Language Identification on the Web: Extending the Dictionary Method

Radim Řehůřek[1] and Milan Kolkus[2]

[1] Masaryk University in Brno, xrehurek@fi.muni.cz
[2] Seznam.cz, a.s., milan.kolkus@firma.seznam.cz

Abstract. Automated language identification of written text is a well-established research domain that has received considerable attention in the past. By now, efficient and effective algorithms based on character n-grams are in use, mainly with identification based on Markov models or on character n-gram profiles. In this paper we investigate the limitations of these approaches when applied to real-world web pages. The challenges to be overcome include language identification on very short texts, correctly handling texts of unknown language and texts comprised of multiple languages. We propose and evaluate a new method, which constructs language models based on word relevance and addresses these limitations. We also extend our method to allow us to efficiently and automatically segment the input text into blocks of individual languages, in case of multiple-language documents.

1 Motivation

The amount of information available on the net is staggering and still growing at a fast pace. To make this information available, applications have sprung up to fill the void and gather, process and present Web information to the knowledge-hungry user. Unfortunatelly, documents on the Web have historically been created with human reader in mind, in formats such as HTML, and are not readily understandable by computers. Although XML and semantic mark-up (e.g. the `xml:lang` attribute, or the `<div lang="en">` construct) have been introduced to alleviate these problems, reality remains that many documents do not make use of metadata tags or, even worse, make use of them incorrectly and provide misleading information.

By not having metadata provided for us, or by deciding not to trust it, we are left with deducing information from the text itself. This is the domain of natural language processing (NLP) and text mining. This article deals with one aspect of text mining, namely telling which language (or languages) is a given Web page written in.

2 Related work

A general paradigm in automated language identification is to create language models during a training phase and compare input document against these mod-

els during language identification. This places the task into the domain of *supervised learning* methods. Another consequence is that the set of target languages needs to be known beforehand, which makes language identification a *classification* problem.

A "common words" approach [1] is based on the observation that for each language, there is small class of words that carry little information but make up a large portion of any text. These are called *function words* or *stop words* and their presence is to be expected as word distribution follows Zipf's law.

In [2] it is noted that humans need surprisingly little in order to correctly identify a language. Interestingly, this is the case even if they are not proficient in that language or when the text snippet is quite short. This observation leads to a class of algorithms based on character (or even byte) n-grams, as opposed to more linguistically refined syntactic or semantic methods.

- A popular tool called *textcat* [3] constructs a ranking of the most frequent character n-grams for each language during the training phase and proclaims this ranking the *language model*. For classification, a ranking is constructed for the input document in the same fashion and is compared against each available language model. The closest model (in terms of ranking distances, see [3] for details) wins and is returned as the identified language.
- Another character n-gram approach pioneered by [2] computes likelihood of generating the observed character sequence explicitly, through use of higher order Markov models. Let S be a sequence which consists of n characters $(s_1, \ldots, s_n)$. Then the probability this sequence was generated by Markov model L of order k is given by

$$P(S \mid L) = p(s_1, \ldots, s_k \mid L) \prod_{i=k}^{n} p(s_{i+1} \mid s_{i-k+1} \ldots s_i, L),$$

  where the first factor is the initial state distribution and the conditional probability describes transitions. Training the language model consists of estimating these transition probabilities. Again, winner is the language with the best likelihood of generating the input text. It is observed that using character trigrams, i.e. Markov models of order 2, already gives optimal results and increasing the model order therefore cannot affect performace much. For a comparison of character trigrams to "common words", see [4].
- A related approach makes use of Shannon's information theory and compares language entropies [5]. Here Markov models are also estimated based on training data. In contrast to [2], all models of orders $0, \ldots, k$ are used and their relationship explicitly modeled. This allows the algorithm to fall back to lower order models in case of insufficient data through mechanism called *escape probabilities*. Decision function views input as a stream of characters and in accordance with information theory tries to predict the next character in the stream. Success of these predictions is measured by cross-entropy and the model with the lowest cross-entropy after having processed the whole stream wins. Because of its ties to information theory and language compression, this technique is sometimes called the *compression* technique.

Apart from individual algorithms, research into language recognition has also identified key factors which directly influence performance:

- **Size of training data**. Methods are evaluated based on how quickly their models converge, given differing sizes of training corpora. Note that more is not necessarily better here, as there is a risk of *overtraining* or *overfitting* the training data.
- **Size of input text**. Methods can be distinguished by how much text they need to be given in order to reliably identify its language. Amount of text can be roughly divided into *small* (a phrase, less than 30 characters or up to 5 words), *large* (a paragraph, more than 300 characters or 50 words) and *medium* (a sentence, in between).

3 Proposed Method

Motivation for Change

Summing up the previously mentioned articles, there are several reasons behind the success of language modelling via character n-grams:

- **Fast convergence**. Very small training corpora (in the order of hundreds of kilobytes of text) are required to learn the models. See e.g. [6] for a study on speed of model convergence for character bigrams and trigrams.
- **Robust**. As long as the overall letter distribution in input document follows that of training examples, problematic language phenomena such as neologisms (words newly introduced into the language), spelling errors, rare inflections or unknown words are handled gracefully.
- **Domain independent**. In [2] this approach was applied to a domain as distant as that of genetic sequence identification. Another often highlighted feature is that character n-gram methods do not require tokenization of the input, making them also suitable for Asian languages where tokenization is an interesting challenge in itself.

We implemented, for some time used and then evaluated an algorithm based on the compression technique [5]. We estimated all i-gram distributions for $i = 0, \ldots, n$ and then combined them through an *Expectation Maximization* (EM) smoothing algorithm on held-out data. We were interested in detecting nine European languages: French (fr), English (en), Italian (it), Spanish (es), Slovakian (sk), Czech (cs), Slovenian (sl) and Polish (pl). Although this method worked almost perfect on our test data, a number of real-world issues soon became apparent when applied to the task of Web page classification. The main problem was not insufficient accuracy of classification as such, but rather a shift in the formulation of the language identification problem:

- **No *unknown language* option**. All methods listed above based on entropy, Markov processes or n-gram profiles return the nearest, best-fitting

language. They assume that a) the set of languages is complete, known and trained for beforehand and b) that the input text is in exactly one of them. While the first assumption can be dismissed as our own decision, the latter is unrealistic for the Web.

- **Multiple languages**. In an also rather frequent scenario, there are parts of the input document which are in different languages. This may stem from a page's logical structuring (menu, text body, copyright notices) but also from the nature of the text body itself. This moves the document away from any one model and the language models become mixed in undesired ways. As a result, the document may even be identified as a completely unrelated language not present in the input text at all. In our experience, multilingual documents somehow often ended up being marked as Slovenian.
- **Close languages (same language family)**. As seen above, our language set includes Slovenian, Slovakian, Czech and Polish, which are all Slavic languages with considerable grammatical as well as lexical overlap. This is exacerbated by the fact that real texts on the Web often come in deaccented version, so that the trained models are unable to even theoretically take advantage of otherwise telling national characters (*ř* for Czech, *ľ* for Slovak etc.).

As a special case of the second point, there are many pages where letter distribution is heavily skewed by repetition of certain words or phrases. This includes discussion lists with `In reply to:` fields and so on. This problem does not come up in well-behaved corpora, but quickly becomes a nuisance when dealing with the Web.

To address the first two issues, we tried augmenting our implementation of the n-gram algorithm. We looked for a minimum threshold for each language that a document score has to exceed in order to be identified as that particular language, even if its score is the best available. Note that for language detection, document length is not an issue, as all models are evaluated on the same number of n-grams and the score numbers are thus directly comparable. For the fixed threshold to work, however, the scores need to be normalized to negate the effect of varying document lengths, as adding even one n-gram changes the order of magnitude of the probability scores.

Although we tried setting this threshold automatically, based on held-out training data, the results were not satisfactory. It appears that the per-character cross-entropies are dependent on the contents of text n-grams in a way that prohibits direct absolute comparison against any fixed threshold. In other words, it proved impossible to find a threshold that would allow us to tell "this best fitting language is in fact an error". We also tried learning a special *unknown language* model from a hotch-potch of documents in various random languages. This worked better and solved the Slovenian classification problem, but seems rather ad-hoc and theoretically unfounded.

To avoid headache of further complicating an already complex algorithm, we set out to try a different approach.

Dictionary Method

In [2], dictionary methods (i.e. methods based on words rather than characters) are discussed and dismissed, based on their best known representative, "common words", being too restrictive and only applicable to longer texts.

Going through the list of n-gram advantages, the benefits of broad domain independence, no required tokenization and fast model convergence will indeed have to go. Since our goal is to tell apart European (Latin character based) natural languages, the first two are not really a concern. The last one, small amount of training examples required, was perhaps an asset back when these methods were developed. In the present day, with Web as Corpus [7] projects and NLP advancements, fast indexing and retrieval techniques, the amount of available data is no longer a critical issue. The same cannot be said for runtime performance, as the same reason why there are many documents requires us to process them at increased speed. For these reasons we decided to revisit the dictionary method.

We take a qualitatively different approach to constructing the dictionary language models. Rather than looking for words that are common in a given language (called *function* or *stop* words), we note which words are *specific* for a language, or rather, *how specific* they are. The foundation of our algorithm is a relevance mapping

$$rel(word, language) : W \times L \mapsto \mathbb{R}$$

where W is a set of all words present in the training data and L the set of considered languages. We call the real-valued score of word $w \in W$ in a language $l \in L$ its *relevance*. In other words, the mapping is not binary as in the case of the "common words" approach, but rather a soft grading of words. Positive relevance hints at the word being indicative of the language, relevance around zero naturally corresponds to "no correlation" and negative values to "this word is indicative of *absence* of the language". We will call these *positive*, *near-zero* and *negative* evidence, respectively.

Naturally, the relevance mapping is constructed automatically from labeled training data. In contrast to character n-gram models, the convergence is much slower and significantly larger training corpura are required. We estimate the word relevance using reasoning detailed in [8]. Their idea, although developed for classifying documents into topics, can also be applied to our problem of language identification. Below we give a short overview of the main assumptions and steps behind derivation of the final relevance formula; for a more thorough discussion on various aspects, please see the original article [8].

We start by noting frequencies of words $w_1, w_2, \ldots, w_N$ within each language corpus and compare them to frequencies in a general, *background* corpus. In this context, a corpus C is simply a collection of D documents, $C = (d_1, d_2, \ldots, d_D)$. For each language $lang$, we have a corpus C_{lang} of documents only in that language, plus one general background corpus which represents a collection of documents of background language $lang_0$. This background language is ideally completely language neutral, or more realistically represents the distribution of

all languages on the Web. To approximate $lang_0$, we consider the union of all language corpora to be the background corpus, $C_0 = \bigcup C_{lang}$. The *uncorrected observed frequency* of word w in language $lang$ is then

$$\bar{g}_{lang}(w) = \frac{TF(w, C_{lang})}{\#(C_{lang})}, \tag{1}$$

with $\#(C)$ being the total number of words in corpus C and TF the number of occurences of a word in a corpus, and

$$g_0(w) = \frac{TF(w, C_0)}{\#(C_0)} \tag{2}$$

for the background language.

From the assumption of languages being modelled as Bernoulli (word unigram) sources, the probability that a document d that contains f_i instances of word w_i was produced by language $lang$ is given by the multinomial

$$P(d \mid lang) = \binom{f_0 + f_1 + \cdots + f_N}{f_0, f_1, \ldots, f_N} \prod_{i=0}^{N} g_{lang}(w_i)^{f_i}. \tag{3}$$

To avoid singularities for zero frequencies, the *Jelinek-Mercer smoothing* correction is introduced

$$g_{lang}(w) = \alpha g_0(w) + (1 - \alpha)\bar{g}_{lang}(w) \tag{4}$$

for some small value of α, such as 0.1.

After substituting (4) into (3), we compute logarithm of probability ratio that a document was emitted by $lang$ rather the background language $lang_0$ by

$$log \frac{P(d \mid lang)}{P(d \mid lang_0)} = \sum_{i=0}^{N} f_i log \frac{\alpha g_0(w_i) + (1 - \alpha) g_{lang}(w_i)}{g_0(w_i)} \tag{5}$$

An interesting observation the authors present is that negative and near-zero evidence contributes very little to classification accuracy. In fact, according to [8], accuracy actually *improves* when near-zero and negative evidence is purposefully omitted. Translated to our language identification problem, we only pay attention to words that are highly indicative of the given language, disregarding near-zero and negative evidence entries. This has the pleasant side-effect of keeping the models reasonably sized, despite there being virtually tens of millions of possible words in each language relevance mapping. With this simplification and some minor mathematical tricks (see [8]) the formula becomes an elegant and manageable

$$\sum_{g_L(w_i) \ll g_{lang}(w_i)} f_i \cdot rel(w_i, lang), \tag{6}$$

where $rel(w, lang) = log(g_{lang}(w)) - log(g_0(w))$ is our desired relevance of word w in language $lang$. Put in words, the relevance of a word measures the

orders of magnitude by which it is more frequent in the specific language corpus compared to the background corpus. This a surprisingly simple relationship, given we started only from the assumption of word independence (Bernoulli model). Another way to look at the formula is to realize that f_i corresponds to Term Frequency (TF) and $rel(w_i, lang)$ to a kind of Inverse Document Frequency (IDF) component, linking this result to the general framework of TF-IDF classifiers. Obviously this is a sharp divergence from the idea of identifying languages by the most "common words".

With all pieces in place, how do we go on choosing which languages a sequence of words belongs to? According to the above derivation, we simply iterate over words that are distinctive of each language and sum their relevancies. We may compare this value to a threshold to immediately see if there was enough evidence to proclaim the document d as belonging to language *lang*. But to abstract from document length, we first divide this sum by the length of the document in words, that is, we take average of the individual word relevancies. The final decision function which identifies document d as coming from language *lang* is then

$$score(d, lang) = \frac{\sum_{g_L(w_i) \ll g_{lang}(w_i)} f_i \cdot rel(w_i, lang)}{\sum f_i} \geq t_{lang}. \tag{7}$$

The threshold t_{lang} is found, for each language separately, by optimizing an objective function on held-out data. One candidate for objective function is the F_1 measure, the generalized formula of which is

$$F_\beta = (1 + \beta^2) \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}.$$

F_1 measure is popular in Information Retrieval and defines an equal trade-off between precision and recall. Other objective functions are possible, depending on the desired application of language identification. If the cost of not identifying the right language (*false negative*) is higher than cost of erroneously identifying an unwanted language (*false positive*), higher preference should be given to recall (e.g. via the F_2 measure) and vice versa. This effectively lowers (resp. raises) the estimated language threshold.

Contrary to results obtained from using thresholds for character n-gram method, detecting unknown language works quite reliably (see Evaluation section). Because some words may be indicative of several languages (such as the previously mentioned lexical intersection of Slavic languages), more than one language may be recognized, too.

Practical Considerations

As noted earlier, runtime performance of classification is important. Interpreting equation (7), the algorithm consists of tokenizing input text, averaging token relevancies and comparing this sum to a precomputed threshold. This can be

Table 1: Overall data and model statistics.

Language code	Dump size [GB]	No. unique sentences [k]	No. test documents small	medium	large	Dictionary model size [words]
cs	4.8	2,926	814	907	814	551,126
de	39.4	27,010	461	916	762	944,450
en	208.3	74,926	448	980	998	548,649
es	18.9	10,848	520	891	742	318,423
fr	39.8	18,048	483	852	765	373,432
it	26.0	11,529	469	836	727	378,817
pl	18.0	10,157	286	878	784	799,180
sk	3.3	1,769	275	916	768	474,003
sl	2.8	1,472	249	916	795	288,442

done extremely fast, using any of the many commonly available data structures which map strings into numbers.

As for memory considerations, the mapping that needs to be stored is in fact very sparse, consisting of only those words which are distinctive for a language. In fact, the size of each language model when stored as *Patricia trie* [9] was in the tens of megabytes, which is comparable to size of our character pentagram models. This is not surprising as character pentagrams already come close in length to whole words.

We solved the practical question of obtaining large and representative language corpora by using Wikipedia dumps [10]. As research into Web corpora [7] rapidly progresses, it can be expected that assembling large text collections will become less and less of a problem in the future. It must be kept in mind however that common NLP techniques like stemming or lemmatization may not be applied, as these are dependent on language—the very thing we don't know and want to determine in the first place.

Evaluation

To evaluate our algorithm, we trained it on Wikipedia dumps [10] of the nine target languages. As a reminder, these are French (*fr*), English (*en*), Italian (*it*), Spanish (*es*), Slovakian (*sk*), Czech (*cs*), Slovenian (*sl*) and Polish (*pl*). To avoid overfitting the training data, we discarded duplicate sentences and only used each sentence once in our corpus. Sentences with non-Latin (mostly Asian and Arabic) characters were also ignored. Some data statistics can be seen in Table 1, where the number of unique sentences corresponds to the size of training data. In the same table we also give final model statistics. We put aside three thousand sentences of differing lengths for each language, to be used as test data. These were divided into *small*, *medium* and *large* sub-corpora (with texts of 2–5 words, 6–50 words and over 50 words, respectively), so that each subcorpus contained exactly 1,000 texts. We manually checked the test corpora and estimated that the ratio of erroneously labeled examples is

Table 2: Evaluation on test data.

language code	n-gram method, text size: small	medium	large	dictionary method, text size: small	medium	large
cs	81.9/75.2	96.8/96.0	100.0/100.0	64.9/84.0	85.2/96.9	97.8/99.6
pl	84.1/67.6	97.4/96.5	97.9/97.2	82.9/90.2	95.5/97.0	96.9/97.5
sk	81.6/77.7	97.7/96.9	99.3/99.0	57.8/82.9	71.4/96.6	87.6/96.7
sl	89.3/79.7	97.8/97.2	99.3/99.2	68.6/88.2	91.9/97.2	98.8/99.0
it	81.9/58.4	98.7/96.4	99.9/99,8	78.6/88.1	95.8/98.0	99.4/99.7
fr	80.1/52.9	98.3/97.3	99.8/99.6	82.7/88.7	98.4/99.0	99.5/99.6
de	85.2/73.6	98.8/98.1	99.0/98.4	85.7/91.8	98.2/99.6	98.8/99.2
es	81.5/61.6	99.0/98.1	100.0/99.9	73.2/86.4	94.3/98.9	99.3/99.8
en	81.4/51.7	99.4/98.2	99.8/99.1	86.1/91.6	99.2/99.7	99.8/99.4

Precision/recall on test data, in percent.

– about 10% for medium length documents (mostly municipality and proper name enumerations),
– about 20% for long texts (same reason, plus many texts are in fact English phrases such as song or book titles)
– and as much as 50–70% for the short texts.

Short texts are especially bad because they concentrate "sentences" consisting of formulas, location entries, article headings with a person's name and lifetime and so on. All of these often refer to foreign institutions and have no connection to the language of the main article. Final sizes of test corpora after removing these problematic texts are given in Table 1.

Classification results are summarised in Table 2, which also includes results of our implementation of the cross-entropy based character n-gram algorithm described earlier, on the same data. Recall is measured as the ratio of true positives to all available positives (including false negatives), precision is the number of true positives divided by the number of all positives returned (including false positives). Note that this gives more room for precision errors to the dictionary method, which can return multiple false positives for each document, unlike the n-gram method that returns at most one incorrect language per document.

Table 3: Evaluation on pruned test data.

language code	n-gram method, text size: small	medium	large	dictionary method, text size: small	medium	large
cs	93.5/92.4	98.7/98.7	100.0/100.0	73.9/99.8	86.7/100.0	98.1/100.0
en	93.1/67.3	99.7/98.6	100.0/100.0	98.4/100.0	99.7/100.0	100.0/100.0

Precision/recall on pruned test data, in percent.

Inspection of results reveals that the errors closely follow the data problems described above. On one hand this is vexing because it prohibits more exact evaluation. On the other hand it shows that despite the considerable amount of noise in training data (which obviously shares the same problems as the test data) and in face of contradictory information, the classifiers are able to generalize. However, we'd like to stress the fact that our goal here is not to discuss the absolute numbers, but rather to juxtapose and compare two language identification methods on the same dataset.

To confirm our hypothesis of poor data quality, we manually checked labels of all Czech and English test examples. The labeling error was about 1% for medium and large texts and about 40% for texts of small length. We expect this error to be similar for the seven remaining languages, too. We re-ran language identification experiments on the two manually pruned corpora, with results summarised in Table 3. Many short Czech documents are classified as both Czech and Slovak by the dictionary method, resulting in lower precision but still excellent recall.

We conclude that the numbers are sufficiently high (in fact, after discounting the test data noise, nearly optimal) for both algorithms. The main difference and actually the reason why we developed our dictionary method in the first place is the added value of being able to return a set of languages as identificaton result, including the elusive empty set.

4 Segmenting for Language

There is one item on our language identification wish-list that hasn't been covered yet: correct classification of documents that contain blocks from different languages. While our character n-gram method based on cross entropy returns a random language in this case, dictionary method returns an unknown language. Both results are wrong. A practical extension to either algorithm would ideally allow us to locate and identify all compact single-language blocks.

To our knowledge, the only attempt at language segmentation was made in [5]. The authors consider all possible combinations of language change at each character in the input text and measure the resulting entropy on such text blocks. Although they report brilliant results of 99.5 % accuracy on *character level*, the method misses the mark in terms of speed by several orders of magnitude. Even with advanced dynamic programming optimizations, it took tens of seconds to segment a text [5].

Here we describe and evaluate an algorithm that segments input text into monolingual blocks.

Let $S_{lang}(d) = (score(w_1, lang), \ldots, score(w_n, lang))$ be a sequence of individual unit scores (word relevancies or n-gram probabilities) for the n units in document d. We can view this sequence as a real-valued signal and use signal processing to smooth the signal, removing local extrema,

$$(smoothed)_i = fnc_{SIZE}(score(w_{i-SIZE}), \ldots, score(w_{i+SIZE})), \qquad (8)$$

for any language *lang*. We use *median* with sliding window size $SIZE = 2$ as the smoothing function while noting that there is a direct connection between the smoothing window size and robustness to short extra-lingual segments in text. These manifest themselves as sharp local valleys and correspond to proper nouns, typing errors and other text anomalies. Although strictly speaking they really are different from the surrounding text, our task is to identify coherent language blocks that are meaningful on discourse rather than token level.

Once we have smoothed signals for all available languages, we identify local minima in them. This gives us a first estimate of potential segment boundaries. The proposed segment boundaries are not final though—many of them correspond to local minima in between two segments of the same language. We remerge these back into a single segment. Note that in this way we prohibit having two consecutive segments of the same language, but we may still arrive at segments with no language assigned to them. It is also possible to have a segment with more than one language. This means the text may have been written in either and is indistinguishable. This often occurs with shorter *cs*/*sk* passages and reflects real ambiguity of input.

Complexity of the whole procedure is linear in the number of words and languages, $O(|d| \times |L|)$.

Evaluation

To evaluate language segmentation, we constructed an artificial corpus. The corpus contains 1,000 documents, each one of them being a concatenation of 1 to 4 segments in different languages. The numbers were picked to somewhat mimick situation on the Web, with 4 languages in a single document as an extreme case. Language segments are pooled randomly from a collection of medium-length texts in that language (6 to 50 words).

We give this concatenation to our segmentation algorithm, with signal scores based on word relevancies, and mark down languages predicted for each token. This per-token evaluation records *success* each time a token was assigned precisely the one language that was expected, and $failure$ otherwise. Accuracy is then computed as $\#success/(\#success + \#failure)$. Note that assigning multiple languages or no language at all to a token always equals an error.

The algorithm misclassified 1,420 out of possible 49,943 words. This corresponds to 97,16% accuracy. In 603 cases, boundary was missed by one word, which is still an acceptable error for our purposes. Discounting these off-by-one boundary errors, accuracy climbs to 98,34%. Closer inspection of the 817 misses left shows that some of them come from English collocations like *grand theft auto* which are embedded inside non-English text segments and regrettably misclassified as English by the algorithm. The real accuracy is therefore probably slightly higher, depending on mode of application.

Although these results are lower than those reported in [5], the algorithm enjoys conceptual clarity and impressive runtime performance.

5 Conclusion

The article's main contribution is revisiting and re-evaluation of some of the assumptions made 15 years ago, when the domain of automated language identification was being shaped. It proposes a straightforward, fully automated method which learns a decision function from training data. The decision function is based on word relevancies and addresses some aching problems of popular character n-gram based methods, while retaining character n-gram's excellent accuracy and actually improving runtime efficiency. Another important benefit of using words instead of character n-grams is that the system is more open to human introspection, more predictable in ways of interpreting its results ("looking inside the box") or selectively changing its behaviour—something of considerable value in real systems.

A general segmentation algorithm is described which is based on the notion of language signal strength within the input document. The algorithm is evaluated to behave acceptably using word relevancies and solves the problem of language identification in multilingual documents.

References

1. Ingle, N.: A Language Identification Table. Technical Translation International (1980)
2. Dunning, T.: Statistical Identification of Language (1994)
3. Cavnar, W.B., Trenkle, J.M.: N-gram-based text categorization. In: Ann Arbor MI. (1994) 161–175
4. Grefenstette, G.: Comparing two language identification schemes. In: Proceedings of the 3rd International Conference on the Statistical Analysis of Textual Data (JADT'95). (1995)
5. Teahan, W.: Text classification and segmentation using minimum cross-entropy. In: Proceeding of RIAO-00, 6th International Conference "Recherche d'Information Assistee par Ordinateur", Paris, FR. (2000) 943–961
6. Souter, C., Churcher, G., Hayes, J., Hughes, J., Johnson, S.: Natural Language Identification Using Corpus-Based Models. Hermes Journal of Linguistics **13** (1994) 183–203
7. Kilgarriff, A.: Web as corpus. In: Proceedings of Corpus Linguistics 2001. (2001) 342–344
8. Kornai, A. et al.: Classifying the Hungarian Web. In: Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1, Association for Computational Linguistics Morristown, NJ, USA (2003) 203–210
9. Morrison, D.: PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric. Journal of the ACM (JACM) **15**(4) (1968) 514–534
10. Wikimedia Foundation Project: Wikipedia Static HTML Dumps. (http://static.wikipedia.org/) June 2008 Edition.