 300 or 400 *n*-grams, respectively, trained on 20–120 K of text.

The Cavnar and Trenkle approach has been implemented numerous times in popular libraries and applications, including van Noord's `TextCat` Perl script (van Noord, 2012), Java and Python reimplementations of `TextCat`, and in the language identification used by the OpenOffice.org/LibreOffice word processing application to select the appropriate spell-checking dictionary. These implementations typically include models for between 70 and 100 language/encoding pairs.

Dunning (1994) distinguished between English and Spanish on inputs as small as ten bytes using Hidden Markov Models (HMMs) which roughly correspond to the product of successive conditional *n*-grams in the text. Laplace's sample size correction was used to smooth probabilities, and the stationary $n+1$-gram probabilities were estimated to form the final model, incorporating evidence from all lower-order models. Dunning reported accuracy of about 92% for 20-byte inputs with 50 KB training data for bigram and trigram models, increasing to more than 99% for inputs of 500 bytes.

Giguet (1995) combined statistics with linguistic knowledge to classify individual sentences among French, English, Spanish, and German. Sub-methods included lists of function words, presence of accented characters, bigram and trigrams, and frequent word endings. Discrimination among the four languages was perfect for sentences of eight or more words, but degraded rapidly for shorter sentences.

Ljubešić et al. (2007) also combine some linguistic knowledge with *n*-gram statistics and additionally employ a form of discriminative training, in order to distinguish among the closely-related languages Croatian, Slovenian, and Slovak and to distinguish between them and other languages. The linguistic filtering consisted of thresholding on the percentage of the 100 most common words and 20 most frequent language-specific characters. 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 appearance of *any* word on a stop-word list in a document tentatively classified as that language would switch the classification if no stop-words for the other language were present. The authors report a final classification accuracy among the three languages in excess of 99% at the document level (news articles, i.e. a few thousand characters of text).

Ahmed et al. (2004) use the same incremental computation of the inner product described in Section 2.2, but coin the name Cumulative Frequency Addition to describe it. They report results on a collection of twelve languages using Latin-based alphabets, employing *n*-grams of sizes 2 through 7 from training data of 65–105 K for the various languages, and compared performance against Cavnar and Trenkle-style rank-order statistics and a Naive Bayes classifier. On 50-byte test strings, classification accuracy among the twelve languages was 88.66% for Naive Bayes, 96.56% for rank-order statistics, and 97.59% for Cumulative Frequency Addition (inner product).

Padró and Padró (2004) compared three different language identification methods (HMMs, Damashek's trigram frequency vectors (Damashek, 1995), and the `TextCat` implementation of Cavnar and Trenkle's rank-order algorithm) and characterized their performance on a six-language news-article collection with varying parameter settings. They found that precision peaked with 25,000 words of training data for rank-order and trigram-vector methods and showed minimal increase beyond 50,000 words for HMMs. HMMs achieved close to maximum precision on texts of just 20 characters when distinguishing between two languages, and 30 characters when distinguishing between all six. Overall, on inputs of less than 150 characters, HMMs substantially outperformed the trigram-vector method, which in turn substantially outperformed `TextCat`.

Tromp and Pechenizkiy (2011) apply language identification specifically to short strings, using Twitter messages as training and test data. They extend the Cavnar and Trenkle approach into a graphical model which captures ordering information among trigrams for multiple languages as language-specific transition probabilities in the graph. Identification is then performed by computing the path-matching scores for an input string in the graph over each language. Classification was tested for a collection containing Dutch, English, French, German, Italian, and Spanish, resulting in an accuracy of up to 98.3%.

Carter et al. (2011) 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. This is similar to our assumption that successive strings are much more likely to be in the same language than in different languages. They tested their algorithm on a collection containing Dutch, English, French, German, and Spanish and found that applying these priors improved overall accuracy across the five languages from 91.0% to 93.2%.

The reader may have noted that nearly all of the above work used very small numbers of languages. A few commercial offerings now provide language identification for dozens or more languages; the most comprehensive we have found to date is from LexTek International, which claims 260 language/encoding pairs for its LexTek Language Identifier SDK (LexTek International, 2012) as of April 2012.

A further unstated assumption of all of the above work is that the input is in fact entirely text, and does not contain non-text portions. The work described here relaxes the assumption by first extracting strings of text using a classification of the character encoding before performing language identification.

7. Conclusions

This paper has presented a method for accurately identifying the language of text for much larger numbers of languages than have been dealt with by a single system in the past, and has combined it with string extraction to perform identification solely on the text portions of

arbitrary input. Discriminative training to determine *n*-grams which are *negative* evidence for a language reduces the error rate by approximately one-third, while grouping mutually-intelligible languages and dialects into equivalence classes can reduce the error rate by a further one-third if one does not require an absolutely precise identification. The performance of the language identification is improved by biasing scores toward the language of the preceding context, on the assumption that consecutive strings are likely to be in the same language.

Model-based filtering results in very low false-alarm rates when attempting to extract strings from random data, yet preserves very low miss rates for true text. The false-alarm rate with thresholds set for high precision is equivalent to just 116 bytes erroneously extracted per megabyte of input, while the corresponding miss rate is equivalent to missing one in every 10,700 strings.

References

Ahmed Bashir, Cha Sung-Hyuk, Tappert Charles. Language identification from text using n-gram based cumulative frequency addition. In: Proceedings of student/faculty research day. CSIS, Pace University; May 2004.

Carter Simon, Tsagkias Manos, Weerkamp Wouter. Semi-supervised priors for microblog language identification. In: Proceedings of the Dutch–Belgian information retrieval workshop (DIR-2011). Amsterdam; February 2011. http://-wouter.weerkamp.com/downloads/poster-dir2011-lid.pdf.

Cavnar WB, Trenkle John M. N-gram-based text categorization. In: Proceedings of third annual symposium on document analysis and information retrieval. Las Vegas, NV: UNLV Publications/Reprographics; April 1994. p. 161–75.

Damashek Marc. Gauging similarity with n-grams: language independent categorization of text. Science 1995;267(5199):843–8.

Dunning Ted. Statistical identification of language. Technical report MCCS 94-273. New Mexico State University; 1994.

Free Software Foundation. GNU binary utilities, http://www.gnu.org/software/binutils/; April 2012.

Giguet Emmanuel. Categorization according to language: a step toward combining linguistic knowledge and statistic learning. In: Proceedings of the international workshop of parsing technologies (IWPT-1995); September 1995.

Koehn Philipp. Europarl: a parallel corpus for statistical machine translation. In: Proceedings of the tenth machine translation summit (MT Summit X). Phuket, Thailand; 2005. p. 79–86.

LexTek International. LexTek language identifier SDK, http://www.lextek.com/langid/li/languages.htm; April 2012.

Ljubešić Nikola, Mikelić Nives, Boras Damir. Language identification: how to distinguish similar languages. In: Lužar-Stifter Vesna, Hljuz Dobrić Vesna, editors. Proceedings of the 29th international conference on information technology interfaces. Zagreb: SRCE University Computing Centre; 2007. p. 541–6.

Padró Muntsa, Padró Lluís. Comparing methods for language identification. In: Procesamiento del lenguaje natural, vol. 33; September 2004. p. 155–61.

Paul Lewis M, editor. Ethnologue: languages of the world. 6th ed. Dallas, TX: SIL International. Online version: http://www.ethnologue.com/.

The Unicode Consortium. The Unicode standard. Version 6.1.0. Mountain View, CA: The Unicode Consortium, http://www.unicode.org/versions/Unicode6.1.0/; 2012.

Tromp Erik, Pechenizkiy Mykola. Graph-based N-gram language identification on short texts. In: Proceedings of the 20th annual Belgian–Dutch conference on machine learning (BENELEARN-2011); 2011.

van Noord Gertjan. TextCat Perl script; April 2012. http://-www.let.rug.nl/vannoord/TextCat/.