

# Language Recognition for Mono- and Multi-lingual Documents

## Abstract

In this paper we describe language recognition algorithms for mono- and multi-lingual documents that are based on mixed-order n-grams, Markov chains, maximum likelihood, and dynamic programming. We compare the monolingual algorithm to those suggested by other researchers. This comparison suggests that this algorithm significantly outperforms commonly used language recognition algorithms. We then describe the multilingual algorithm, which allows for segmenting a multilingual document into single language chunks and identifying the languages of those chunks.

## Keywords

language recognition, n-gram model, corpora, dynamic programming

## 1. Introduction

Language recognition algorithms are an essential component of many natural language processing systems. For example, a system that automatically translates web-documents must first determine the language of that document. Moreover, in multilingual documents, the system must determine monolingual segments and the language of each segment. Language recognition algorithms are also essential in the *development* of many natural language processing systems. For example, very large corpora are often required in the development of machine translation systems and a language identification system can be combined with a web spider to automatically collect such corpora.

These recognition algorithms are not just concerned with the language of electronic texts, but also with how the characters of that language are encoded. This specification is important because the characters of a given language may be represented by several distinct encoding methods or code sets. For example, Japanese text may be in JIS-X-0208-1990 or JIS-X-0212-1990, among others, and Chinese text might be in GB-2312-80 or Big Five. To ease exposition in this paper, we use the term *language* to mean a language/code set pair.

In this paper we describe a language recognition algorithm for multilingual documents based on mixed order n-grams, Markov chains, and maximum likelihood. We present the results of a study comparing the performance of this algorithm to that of three other language recognition algorithms. These results suggest that the proposed algorithm significantly outperforms commonly used algorithms for this task. We then describe a language recognition algorithm for multilingual documents and examine the performance of the algorithm.

## 1. Recognition Algorithm for Monolingual Documents

Many previous approaches to language identification have relied on lists of words that commonly appear in the languages of interest (Ingle 76). That is, for every language of interest we construct a list containing the most frequent words in that language. This, certainly, is a common sense approach to the problem of language identification. *The*, *of*, and *and* are common words in English, and *el*, *de*, *que*, and *y* are common words in Spanish. If you are trying to identify the language of a text and you see a lot of *the's of's* and *and's* in the text a good guess would be that the text is in English. Similarly, if you see *el*, *que*, and *de*, a good guess would be Spanish. While this approach is simple, intuitive, and performs adequately in many cases, it does have several drawbacks. First, while the approach works well for identifying texts of moderate length, its performance significantly degrades as the text to be

identified gets smaller. This will be seen in §3 where we perform an evaluation of several algorithms. Second, since it is a word-based approach it crucially depends on the ability to segment the text into words. This is easy if the task is to recognize languages like English and Spanish where words are bordered by spaces, but is significantly more challenging if we include languages that do not use spaces as word delimiters such as Chinese, Japanese, Korean, and Thai.

The approach we propose here is not dependent on identifying the words of a text. Rather, it is a statistically based approach that learns to distinguish between languages by building an n-gram model.<sup>1</sup> It is similar to the work of Cavnar and Trenkle (Cavnar & Trenkle 94), among

others, in that it makes use of mixed-order n-grams.<sup>1</sup> In contrast to the Cavnar and Trenkle algorithm, which makes use of an ad hoc rank-order distance measure, the algorithm we propose here is theoretically well motivated based on a maximum likelihood approach.

The algorithm has two main phases: training and classification.

## 2.1 Training

The algorithm is a variable length n-gram approach. We start with an empty common n-gram pool, *CNP*, which we will fill with n-grams from each training text— $m_1$  unigrams,  $m_2$  bigrams, ...  $m_n$  n-grams. For every language we extract the n-grams ( $n=1 \dots N$ ) from the training text. (Our current implementation uses an  $N$  of 4.) We maintain separate lists for each n-gram length. That is, we have separate lists for unigrams, bigrams, and so on. Next, we add a portion of these n-grams to *CNP* (the common pool); first we add unigrams, then bigrams, and so on.

We select the unigrams to add to *CNP* as follows. For every unigram,  $a_1$  observed in the training data, we compute the following training weight:<sup>2</sup>

$$W(a_1) = -p(a_1) \log p(a_1).$$

We then order the list of all unigrams in descending order on this value and place the top  $m_1$  unigrams in the common pool, *CNP*.

Then we use the following recursive procedure. Suppose we have already picked up n-grams of length 1 ... (k-1). For every k-gram ( $a_1 a_2 \dots a_k$ ) we compute a training weight. If the (k-1)-gram ( $a_2 \dots a_k$ ) has not been included in *CNP* the training weight value is<sup>3</sup>

$$W(a_1 a_2 \dots a_k) = -p(a_1 a_2 \dots a_k) \log p(a_k | a_1 a_2 \dots a_{k-1})$$

If ( $a_2 \dots a_k$ ) is in *CNP*, then we use a slightly different formula:

$$W(a_1 a_2 \dots a_k) = -p(a_1 a_2 \dots a_k) (\log p(a_k | a_1 a_2 \dots a_{k-1}) - \log p(a_k | a_2 a_3 \dots a_{k-1})).$$

Next we sort the k-gram list in descending order on this weight and place the  $m_k$  top k-grams in *CNP*. The training weights describe the reduction of the cross entropy between training data and the model if we include the k-gram ( $a_1 a_2 \dots a_k$ ) in our pool. We repeat the procedure for  $k=2 \dots N$ . For the comparative experiments described in §3 we used an  $m_1$  of 170, an  $m_2$  of 200, an  $m_3$  of 400, and an  $m_4$  of 230. The number of unigrams chosen was enough to include all the unigrams for all languages in the training data. These numbers have been determined through experimentation.

At the end of this process *CNP*, the common n-gram pool, contains the union of the selected n-grams for all  $L$  languages in our training set. For every n-gram ( $n=1 \dots N$ ) in *CNP* we compute a primary recognition weight  $L$ -dimension vector, *PRW*, as follows. Each  $i$ -th component of the vector,  $PRW_i$  is associated with the  $i$ -th language, and contains a recognition weight. For unigrams, the recognition weight is

1. The n-grams of a text are all the character sequences of length  $n$  contained in that text. For example, *unmarked helicopters*, contains 20 unigrams ( $u, n, m, a, r, \dots$ ), 19 bigrams ( $un, nm, ma, \dots$ ), 18 trigrams ( $unm, nma, mar, \dots$ ) and so on.

1. See also, Churcher, Hayes, Johnson, and Souter (Churcher et al. 94) and Dunning (Dunning 94).

2.  $p(a)$  represents the probability of  $a$ .

3.  $p(a_n | a_1 \dots a_{n-1})$  represents the conditional probability. That is, the probability of character  $a_n$  given that the immediately previous characters were  $a_1 \dots a_{n-1}$ . For example,  $p('e' | 'th')$  is the probability that the third character is 'e' given that the first two characters are 'th'.

$$PRW_i(a_1) = -\log p_i(a_1)$$

For all other n-grams, the recognition weight is:

$$PRW_i(a_1a_2\dots a_{n-1}) = -\log p_i(a_n | a_1a_2\dots a_{n-1})$$

That is, the anti-log of the conditional probability of  $a_n$  given  $a_1a_2\dots a_{n-1}$  in the training set associated with the  $i$ -th language. If some n-gram in  $CNP$  has not been encountered in some language training data, its recognition weight for this language is defined to be some maximum value  $MAX$  (in our experiments,  $MAX = 20$ ). At the end of this process we have a primary recognition weight vector,  $PRW(a_1a_2\dots a_k)$  for every n-gram,  $a_1a_2\dots a_k$ , in our n-gram pool,  $CNP$ . Now we define the recognition weight L-dimension vector of any N-gram (encountered or not encountered in our training data) as

$$RW(a_1a_2\dots a_N) =$$

In addition to developing the  $CNP$  containing recognition weight vectors, we also compute a weight average and weight dispersion for each language (these are used in the verification step of the recognition phase described in the following section). This is done as follows. We divide the training text of a given language,  $i$ , into  $K$  500 byte segments ( $x_1x_2\dots x_{500}$ ). For every segment  $k$  we compute:

The weight average for language  $i$  is defined as

The dispersion for language  $i$  is defined as:

## 2.2 The Recognition Phase

The recognition process consists of two steps. First, in the classification step, we tentatively classify the text to be recognized as being in one of the languages specified in the training phase. Next, in a verification phase, we determine how well the text to be recognized fits the proposed language. If the fit is not good enough, we classify the language of the text as unknown.

Classification Step: Let  $x_0x_2\dots x_{S-1}$  be the byte sequence of the text to be classified. We define a result recognition weight vector as:

The recognition result (the proposed language of the text) is

(That is, the text is classified being in the language,  $i^*$ , associated with the result recognition weight vector component with the lowest value.)

Verification Step: If<sup>1</sup>

Then we recognize the text as belonging to language  $i^*$ . If this does not hold then the text is classified as “unknown”.

## 1. Evaluation of Recognizer

We evaluated this language recognizer by comparing its performance to that of three other language recognition algorithms: a simple common words algorithm, a common words algorithm that uses a notion of distance, and the Cavnar and Trenkle n-gram algorithm. These algorithms are described below. They share the following characteristics:

- There are two distinct phases: a training phase where models for each language are developed, and a recognition phase where texts are categorized based on those models.
- Our implementations of each algorithm use a 50k training text for each language.
- The text is tokenized and punctuation and digits are discarded.

### 3.1 Common Word Algorithm

Training phase: For every language we develop a model as follows:

- Determine the frequency of the words in the training text.
- Sort the words in order of descending frequency.

The language model consists of an unordered set of the top N words in this sorted list (in our implementation  $N = 1000$ )<sup>1</sup>

Recognition phase: Let  $t$  be the text to be classified. Determine the score for every language model as follows. For every word in the model, count how many times that word occurred in text  $t$ . The language model score is the sum of all these word counts. The model with the highest score wins. That is, the text,  $t$ , is classified as the language of the highest scored model. This simple algorithm was successfully used in a previous version of our corpus collection spider.

### 3.2 The Minimal Distance Approach

Training phase: For every language develop a model as follows:

- Determine the frequency of the words in the training text.
- Sort the words in order of descending frequency.
- The language model consists of an ordered list of the top N words in the sorted list (in our implementation  $N = 1000$ )

Recognition phase: Let  $t$  be the text to be classified. Develop a profile of  $t$ ,  $P(t)$ , by

- Determining the frequency of the words in  $t$ .
- The profile is the list of these words sorted in order of descending frequency.

Compute the distance between each language model and this profile as follows.

- Let the word distance be defined as the difference between the ordinal position of that word in the model and the ordinal position of that word in the profile. For example, suppose that the word *the* is the most frequent word in a language model. That is, it is the first word in the language model list. However, in a profile it is the fifth word in the list. Then the distance would be 4.
- The distance of the language model from the profile is the sum of all the distances of the words in the model.

The text,  $t$ , is classified as the language specified by the model that has the shortest distance.

### 3.3 The Cavnar and Trenkle Algorithm

Training phase: We develop a model for each language<sup>2</sup> as follows:

---

1. The constant, VER\_THR, is used to rule out documents, which are not in the pre-trained languages.

1. In an evaluation of this algorithm (and the following minimal distance one) comparing the performance for different values of N we observed a 25% error reduction when N was 1,000 compared to when N was 100 under the experimental conditions described in §3.4

- Determine the frequency of each n-gram ( $n = 1 \dots 5$ ) in the tokenized text.
  - Sort the n-grams in order of descending frequency.
- The model for a given language consists of the ordered list of the top N n-grams. In our implementation we have used an N of 1000.<sup>1</sup>

Recognition phase: We first develop a profile of the text to be identified by

- Determining the frequency of each n-gram ( $n = 1 \dots 5$ ) in the tokenized text.
- Sort the n-grams in order of descending frequency.

Next we compute a distance from each of the language models described above to this profile. We define the n-gram distance as the difference of the n-gram's ordinal position in the language model and in the profile. For example, if the n-gram *th* occurred in the 14th position in the model and in the 21st position in the profile the distance would be 7. If a model n-gram does not occur in the profile, we set the distance to a large constant.<sup>2</sup> The distance between a language model and the profile is the sum the n-gram distances for the n-grams in the language model. The language with the shortest distance is the classification of the document.

### 3.4 The Method

We compared these four algorithms by using the following method. The training data consisted of 50k samples from the following 34 languages:

Afrikaans	Italian
Albanian	Japanese
Arabic	Korean
Bulgarian	Latin
Chinese	Lithuanian
Croatian	Malay
Czech	Norwegian
Danish	Persian
Dutch	Polish
English	Portuguese
Estonian	Russian
French	Serbian
German	Slovak
Greek	Spanish
Haitian Creole	Swedish
Hawaiian	Thai
Icelandic	Turkish

We did not use Chinese, Japanese, Korean, and Thai to test the two word-based algorithms due to the difficulty of automatically identifying words in these languages.

We evaluated each algorithm under five conditions that varied as to the size of the text to be identified: 1k, 500, 100, 50, and 20 byte samples. The samples were distinct from the training text and were drawn from 200k texts from each language. 200 samples for each language were used to evaluate the algorithms in the 1k condition, 400 samples for the 500 byte condition, 2000 samples for the 100 byte condition, 4000 samples for the 50 byte condition, and 10,000 samples for the 20 byte condition.

### 3.5 Results

The comparison results for the four algorithms (The Common Word (CW) algorithm, the Distance Common Word (DCW) algorithm, the Cavnar & Trenkle (C&T) algorithm, and the algo-

2. Cavnar and Trenkle (1994) state that the algorithm only handles ASCII texts. However, we have successfully used their algorithm with texts in a variety of codesets including mixed-byte codesets.

1. Cavnar and Trenkle (1994) used smaller values for N ( $N = 100, 200, 300, 400$ ). We have found a significant error reduction by using larger values.

2. Cavnar and Trenkle do not specify what this value should be other than to say that it should be set to "some maximum ... value". In our implementation of this algorithm we set this maximum value to 10,000. This produced significantly better results (an 8-fold error reduction) than setting it to the number of n-grams in the profile + 1, or to the number of n-grams in the profile minus the position of the n-gram in the model, or to lower constant values (e.g., 1,000).

rithm we propose (LZC) are shown in the following chart and table.

Error Rate of Algorithms at Different Sample Sizes

Percent Error Rate of Algorithms at Different Sample Sizes

Sample Size	Algorithms			
	LZC	C&T	DCW	CW
1000	0.27	3.03	1.83	1.80
500	0.52	3.10	2.48	2.71
100	2.02	4.23	8.20	10.10
50	4.01	6.86	18.83	22.59
20	11.92	16.56	45.97	48.55

As this chart shows, the performance of all these algorithms is good when the text to be classified is relatively large (500–1,000 bytes). However, even under this condition it should be noted that the algorithm proposed here has less than one sixth the error rate of the other algorithms. A significant percentage of the error rate in the 1,000 and 500 byte conditions is due to misidentification of the test samples from Haitian Creole and its lexifier, French.<sup>1</sup> This is probably due to the fact that creoles borrow much of their vocabularies from their respective lexifiers (Romaine 88). It is interesting to note that all the algorithms performed relatively well on Afrikaans and Dutch, although many creolists regard Afrikaans as a semi-creole of Dutch.

As is clearly shown in the chart, the performance of the word-based approaches degrades significantly as the size of the text to be classified becomes smaller. These algorithms mis-recognize around 20% of the 50 byte test samples and over 40% of the 20 byte samples.

#### 4. Recognition Algorithm for Multilingual Documents

While this algorithm has important uses in many applications it does have some limitations in the current form. For example, suppose we have a multi-language machine translation system that converts web pages into an “English-only” form that is displayed to users. If the original web pages are in a single language then the task is simply to identify the language of the page using the current algorithm “as is” and then applying the appropriate machine translation. However, if the web pages are multilingual then the process is more complex. In this case we need to segment the page into single-language chunks, identify the language of each chunk, and then perform the correct translation. This task is illustrated in the following example.<sup>2</sup>

His example is essentially this (taken from Chechen): if in a language geminate occur only inter-syllabically, and the non-geminate version appears in those intersyllabic positions plus word-initially and word-finally, then if we know the average number of syllables per word, we can make a prediction of the relative frequency of the single and geminate versions of a consonant. If our prediction does not match the reality, then we can infer there is something that remains to be accounted for.

“Le chiffre absolu de la fréquence réelle d'un phonème n'a qu'une importance accessoire. Seul le rapport entre ce chiffre et le chiffre de fréquence attendu théoriquement possède une valeur véritable. (284) Le calcul des probabilités théoriques n'est pas toujours aussi simple que dans les exemples ci-dessus. Mais on ne doit pas se laisser rebuter par les difficultés d'un tel calcul, car c'est seulement par comparaison avec les chiffres de fréquence possible obtenus au moyen de ces calculs que les chiffres de fréquence effective acquièrent une valeur, en montrant si un phonème, dans la langue en question, est beaucoup ou peu utilisé. (285).”

This is a powerful notion that remains to be fully explored. What Troubetzkoy (and others since) have seen is that a study of frequency can often be tantamount to a search for lurking generalizations.

1. Without these two languages the results for the 1,000 and 500 byte tests are as follows:

2. from <http://humanities.uchicago.edu/faculty/goldsmith/Royaumont98/InfoTheory.html>.

As described above, the system needs to segment this document, identify the languages of the chunks (in this case English and French) and translate the French segment into English. Fortunately, an important advantage of the approach presented above is that it can be generalized for multilingual documents. This generalization is accomplished by adding a dynamic programming algorithm based on a simple Markov model of multilingual documents. This algorithm is described in the following section.

#### 4.1 Segmentation algorithm

The algorithm consists of two steps:

- a recognition step where we apply a dynamic programming algorithm to find the segmentation maximizing the likelihood of the total character sequence of the document being processed,
- a verification step, which determines how well every segment fits the language assigned to it.

##### Recognition step

The segmentation algorithm is based on the following Markov model of a multilingual document. The model has one state per language,  $1 \leq i \leq L$  plus one additional 0-state accounting for segments that are not in any of pre-trained languages, in particular, such segments may be in no language at all (e.g., a table of numbers).

If a system is in a state  $i$  it generates characters in the  $i$ -th language depending on the left context according to the  $n$ -gram model described above in §2. Switching from language  $i_1$  to language  $i_2$  is defined by transition probabilities  $p(i_2/i_1)$  and probability distribution of segment length  $r$   $p(r)$ ,  $r_{\min} \leq r \leq r_{\max}$ , ranging from minimal segment length  $r_{\min}$  to maximal  $r_{\max}$ . The algorithm presented below finds the segments and segment languages assuring the maximum likelihood of the observed text given the Markov model. Using antilog, the criterion to minimize is as follows:

$$Q(\{x_s, 0 \leq s < S\}, \{s_k, i_k, 0 \leq k \leq K\}) =$$

where

$$TSW(i_0, i_1, s_0, s_1) =$$

if  $i_1$  is not 0. Otherwise:<sup>1</sup>

$$TSW(i_0, i_1, s_0, s_1) = \text{JUNK\_THR}^{*(s_1-s_0)} -$$

The dynamic programming algorithm finding optimal values of  $\{s_k, i_k, 0 < k \leq K\}$  consists in the following iterative step for

$$\text{Ind}(i_1, s_2) = (i_0^*, s_1^*) =$$

---

1. The constant JUNK\_THR (“junk threshold”) is used to rule out junk chunks of the document. By “junk chunks” we mean chunks that are not in any of the pre-trained languages.

$$TSW(i_0^*, i_1^*, s_0, s_1), \quad F(i_1, s_1) = F(i_0^*, s_0^*) +$$

with initial values:

$$F(i, 0) = 0, \quad 0 \leq i \leq L,$$

$$F(i, s) = \text{INFINITY}, \quad s: 0 < s < r_{min}$$

After the algorithm has finished with the recursive steps, we find the language of the last segment in the optimal set of segments:

The value  $i^*$  together with information stored in array  $Ind(i, s)$  will allow us to get all segments with corresponding language labels from the optimal set of segments. During this process, the optimal number of segments is automatically determined.

### Verification step

The verification step is executed exactly as in the monolingual algorithm regarding every segment as a separate monolingual document.

### 4.2 Evaluation

We evaluated the performance of the segmentation algorithm by comparing its performance on segmenting documents that contain multilingual text. The test data contained text in the following languages:

Afrikaans	Japanese
Albanian	Korean
Arabic	Latin
Chinese	Lithuanian
Croatian	Malay
Czech	Norwegian
Danish	Persian
Dutch	Portuguese
English	Russian
Estonian	Serbian
French	Slovak
German	Spanish
Greek	Thai
Italian	Turkish

The test documents were created by concatenating randomly selected segments from documents in the languages listed above. In this way we created five documents; each document contained 100 segments having the same length (20, 50, 100, 500, or 1000 bytes). We compared the results produced by the algorithm to the known structure of the document. We considered the algorithm generated segment correct if the difference between that segment's boundaries and the boundaries of the actual segment were less than five bytes and if the language label was the same. The following table shows the segment error rate as a factor of segment byte size.

#### Percent Error Rate of Multilingual Algorithm at Different Segment Sizes

Segment Size	% Error Rate
1,000	0
500	0
100	2

50	2
20	8

As this chart shows, this algorithm works extremely well for moderate-sized segments and performs adequately for short (approximately 3 word) segments. This can be seen as even more impressive results if we take into account the difficulty in processing these generated documents compared to processing naturally occurring documents. In naturally occurring documents the different language segments are necessarily separated by punctuation and whitespaces (spaces, tabs, etc.). These separators were not used in creating the constructed documents.

## 5. Conclusions

In this paper we have described language recognition algorithms for mono- and multi-lingual documents. These algorithms are based on mixed order n-grams, Markov chains, maximum likelihood, and dynamic programming. As described in §3, we have conducted a study comparing the performance of the monolingual algorithm to three other language recognition algorithms across 34 languages. The results of that study suggest that the proposed algorithm significantly outperforms the others including the Cavnar and Trenkle algorithm, one of the most popular language recognition algorithms. Moreover, the proposed algorithm has an additional benefit. The other algorithms find the closest language that matches the text to be classified, but give no guarantee that this language is actually the language of the text. The proposed algorithm has a separate verification step that assures, with a controllable degree of certainty, that the text to be classified is actually *in* the closest language. Because of these advantages, the algorithm can be successfully used in a variety of applications.

In §4 we described a language recognition algorithm for multilingual documents, and presented the results of an experimental study which showed that the performance of this algorithm was extremely good and suggests that the algorithm has practical value. This algorithm offers a new reliable tool for natural language engineering, which allows for more sophisticated processing of documents including web pages.

## References

- (Cavnar & Trenkle 94) William Cavnar and John Trenkle. *N-gram-based text categorization*. Symposium on Document Analysis and Information Retrieval, 1994.
- (Churcher, et al. 94) Gavin Churcher, Judith Hayes, Stephen Johnson, and Clive Souter, *Bigram and trigram models for language identification and character recognition*. Proceedings of the 1994 AISB Workshop on Computational Linguistics for Speech and Handwriting Recognition. 1994.
- (Dunning 94) Ted Dunning, *Statistical identification of language*. Computing Research Laboratory Technical Report MCCS-94-273. New Mexico State University. 1994.
- (Ingle 76) N. C. Ingle, *A language identification table*. The incorporated linguist 15(4).98-101 1976.
- (Romaine 88) Suzanne Romaine, *Pidgin and Creole Languages*, Longman, 1988.