Natural Language Identification for OCR Applications

Bachelor thesis

Freie Universität Berlin, Department of Mathematics and Computer Science

June, 8th 2010

Author	Do, Hoang Viet
1st Advisor	Prof. Dr. Raúl Rojas
2nd Advisor	Dr. Marco Block-Berlitz

Table of Contents

Introduction

The writing system is one of the oldest and most used methods to exchange information. It is unfortunate when one is excluded due to visually handicaps. To compensate such disadvantages, the FU Berlin has started the project “*Mikrooptik*”[1]. An intelligent reading-glass, containing an active camera, captures documents in range and reads them using a synthesizer. To ensure a correct pronunciation of each language, the application must identify the document’s natural language.

The present thesis evaluates the currently available approaches of natural language identification and introduces a new feature extraction method “NoSpace” to enhance the precision when dealing with OCR applications. In the evaluation, NoSpace could increase the precision by around 10% to 15% depending on the approach. All approaches are implemented and evaluated using MiLiD [Minimal Language Identification (System)], a new framework for language identification.

Starting with a general introduction of language identification including its states of the art, the first part discusses the 3 most important approaches and the new feature extraction method for OCR applications “NoSpace”. The second part depicts the implementation of MiLiD and its essential components and classes. The approaches are evaluated in the last part. In contrast to previous works, all approaches use the same feature extraction methods in order to ensure a more comparable result.

[1] http://www.inf.fu-berlin.de/groups/ag-ki/Drittmittelprojekte/Mikrooptik.html accessed on May, 1st 2010

I Theoretical Basics

The first part consists of three chapters.

In the first chapter, the basic terms used in the matter of language identification, the basic idea and the state of the art are described. In the second chapter, the available feature types are introduced and compared. In chapter 3, the approaches are discussed and in chapter 4, the new counting method is described.

1 Basic idea of language identification

In this chapter, basic terms used in the matter of language identification and the basic idea of language identification are described as well as the state of the art.

1.1 Basic Terms

In the present thesis, the term **language identification (abbr. LID)** describes the (automatic) process of determining the natural language of written texts[2]. It can be subdivided into the *training phase* and the *classification phase*.

In the **training phase**, the system extracts language features from the given training corpus to generate language. A corpus is a (large) collection of electronically stored written texts. Since the languages are given, this training phase can be classified as **supervised learning** in the matter of machine learning.

In the **classification phase**, the system classifies the language of the given document:

1. A model is generated for the given document.
2. Afterwards, the similarity between the document's model and each language models is computed.
3. The language model, which is most likely, is chosen as the language used in the document.

Ultimately, the LID problem is a classification problem with the languages as the classes and the unknown document's language as the query. The questions are "which feature types and similarity measure is most suitable"?

Figure 1 illustrates the general architecture of a language classifier.

[2] In case of spoken text, the process is called **speech recognition.**

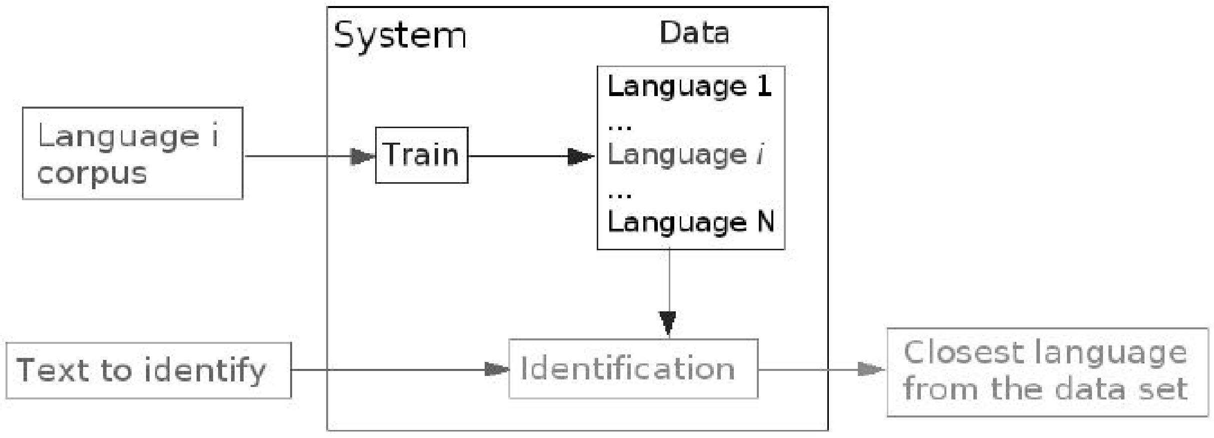


Figure 1: General architecture of a language identifier; cf. (Padró Cirera and Padró Cirera 2004, 2).

1.2 State of the art

Different approaches have achieved a precision of 100% using sufficient data, c.f. (Grothe, et al. 2008, 984, Artemenko and Shramko 2005, 119 ff., Prager 1999, 5). For example, Prager has reported, that his system "Linguini" requires 500 bytes to identify 13 languages[3] with a precision of 100%, cf. (Prager 1999). Using 200 bytes, the precision is still about 99%! Dunning reported a precision of about 92% using his approach to identify test strings of 20 bytes. But his classifier only recognized English and Spain (Dunning 1994, 16). As Padró has proven, the precision depends on i. a., the number of language, the similarity of the languages, cf. (Padró Cirera and Padró Cirera 2004, 4 ff.). Therefore 500 bytes is a parameter depended value. E.g. by decreasing the number of languages, the data required might decrease.

Silvia et al. proposed an approach for identifying language variant, cf. (Ferreira da Silva und Pereira Lopes 2006). The team reported that several lines of a document are required to achieve sufficient precision.

2 Language identification approaches

As mentioned in the previous chapter, the LID problem is a classification problem. In this chapter, the available feature types are discussed.

2.1 Using words as feature type

A system, using words as feature type, is looking for frequent words and short words such as conjunctions in the training process. In the classification process, the system is looking for these words in the document. The language model, causing the most hits, is chosen.

[3] Catalan, Danish, Dutch, English, Finnish, French, German, Icelandic, Italian, Norwegian, Portuguese, Spanish, and Swedish

In the present thesis, word based approaches won't be discussed in more details. Since it has been proven by previous works, that word based approaches are not suitable for very short texts since the amount of information available is lower in comparison to N-gram based approaches (Artemenko and Shramko 2005, 22 - 26).

In addition, the precision of word based approaches is greatly reduced when processing languages with high morphology such as German since $ein \neq eins \neq eine \neq einer \neq eines \neq \cdots$.

Furthermore, it is highly sensitive to spelling errors. These errors occur in optical character recognition systems such as the Mikrooptik Project.

Interested readers are referred to Artemenko' & Shramko' master thesis (Artemenko and Shramko 2005, 16 - 21).

2.2 Using N-grams as feature type

An **N-gram** is a contiguous sequence of characters of the length N (Cavnar and Trenkle 1994, 2). 1-grams are called uni-gram, 2-grams are called bi-grams, 3-grams are called tri-grams, and so on (quad, pent, hex, and so on). E.g. the word "text" contains the following N-grams:

1. **Uni-grams:** t, e, x
2. **Bi-grams:** te, ex, xt
3. **Tri-grams:** tex, ext
4. **Quad-grams:** $text$

Using high N-grams leads to higher more memory since there are more different letter combinations. E.g. in German there is around 26 different 1-Grams, whereas the amount of 4-grams is hardly countable).

In some approaches, the query document is tokenized and the beginning and the ending of each word is marked with a special character such as the underscore. In addition, this character is used to expand the N-grams to match a certain length (Cavnar and Trenkle 1994, 2-3, Ferreira da Silva und Pereira Lopes 2006, 2). E.g. the word "text":

1. **Uni-grams:** t, e, x
2. **Bi-grams:** $\_t, te, ex, xt, t\_$
3. **Tri-grams:** $\_te, tex, ext, xt\_, t\_\_$
4. **Quad-grams:** $\_tex, text, ext\_, xt\_\_, t\_\_\_$

In general, a blank-padded string of length k has $k + 1$-bi-grams, tri-grams, and so on (Cavnar and Trenkle 1994, 3).

Performing a tokenization, more language information can be gathered. On the other hand, this makes the particular approaches unsuitable for languages, that are hardly tokenized such East Asian languages such as Chinese and Japanese.

3 N-gram based approaches

In this chapter, 3 N-gram based approaches are discussed:

1. Ad-hoc Ranking (Cavnar and Trenkle 1994)
2. Vector Space Model (Damashek 1995, Prager 1999)
3. Bayes Decision Rules and Markov Chain (Dunning 1994)

In general, other approaches are mainly enhancement, combination or customization of these 3 approaches.

3.1 Ad hoc Ranking

In this approach, the language models and document models are ordered lists of N-grams (N=1, 2, 3, 4, 5). The N-grams are sorted by their absolute frequencies within the language (training-corpus). The list sizes are limited to 300. This value has been chosen based on their observations. Cavnar and Trenkle recommend the customizing of this value as it fits the application. The very highest ranking N-grams are mostly uni-grams and simply reflect the distribution of the alphabet. Whereas starting around 300, N-grams are specific to the subject. The lists are named profiles. The language profiles are named category profiles. (Cavnar and Trenkle 1994, 4,5).

As discussed in chapter 2.2 *Using words as feature type* on page 4. Cavnar and Trenkle have performed a tokenization step and added an underscore as a marking character for the beginning and ending of each word.

Language profiles are computed in the training phase. In the classification phase, the document profile is computed and the distances between the document model and each language model are computed based on the rank of each N-gram of the two profile. Figure 2 illustrates the calculation.

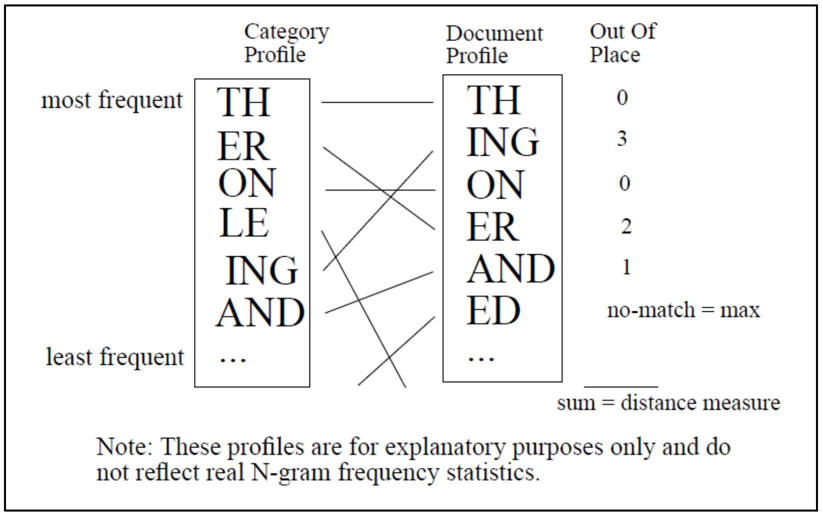


Figure 2 Calculating the Out-Of-Place measure between two profiles (Cavnar and Trenkle 1994, 6)

For each N-gram, the "out-of-place"-value is calculated. This is the absolute difference in rank of an N-gram in the document profile and its counterpart in the language profile. Ultimately, N-grams, having the same rank in the document profile and language profile, have the out of place value 0.

The "total distance" is the sum of all out-of-place values. If an N-gram of the document profile cannot be found in the language profile, a certain "max-value" is added to the total distance. This max-value is not specified in the paper. Grothe et al. have used a dynamic max-value, which results in a higher precision in comparison the precision obtained with a static chosen max-value (Grothe, et al. 2008, 984). In the end, the language profile, causing the smallest total distance, is chosen.

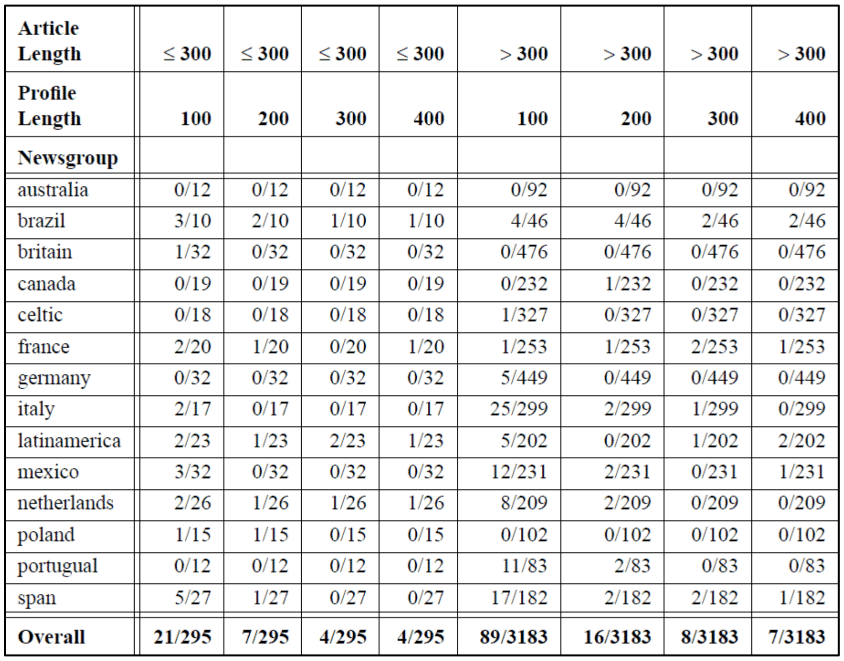
The present system MiLiD supports dynamic max-value. Depending on the parameters the language profile size or document profile size is scaled and used as the max-value. The best precision could be achieved with a max-value equal to twice the size of the language profile size.

3.1.1 Test Result

Cavnar & Trenkle have tested their classifier on 3713 language samples collected from 14 $soc.culture$-newsgroups. All samples were encoded using standard ASCII, with few typographical conventions to handle such things as diacritical markings. 7 Languages were used: French (France), Italian (Italy), Spanish (Latin-American, Mexico and Spain), Dutch (Netherlands), Polish (Poland), Portuguese (Brazil and Portugal), and English (Australia, Britain, Canada, and Celtic). Inappropriate samples, e.g. mixed languages, were removed. The resulting test set consisted of 3478 usable articles. (Cavnar and Trenkle 1994, 6 - 7)

The samples were subdivided into 2 groups "≤ 300" and "> 300" and classified using 4 different profile length including 100, 200, 300 and 400. (Cavnar and Trenkle 1994, 7 - 9)

Table 1 Ratio of Incorrect Classifications to Total Possible Classifications (Cavnar and Trenkle 1994, 9)



Article Length	≤ 300	≤ 300	≤ 300	≤ 300	> 300	> 300	> 300	> 300
Profile Length	**100**	**200**	**300**	**400**	**100**	**200**	**300**	**400**
Newsgroup								
australia	0/12	0/12	0/12	0/12	0/92	0/92	0/92	0/92
brazil	3/10	2/10	1/10	1/10	4/46	4/46	2/46	2/46
britain	1/32	0/32	0/32	0/32	0/476	0/476	0/476	0/476
canada	0/19	0/19	0/19	0/19	0/232	1/232	0/232	0/232
celtic	0/18	0/18	0/18	0/18	1/327	0/327	0/327	0/327
france	2/20	1/20	0/20	1/20	1/253	1/253	2/253	1/253
germany	0/32	0/32	0/32	0/32	5/449	0/449	0/449	0/449
italy	2/17	0/17	0/17	0/17	25/299	2/299	1/299	0/299
latinamerica	2/23	1/23	2/23	1/23	5/202	0/202	1/202	2/202
mexico	3/32	0/32	0/32	0/32	12/231	2/231	0/231	1/231
netherlands	2/26	1/26	1/26	1/26	8/209	2/209	0/209	0/209
poland	1/15	1/15	0/15	0/15	0/102	0/102	0/102	0/102
portugual	0/12	0/12	0/12	0/12	11/83	2/83	0/83	0/83
span	5/27	1/27	0/27	0/27	17/182	2/182	2/182	1/182
Overall	**21/295**	**7/295**	**4/295**	**4/295**	**89/3183**	**16/3183**	**8/3183**	**7/3183**

The result is shown in Table 1. The profile length seems to improve the precisions slightly. E.g. Span, Italy

The advantage of this approach is the very low consumption of memory. Let n be the profile length. Since we can only save n N-grams, the maximal memory consumption is

$$N * n$$

Since n is independent of the training size, we actually have $O(1)$.

A disadvantage of this approach lays in the use of a tokenization step. Languages such as Chinese, Japanese, and so are hardly tokenized.

3.2 Vector Space Model

In this approach, language models and document models are vectors of a multi-dimension space. The similarity is measured through the cosine distance of the document vectors and the language vectors. Given the language model a and the document model x, the cosine distance is defined as

$$\cos\alpha = \frac{\vec{a} * \vec{x}}{|\vec{a}| * |\vec{x}|} = \frac{\sum(a_i\, x_i)}{\sqrt{(\sum a_i)^2 (\sum x_i)^2}} \in [0,1]$$

The language model, causing the smallest cosine value, indicates the used language, cf. (Prager 1999, 6).

Prager has used the absolute frequency of the features weight by the inverse document frequency idf, a term well known from the field of document retrieval. Given n_i as the number of documents, in which the N-gram i occurs despite its frequency in this document, the idf is defined as

$$idf = \frac{1}{n_i}$$

Prager has tested more aggressive and less aggressive weighting and reported the present weighting as the best performing, cf. (Prager 1999, 3).

Given m_i as the absolute frequency of the N-gram i in the training corpus, we can apply filtering using an arbitrary k as follow

$$m_i < n_i * k$$

i will be excluded if the present condition is true. Prager has evaluated the parameter k in the range of 1 to 10 and reported values in range of 3 to 5 as best performing. (Prager 1999, 3)

3.2.1 Test Result

Prager has trained his classifier Linguini to identify 13 Western European languages using different training size for each language, ranging from 19,759 to 99,743 bytes.

For the evaluation, a collection of about 100 Kbytes was gathered from the internet and inspected manually for "any obvious inclusions" of foreign language text strings. The texts were split into chunks respecting word-boundaries. This caused the chunk sizes to be <u>at least</u> 20, 50, 100, 200, 500 and 1000 then precisely 20, 50 and so on, cf. (Prager 1999, 3).

Prager has used stationary N-grams with N ranging between 2 and 5. In addition, he evaluated his approach using short words, common words and a combination of N-grams, short words and common words. In case a string required double-byte encoding, Linguini compared byte sequences instead of N-grams (character sequences), c.f. (Prager 1999, 4 - 5).

Table 2 Performance of Linguini depends on feature type and chunk size. (Prager 1999, 5)

	Chunk Size					
Feature-set	20	50	100	200	500	1000
2-grams	68.8	86.2	93.5	97.7	98.8	100.0
3-grams	79.5	93.0	97.7	99.3	100.0	100.0
4-grams	83.6	94.3	98.2	99.6	99.9	100.0
5-grams	81.4	93.1	97.8	99.4	99.9	99.9
Words	69.7	86.6	94.7	98.1	99.9	100.0
SWords	61.3	81.5	92.1	97.1	99.6	100.0
SW+3grams	83.8	94.9	98.5	99.7	100.0	100.0
SW+4grams	84.9	95.3	98.6	99.7	99.9	100.0
W+4grams	85.4	95.6	98.7	99.7	99.9	100.0

The result is illustrated in Table 2. Using combined feature type performed best. Quad-grams performed best with text less than 500 bytes when using N-grams only.

For languages with different dialects, multiple language vectors were generated using the same language label. This increased the performance. For instance, Norwegian has two major dialects (Bokmal and Nynork). Using merely one profile with an equal share of each dialect results in lower precision since Danish is more likely to Bokmal than the 50-50 profile. Figure 3 illustrates the situation

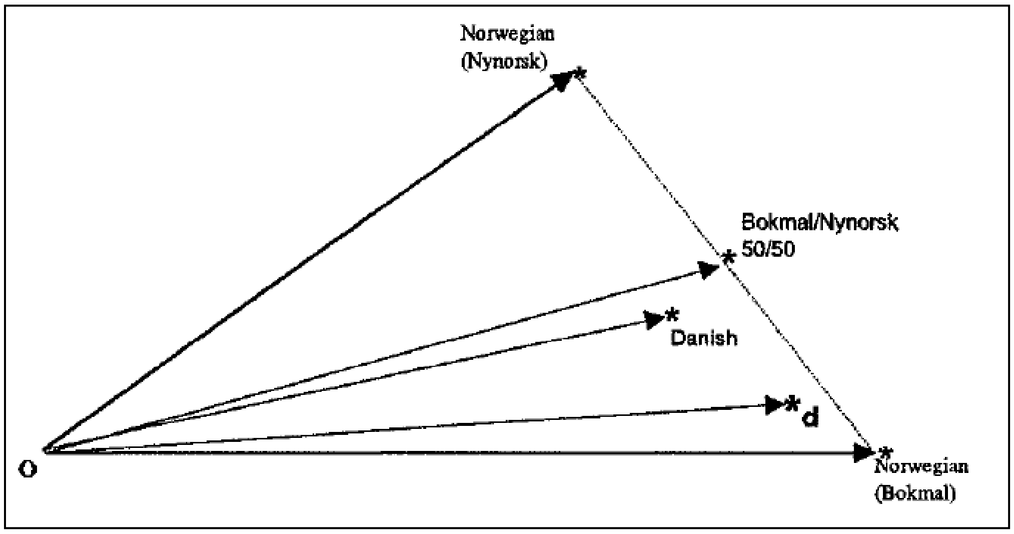


Figure 3 Illustration of the misclassification of document d when using one profile for the 2 major dialects of Norwegian (Prager 1999, 7).

3.3 Bayesian Decision Rules and Markov Model

In this approach, the insights of the probability theory are applied to identify languages. Markov Chain is used to model the language information and the Bayes Decision Rules are used to identify the languages.

3.3.1 Markov Model

A **Markov Model** is a random process in which the probability distribution of the next state depends solely on the current state. More formally, a Markov Model defines a random variable whose values are sequences of states. The probability of a particular variable S is defined as

$$p(S) = p(s_1 s_2 \dots s_N) = p(s_1) \prod_{i=2}^{N} p(s_i | s_{i-1})$$

The probability of S is completely characterized by the initial state distribution $p(s_1)$ and the transition probabilities $p(s_i|s_{i-1})$. The **initial state distribution** $p(s_1)$ describes the probability of state s_1. The **transition probabilities** $p(s_i|s_{i-1})$ describe the probabilities for the transition from the previous state s_{i-1} to the current state s_i. (Dunning 1994, 6)

Applied to the LID, S is a string defined as a character sequence $(s_1 s_2 \dots s_n)$, where each character is a state. The transition probabilities define the probabilities, that the next character of s_{i-1} is s_i. E.g. the string “text” is defined as

$$p(text) = p(s_t s_e s_x s_t) = p(s_t) * p(s_e|s_t) * p(s_x|s_e) * p(s_t|s_x)$$

Dunning has broadened the definition to capture more language features: bi-grams, tri-grams, etc. By using the last k states as the label for the current state, the transition probabilities for the relabeled model become

$$p(s_{i+1} \dots s_{i+k}|s_i \dots s_{i+k-1}) = p(s_{i+k}|s_i \dots s_{i+k-1})$$

Thus the probability for a particular string S is

$$p(S) = p(s_1 \dots s_N) = p(s_1 \dots k) * \prod_{i=2}^{N} p(s_{i+k}|s_i \dots s_{i+k-1})$$

These models are known as **Markov models of order k**. Such model is completely described by the initial state distributions $p(s_1 \dots s_k)$ and the transition probabilities $p(s_{i+k}|s_i \dots s_{i+k-1})$. Although low order Markov model does not capture very much structure of a language in comparison to higher order Markov model, it is not necessarily better in the purpose of language identification since the amount of training data required to accurately estimate the parameters is roughly exponential in the order of the model (Dunning 1994, 7).

3.3.2 Bayes Decision Rules

Given the task of deciding which of the given phenomena $A, B, C, \dots$ has caused a particular observation X, we can minimize our probability of error by computing which most likely to has caused the observation. Applied to the purpose of language identification, the phenomena are the available languages and the particular observation is the document.

Since we know, that the used language affects the observation, we can apply the Bayes’ Theorem (A is used without lack in generality) (Dunning 1994, 8):

$$p(A|X) * p(X) = p(X|A) * p(A)$$

The a priori probability $p(X)$ is set to 1 since it is the actual observation and it is constant for each language

$$p(A|X) = p(X|A) * p(A)$$

The a priori probability $p(A)$ is truly unknown. It describes the probability of the language A. Good results can often be achieved by assuming that they are equal (Dunning 1994, 9). It is often set to $1/(\text{number of languages})$ used in the training stage (Artemenko and Shramko 2005, 37).

Using the concept of Markov model, the probability of observing string S, given a particular Markov model A, is (Dunning 1994, 9)

$$p(S|A) = p(s_1 \dots s_k|A) \prod_{i=k+1}^{N} p(s_i|s_{i-k} \dots s_{i-1}|A)$$

$p(s_1 \dots s_k|A)$ describes the probability of the initial string $s_1 \dots s_k$ of length k of the language A.
$p(s_i|s_{i-k} \dots s_{i-1}|A)$ describes the probabilities, that the characters s_i are the next characters of the substrings $s_{(i-k)} \dots s_{i-1}$ of length k.

Dunning simplified this formula by grouping all likely terms

$$p(S|A) = \prod_{w_1 \dots w_{k+1} \in S} p(w_{k+1}|w_1 \dots w_k|A)\, T(w_1 \dots w_{k+1}, S)$$

$T(w_1 \dots w_{k+1}, S)$ quantifies the occurrences of the $k+1$-grams in the string S. This product is calculated for all $k+1$-grams founded in the string S. (Dunning 1994, 9).

To avoid numerical underflow, the logarithm is used instead

$$p(S\,|\,A) = \sum_{w_1 \dots w_{k+1} \in S} T(w_1 \dots w_{k+1}, S) \log p(w_{k+1}|w_1 \dots w_k|A)$$

The language model, which caused the largest result, is picked as the used languages (**assuming equal a priori probabilities!**). (Dunning 1994, 9)

As described in the formula, the language models are matrices storing the transition probabilities while the document model is a list storing the N-grams and its frequencies.

The transition probabilities can be estimated using the ratio of counts derived from the training. In case, the training data of the language A is given as a string T_A

$$p(w_{k+1}|w_1 \dots w_k|A) = \frac{T(w_1 \dots w_{k+1}, T_A)}{T(w_1 \dots w_k, T_A)}$$

$T(w_1 \dots w_{k+1}, T_A)$ describes the absolute frequency of the $k+1$-grams counted from the training data. $T(w_1 \dots w_k, T_A)$ describes the same for k-grams. The ratio describes the transition probability. This method is known as the **maximum likelihood estimator** (Dunning 1994, 10).

$k+1$-grams, which do not appear in the training data, will cause $p(S|A) = 0$ in the classification process and leads to misclassification. To avoid this problem, an approximation is used instead

$$\hat{p} = \frac{T(w_1 \ldots w_{k+1}, T_A) + 1}{T(w_1 \ldots w_k, T_A) + m}, \qquad m = |A|$$

m is the size of the alphabet of the language A. This is called **Laplace's sample correction**. In this thesis, the derivation of this formula from above is skipped. Interested readers are referred to (Dunning 1994, 10 - 12).

3.3.3 Test Result

Dunning has collected a bilingual (English/Spanish) test corpus and training corpus. The training corpus consists of 50 training texts: 10 training documents for each length of 1000, 2000, 5000, 10.000 and 50.000. The test corpus consists of 600 test texts: 100 test samples for each length of 10, 20, 50, 100, 200 and 500 bytes. The test samples were examined manually. Any strings which were not "clearly" (Dunning 1994, 13) Spanish or English were replaced with "newly generated" (Dunning 1994, 13) strings. In addition, Spanish samples containing English language were excluded and vice versa (Dunning 1994, 13 - 14).

In the evaluation, Markov models of order 0 to 4 have been used. The result is shown in Figure 4. An accuracy of about 92% has been achieved with test documents of only 20 bytes using a classifier trained with 50K training data. The precision were improved up to about 99.9% when classifying test documents of 500 bytes using model order of 3 and 4. For shorter documents or less training data bigrams (mode order 1) were most suitable. While with larger documents only the unigrams (model order 0) had relative high error rates (Dunning 1994, 16).

Dunning also applied and evaluated his approach for genetic sequences. Interested readers are referred to (Dunning 1994, 14 - 15).

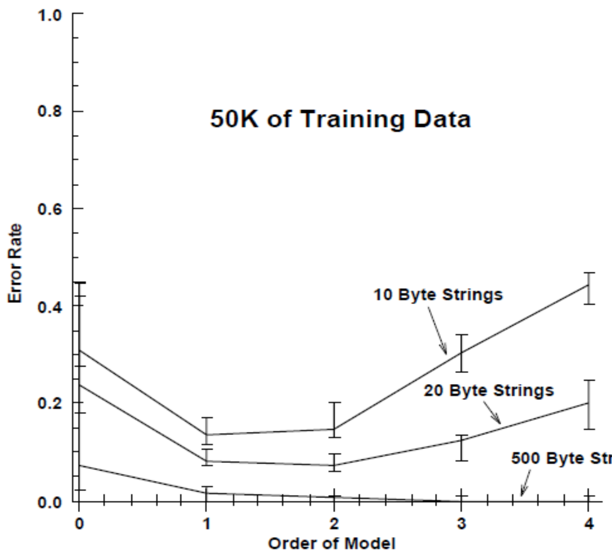


Figure 4 Test result of a classifier trained with 50K bytes training data (Dunning 1994, 29).

4 Language Identification for OCR Applications

In the traditional LID, the system classifies documents with no (or at least minimal) errors such as misspell. But these errors are common and in the matter of OCR they are quite frequent. To improve the precision, the feature extraction method "NoSpace" is introduced in this chapter.

4.1 The basic idea

Assuming the system does not have the original document, it is hard to say whether a character is an error, produced by the OCR or not. In addition, assuming the "best approaches" of OCR has been used, it might be impossible for the system to say even if the original document is available.

So, the data has to be taken as it is and the system musts decide whether how fault-prone and important the particular information is. For high fault-prone information, the particular information shall be dropped in a way that does not disturb other information.

The characters of a document can be classified in 4 categories:

1. Alphabetic characters including accents
2. Numbers. E.g. 1 2 3 4
3. Whitespace (including line break, tab and so on).
4. Special characters such as punctuation (all other characters). E.g. , . ; !

Each category is discussed in the following chapters.

4.2 Alphabetic characters

The first character type "alphabetic characters including accents" is the main information source of the document. It is essential and must remain unchanged even with errors.

4.3 Numbers

Numbers does not contain much information about a language. But just removing them is quite risky. For example: l (lowercase of L) is recognized as 1 resulting in a word like "mies" (German, means bad) instead of "mi1es". The original word was "miles". By removing the number an English word becomes a German, which is actually more troublesome.

The present system removed all numbers in the training corpus. But during the classification, the numbers are kept. In this way, information captured by the lower (valid) N-grams remains valid, whereas the erroneous information provided by (higher) N-grams are dropped since they cannot be found in the language model, some default value (e.g. max-value by N-Gram Profile of Cavnar & Trenkle) are used instead. E.g. mi1es

	Valid	**Dropped information**
Unigrams	m,i,e,s	1
Bigrams	mi, es	i1, 1e
Trigrams		mi1, 1es
Quadgrams		mi1e, i1es
Pentagrams		mi1es

6 N-grams remains valid and no invalid N-grams have been generated. If the number is just dropped, the result will be:

	Valid	**Invalid**
Unigrams	m,i,e,s	
Bigrams	mi, es	ie
Trigrams		mie, ies
Quadgrams		mies

6 N-grams remains valid but 4 invalid N-grams have been generated.

4.4 Whitespaces

Whitespaces are primary used to mark the word boundaries. Therefore they seem to be essential information. On the other hand, the OCR algorithm can easily be confused by optical distortion and produces a lot of false whitespaces. For example: “word” can be recognized as “$w\ o\ \ r\ d$” or “$wo\ rd$” or something worst.

The present system removed whitespaces in the testing corpus. In this way, the additional whitespace are removed but the word boundaries are destroyed. To solve this problem, the whitespaces in the training corpus are removed as well, resulting the training and testing documents become a single “word”. This information provided by whitespaces is dropped completely!

NoSpace does not perform tokenization since it drops all whitespaces. In contrast, it captures words sequences as information. E.g. quadgrams of “HelloWorld”: Hell, ello, lloW, loWo, and so on.

4.5 Special characters

Special characters are neither part of the actual language nor part of any words. Therefore they can be removed without any consequence.

II Developing MiLiD

In the second part, the core components and classes of “MiLiD” are discussed, as well as the tools and API used and the restrictions and ISO applied.

5 Settings

In this chapter, the tools and API used and the restrictions and ISO applied are discussed.

5.1 Requirements Elicitation

As mentioned in the introduction, the present thesis is initiated by the *Mikrooptik* Project. It is an OCR-Application on an embedded device (with limited resources). As the pre-processing of a real-time application, the system has to optimize its memory consumption and CPU usages while still keeping a high precision.

Additionally, the present system has to be fault-tolerate since OCR systems tend to generate spelling errors.

5.2 Tools and API

Microsoft Visual Studio Team Systems 2008 has been used as the development environment on a PC running Windows Seven. The code documentation has been automatically generated using Doxygen[4] 1.6.3. “`Refactor! For C++`“[5] was used for Refactoring.

C++ has been supervised as the programming language based on the project’ programming language.

No additional API has been used for the actual language classifier except for `_wfopen_s`[6]. This function provides the same functionality as `fopen` from the Standard C Library except for the Unicode support. Its invocation is almost the same as `fopen` and can be replaced by a more STL compatible user-defined function.

5.3 Restrictions

Althought it is possible to use multiple character encodings, the present system supports only Little Endian. The BOM (byte order mark), which indicates the encoding Little Endian or Big Endian, is read. An exception is thrown when Little Endian has not been used or the BOM is missing.

All training datas and testing datas are encoded in Little Endian. The system always processes characters as of type `wchar_t`.

5.4 ISO 639-3

The present system uses a 3 character string to present language identifiers, along the lines of the ISO-639-3 standard.

[4] http://www.doxygen.org accessed on March, 13th 2010
[5] http://www.devexpress.com/Products/Visual_Studio_Add-in/RefactorCPP/ accessed on March, 14th 2010
[6] http://msdn.microsoft.com/en-us/library/z5hh6ee9.aspx accessed on March, 13th 2010

However, the language identifiers are not validated whether it is confirmed to the ISO-693-3. But all language identifier will be trimmed to a 3 character string whenever necessary, e.g. saving and loading the language model. This might lead to false-misclassification in the case of multiple language identifiers having the same beginning.

6 The Common Interface `CClassifier`

All classes are defined in the namespace `dph`.

The classifiers of the system have the common interface `CClassifier`. The interface itself is a template class with one abstract model data type T. It defines 4 abstract methods and 1 constant method:

1. `Train`,
2. `LoadClassifier`,
3. `SaveClassifier`,
4. `ClassifyModel, and`
5. `GetResultMap.`

The 5 methods are briefly discussed in the next chapters.

6.1 Train

Train is specified as

```
virtual int Train(const std::map<std::wstring, std::set<std::wstring>>
                  &corpus, size_t limit =-1) =0;
```

The first parameter `corpus` is a standad STL map datatype. The language identifiers are assosiated with the respective set of training files. The amount of characters used for training can be optionaly limited by the second parameters.

If the training corpus contains any not-Little-Endian-documents, this method throws an exception. If a document could not be opened, an exception is thrown as well.

6.2 SaveClassifier

To save the classifier including all languages models, the interface specifies

```
virtual void SaveClassifier (const std::wstring &path) =0;
```

The parameter `path` specifies the saving folder (**not file!**). A folder is required since multiple files are used to save the classifier. If a non-existing folder is passed, an exception will be thrown instead of creating a new folder since the STL provides no functionality for creating folders. No warnings or errors will be created when existing files are overwritten. In addition, the naming schema is not changeable. Therefore it is recommend to put each classifier in an empty folder.

A model file is created for each language model. In addition, an index file is created, which contains a list of all model-files and all parameters of the classifier. This modular way of saving enables an easier way of adding and removing additional language models.

The structure of the model-files and index-files are individual and discussed in the respective chapters.

6.3 LoadClassifier

To load the classifier including all languages models, the interface specifies

```
virtual void LoadClassifier (const std::wstring &path) =0;
```

Loading behaves almost similar to saving the classifier, discussed in the previous chapter.

6.4 ClassifyModel

For the classification process, the interfaces specifies

```
virtual std::wstring ClassifyModel(const T &model) =0;
```

A document model is expected as parameter. The language identifier of the model, which is mostly similar to document models, is returned. In case, 2 models causing the same similiarity measures “???” will be returned. But this is in general mostly unlikely unless the classifier has one languages model twice.

All models of the present systems have a document model creating constructors. This enables calls such as

```
obj.ClassifyModel(T(...));
```

6.5 GetResultMap

`GetResultMap` is a constant method, which returns a constant reference to a map object. The language identifier is used as the key, where the respective similarity measures from the last `ClassifyModel`-call used as the value. This enables a closer look at the last classification.

7 Inherited Classes

In this chapter, the implementation of the approaches will be discussed briefly. A full list of all class member functions and class member variables can be found in the documentations.

7.1 CNGProfClassifier

The `CNGProfClassifier` uses Cavnar’ and Trenkle’ approach to identify the language. A custom class `CNGProfModel` is defined to store the N-Grams profiles.

7.1.1 Constructor

The standard constructor is defined:

```
CNGProfClassifier (unsigned max_profile_size, EMaxValType max_val_type, float
        max_val_scl =1.0f, unsigned min_n =5, unsigned max_n =5)
```

The maximum profile size is specified using the first parameters. All profiles are trimmed to this size after the training process (the remaining information is dropped). The document profiles are not trimmed, but for classification the specified amount is used only. A loading process will fail if any provided profile has a lower maximum profile size. There is actually a current profile size (cf. `m_uProfileSize`) and a maximum profile size (cf. `m_uProfileMaxSize`). The current profile size is the value used in most case. It can be set using `SetProfileSize` with the maximum profile size as its upper bound.

The max-value is specified using the next 2 parameters. The classifier supports dynamic max-values. One can use whether the language profile size or the document profile size. In addition, this value can be scaled by the second parameters.

The two last parameters specify the under and upper bound of N-grams type to be used: $N \in [\mathrm{min\_n}, \mathrm{max\_n}]$.

7.1.2 The index file

The index file is a human-readable plain text file encoded with Little Endian. Each line represents an independent piece of data. The following order is used:

1. **Version Number**: Currently 1. For any other version number, `LoadClassifier` will throw an exception.
2. **Maximum Profile Size**: If the specified maximum profile size is too high, LoadClassifier will throw an exception.
3. **Language Profile List:** Starting from line 3, every line presents another file-path of a new model. It doesn't matter if there is an empty last line or not.

Adding a new language model is easily done by adding a new text line. There is no addition validation mechanism for the maximum profile size. Adding a language model with a higher maximum profile size is stable. But entries of the affected profile above the specified maximum will never be used but it is still loaded, resulting in the consequence of wasting loading time.

In contrast, adding a language model with a lower maximum profile size leads to a higher error-rate because of an inappropriate max-value. The affected profiles will have a reduced probability of being chosen since the evaluation has shown that the optimum of the max-value depends on the language profile size (in case, the language profile is larger than the document profile size, otherwise it depends on the document's profile size).

7.1.3 The model file

The model files are binary files, having the following structure

1. **File Version number:** The first 4 bytes is interpreted as file version number of data type unsigned (4 bytes)
2. **Language Identifier:** 4 wide character including the '\0' (8 bytes).
3. **MaxN:** 4 bytes (unsigned).

4. **Mapping Field Size:** 4 bytes
5. **Mapping Field:** variable length. The mapping field is a continuously enumeration of N-Gram length. The '\0' is counted as well.
6. **N-Grams:** variable length. The N-grams are encoded using Little Endian. The first N-gram is the profile's first item.

7.2 CVectSPClassifier

The `CVectSPClassifier` use the Vector Space Model as described in the previous chapter to identify language. A custom class `CVectSPModel` is defined to represents the vectors.

The classifier contains a filtering mechanism. The N-grams are counted and normalized by their total amount. Afterwards a threshold value is used for filtering.

7.2.1 Constructor

The standard constructor is defined

```
CVectSpClassifier (double threshold, unsigned min_n =1, unsigned max_n =5)
```

`threshold` is the threshold value used for filtering. The two last parameters specify the under and upper bound of N-grams type to be used: $N \in [\mathrm{min\_n}, \mathrm{max\_n}]$.

7.2.2 The index file

The index-file is quite similar to the index-file of `CNGProfClassifier`. Instead of the Maximum Profile Size, the threshold is written in the file.

There is no validation for the loading or saving process except for the version number. Adding language models to the classifier with a different threshold might leads to a higher error-rate. A model with a lower threshold contains more data, resulting in more hits (less 0-values) when comparing the language model and document model' N-Grams. Ultimately, this leads to a lower distances than a model with a higher threshold could get! In contrast, a model with a higher threshold contains less data, resulting in more miss (more 0-values) and ultimately in higher distances.

7.2.3 The model file

Same as CNGProfClassifier. The respective frequency of each N-gram is saved directly after the N-gram itself:

$$< N - Gram >< freq >< N - Gram >< freq > \cdots$$

7.3 CMarkovClassifier

`CMarkovClassiifier` is the implementation of Ted Dunning's approaches. The implementation has got some customization.

A filtering capacity has been added to the classifier. All transition probabilities must be higher than a certain threshold value. **This is not the same as** `CVectSPClassifier`**!**

Instead of the "Laplace Sample Correction", a minimal probability min_prob is used. It is computed by

$$\text{min\_prob} = \frac{\text{min\_prob}}{\text{scale}}$$

The minimal probability is the threshold used for filtering divided by some scalar. No remarkable effects founded for the scalar! The same applies for the Laplace Sample Correction.

7.3.1 Constructor

The standard constructor is defined as

```
CMarkovClassifier (double threshold, double dMinProbScl = 100.0, unsigned
                                 min_n =1, unsigned max_n =5)
```

`threshold` is the threshold value used for filtering. `dMinProbScal` is the scalar used to compute the `min_prob` value. The two last parameters specify the under and upper bound of N-grams type to be used: $N \in [\text{min\_n}, \text{max\_n}]$.

7.3.2 The index file

Same as CVectSPClassifier. The `min_prob` scalar is not stored!

7.3.3 The model file

Same as CVectSPClassifier. The transition probability is saved instead of the frequency.

III Evaluation

The training corpus consists of various documents of 7 different languages collected from the Project Gutenberg[7]. The documents have been inspected manually. All line breaks have been removed as well as anything written in foreign languages such as the document header and the license information. The table below illustrates the training size of each language:

Language (ISO 639)	Training size in MB
deu	2.78
eng	3.40
fra	4.03
ita	4.97
nld	4.58
pol	0.371
por	0.291
spa	5.81

For the testing purpose, documents from various sources (Golem[8], Heise[9], Tageschau[10], BBC[11], NYT[12], CNN[13], Guardian[14], Le Monde[15], Nouvel Obs[16], Corriere Della Sera[17], La Repubbilca[18], El Pais[19], ABC[20]) has been collected. All line breaks have been removed. No further inspection or filtering was applied. Samples have been automatically generated from each document. No random seed "`srand()`" were used, resulting in the same sample-set for all evaluations. There might be test samples containing more foreign languages than the native languages.

Word boundaries were ignored when extracting a sample of length k. E.g. extracting a sample of length k=20, the string is split after 20 characters: "is string will be cu" (from "This string will be cut"). In this way, the test samples are not error-free but the length of the test-samples is more precisely and leads to more comparable results. Test samples of length 20 to 80 were used in all evaluation. The first approach implemented and evaluated was the N-Gram Profile. With test-samples of 80 characters, it already achieved a precision of about 99%. That's the reason for the length of the sample.

[7] http://www.gutenberg.org/wiki/Main_Page
[8] http://www.golem.de
[9] http://www.heise.de
[10] http://www.tagesschau.de
[11] http://www.bbc.co.uk
[12] http://www.nytimes.com
[13] http://www.cnn.com
[14] http://www.guardian.co.uk
[15] http://www.lemonde.fr
[16] http://tempsreel.nouvelobs.com
[17] http://www.corriere.it
[18] http://www.repubblica.it
[19] http://www.elpais.com
[20] http://www.abc.es

The training corpus as well as the testing corpus is encoded with UTF-16LE.

To make the method more comparable the feature extraction method of Cavnar & Trenkle is used for all 3 approaches. This is called "TokAndCount" since the test documents are tokenized and then the features are extracted. Afterward the NoSpace feature extraction method for OCR replaces the "TokAndCount" method. In the end, the new and old feature extraction method were used to identify erroneous test-samples.

8 Compare the 3 approaches

The result of the first evaluation is presented in the chart below.

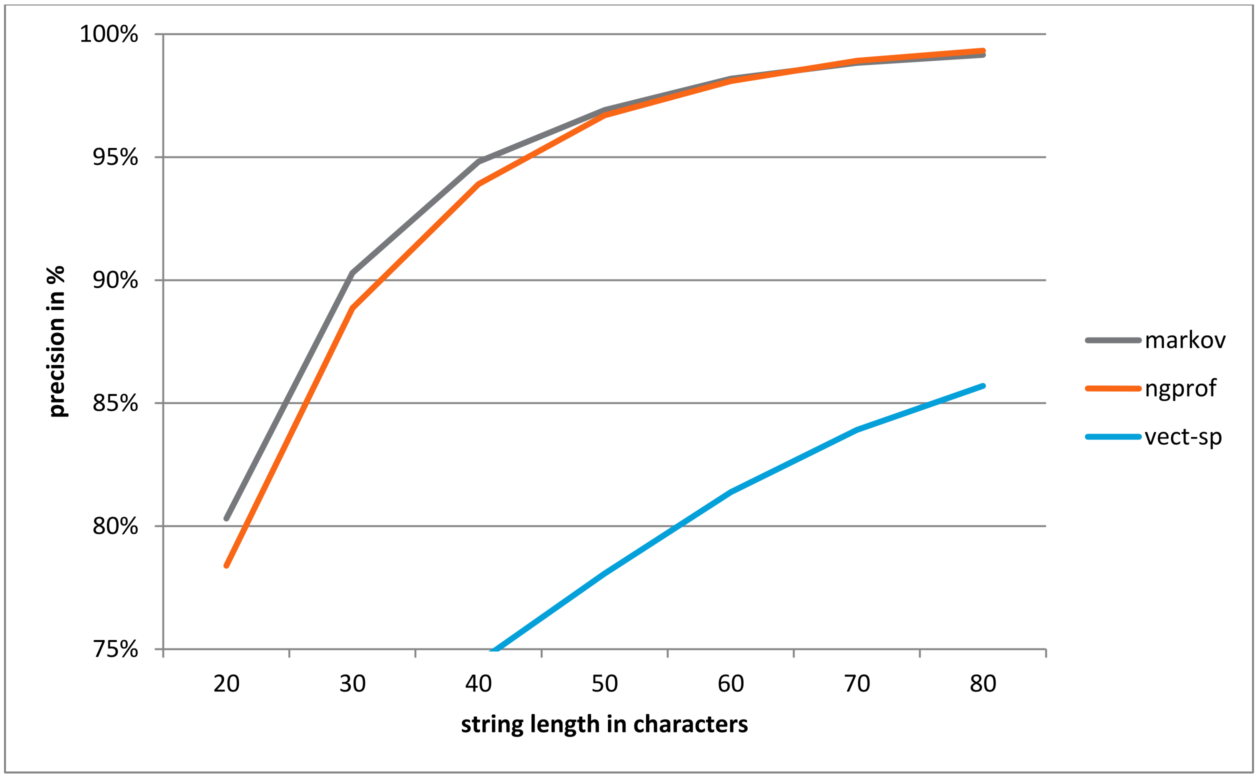


Figure 5: Compare 3 approaches using TokAndCount

Dunning' approach does perform a bit better than Cavnar' & Trenkle' approach when dealing with test-samples shorter than 70 characters. Starting by 70, NG-Profile takes the lead. With just 80 characters a precision of about 99% could be achieved. The full result is listed in the table below.

string-length	markov	ngprof	vect-sp
20	0.803127	0.783900	0.585857
30	0.902947	0.888600	0.692929
40	0.948126	0.939000	0.744751
50	0.969112	0.967000	0.780754
60	0.981896	0.980900	0.813831
70	0.988342	0.989200	0.839107

80	0.991557	0.993300	0.856997

8.1 Related Work

The two scientist Muntsa Padrós and Llúıs Padrós compared Dunning' approach "Markov Model" with Damashek' "Trigram Frequency Vectors Technique" and Cavnar' & Trenkle' "Ad-hoc Ranking". The following parameters have been evaluated: the training size, test size, number of distinguished languages and the languages set (similarity of the languages).

Based on their result, a system, trained with 50K words, is sufficient since the precision does not rise significantly with a higher training size. The number of distinguishable languages affects the precision. Especially, the similarity of the chosen languages (languages set). E.g. Spanish and Catalan (85%) since the dialect of the two languages are very similar. Dunning's approach always achieved the highest precision - especially when dealing with small texts. The authors numbered the minimum required number of character for a precision of 95% (99%, if we exclude Cavnar' N-Gram Profile Technique) at 500; cf. (Padró Cirera and Padró Cirera 2004, 3, 4, 6).

9 Compare TokAndCount with NoSpace

In this test, the counting method "TokAndCount" and the new counting method "NoSpace" is compared:

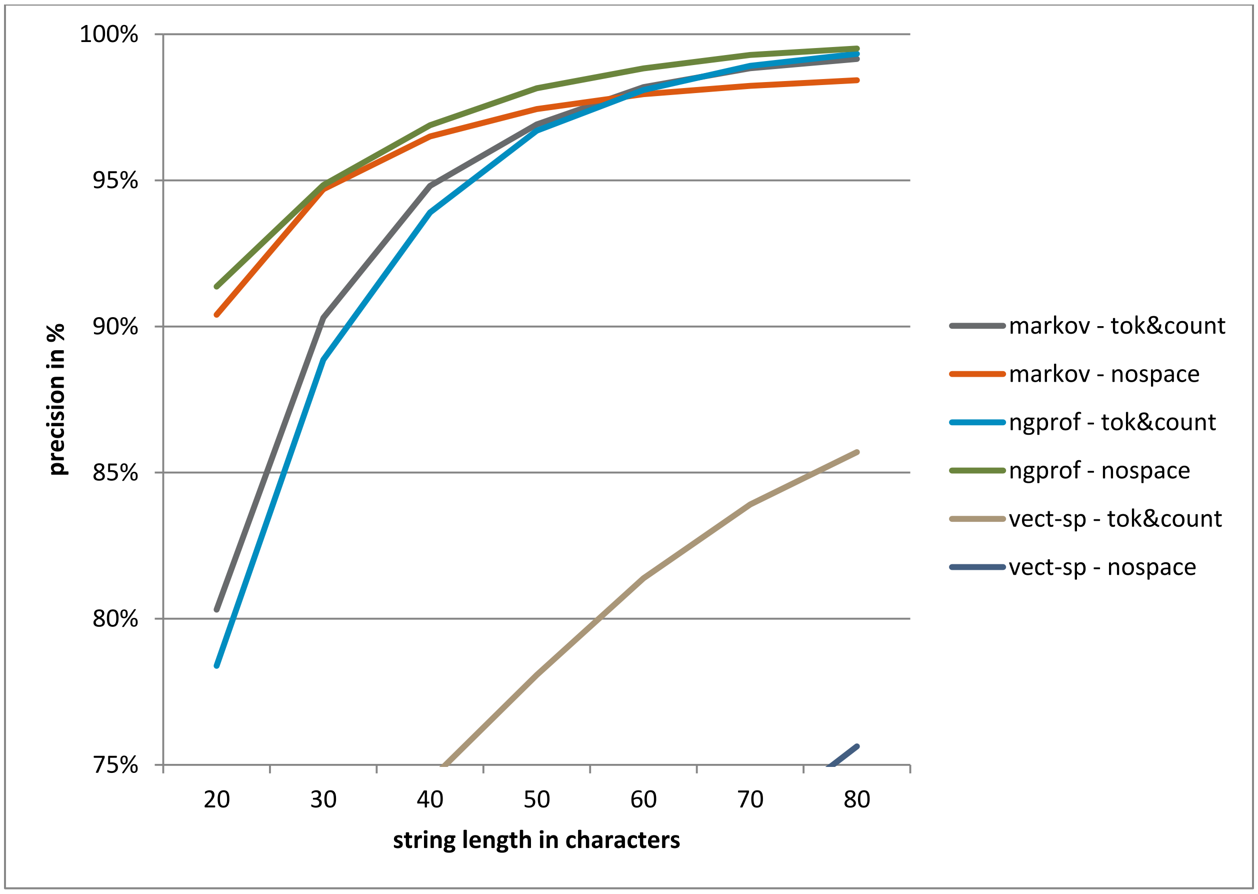


Figure 6: Compare TokAndCount with NoSpace

The new counting method does increase the precision of NG-Profile and Markov from around 80% to about 90%. For test-samples shorter than 60 characters, the two leading approach does performs better with “NoSpace”. But Starting with 60 Dunning’ approach is overtaken.

However, Cavnar & Trenkle’ approach using “NoSpace” takes the leads overall. In Contrast to the other two, “NoSpace” does reduce the precision of the vector space model.

	markov		**ngprof**		**vect-sp**	
string-length	**TokAndCount**	**NoSpace**	**TokAndCount**	**NoSpace**	**TokAndCount**	**NoSpace**
20	0.803127	0.903983	0.783900	0.913645	0.585857	0.419134
30	0.902947	0.946975	0.888600	0.948368	0.692929	0.530096
40	0.948126	0.965029	0.939000	0.968879	0.744751	0.597892
50	0.969112	0.974417	0.967000	0.981527	0.780754	0.647966
60	0.981896	0.979466	0.980900	0.988293	0.813831	0.693605
70	0.988342	0.982357	0.989200	0.992911	0.839107	0.728299
80	0.991557	0.984263	0.993300	0.995076	0.856997	0.756339

10 Identify noise tainted samples

In this test, 20% of each test-sample was automatically replaced with numbers. Adding whitespace would be affectless since they are dropping them anyway in case of adding whitespace by extending the string length. Just replacing a character with whitespace seems quite unrealistic since an OCR application rather misclassifies a character than drop it. The chart below illustrates the result

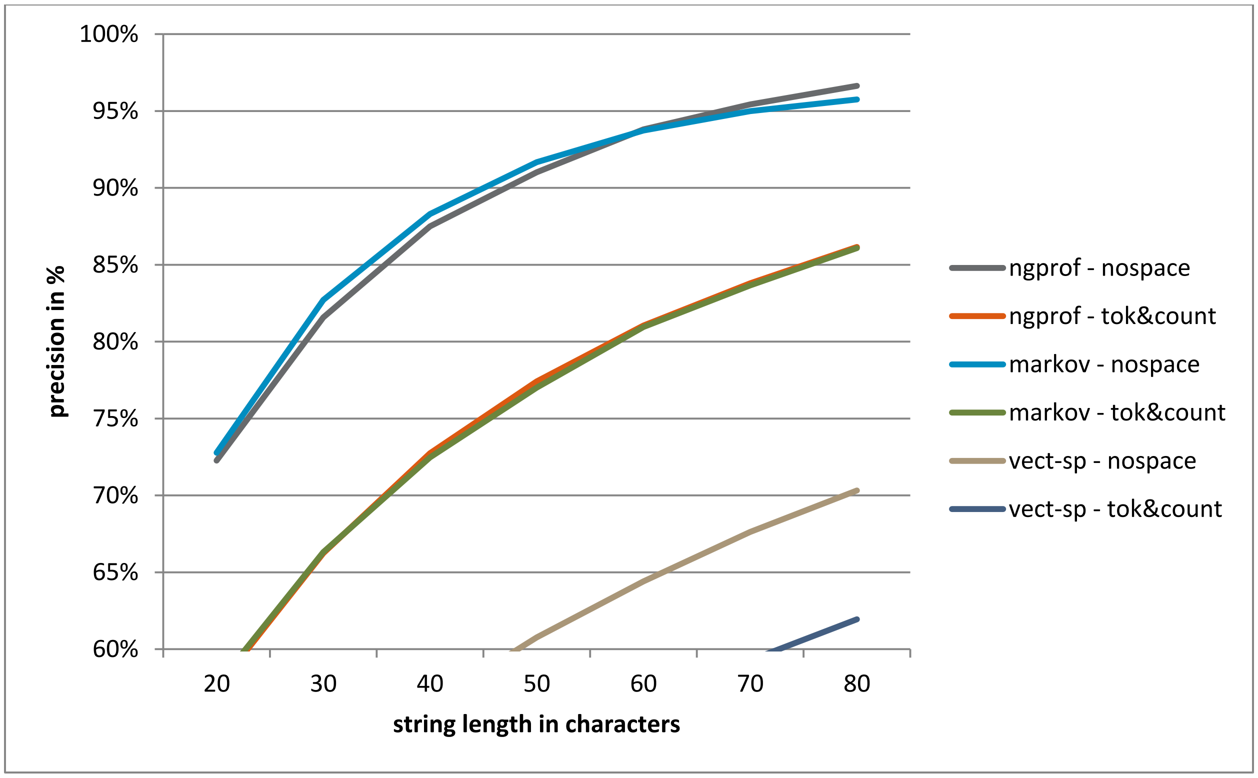


Figure 7: Identify noise tainted samples

Using the NoSpace-counting method, Markov performs slightly better than NG-Profile when dealing with test-samples shorter than 60 and tainted with noise. Starting by 60, NG-Profile takes the lead.

Using the TokAndCount-counting method, Markov and NG-Profile performs almost similar. In comparison to the NoSpace counting method, the precision is around 20% lower. In practice, the precision is even lower when additional false-whitespace is part of the document.

	ngprof		markov		vect-sp	
string-length	NoSpace	TokAndCount	NoSpace	TokAndCount	NoSpace	TokAndCount
20	0.722634	0.574331	0.727697	0.576462	0.444697	0.364166
30	0.815724	0.662400	0.827145	0.663069	0.512490	0.424055
40	0.875021	0.727462	0.882972	0.724703	0.564310	0.477421
50	0.910131	0.774159	0.916710	0.770062	0.607614	0.520041
60	0.938028	0.810455	0.937345	0.809531	0.644097	0.556103
70	0.954228	0.837972	0.949945	0.836586	0.676234	0.592745
80	0.966386	0.861586	0.957524	0.860862	0.703145	0.619448

11 Compare computing time

Beside the precision, the computing time has been evaluated regardless the loading time of the classifier itself. The time measured including the time required to generate a document model and the classification itself. The result is illustrated in the chart below.

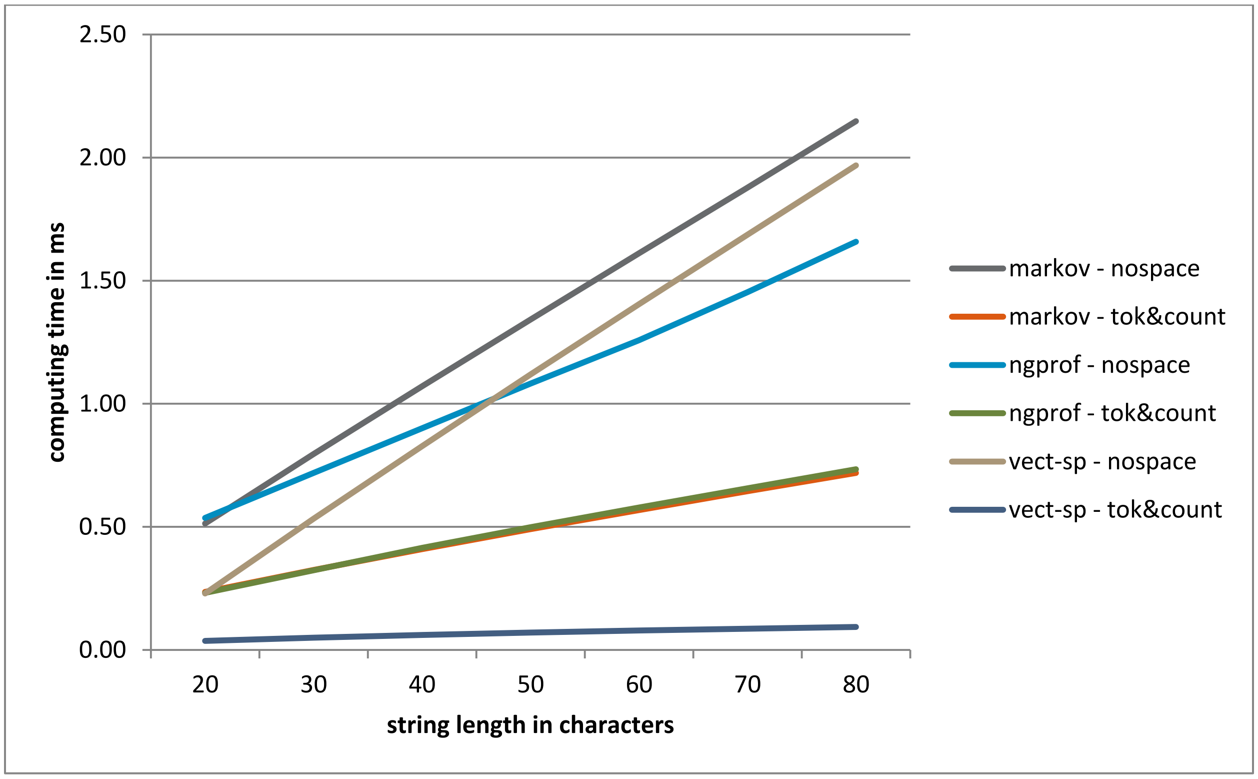


Figure 8: Compare computing time

The counting method TokAndCount performs faster than the NoSpace counting method. Dunning approach using the NoSpace requires the most computing time. The fastest approach is the Vector Space Model using TokAndCount. But all in all it is still within a couple of milliseconds.

	markov		**ngprof**		**vect-sp**	
string-length	**NoSpace**	**TokAndCount**	**NoSpace**	**TokAndCount**	**NoSpace**	**TokAndCount**
20	0.5134960	0.2356380	0.5359850	0.2316520	0.2294950	0.0370197
30	0.7956820	0.3245870	0.7188480	0.3235170	0.5327670	0.0500000
40	1.0710200	0.4092810	0.9005650	0.4144500	0.8278260	0.0610580
50	1.3423600	0.4905020	1.0816300	0.4988490	1.1186700	0.0704424
60	1.6112500	0.5680050	1.2582800	0.5787870	1.4044200	0.0788504
70	1.8780300	0.6448840	1.4534100	0.6568730	1.6864900	0.0863582
80	2.1477000	0.7188140	1.6579900	0.7341610	1.9683800	0.0932807

IV Conclusion / Future Work

In the present thesis, the 3 main approaches of language identification have been discussed:

1. Ad-hoc Ranking
2. Vector Space Model
3. Bayes Decision Rules and Markov Chain

The 3 approaches are implemented and compared in a new framework “MiLiD”. In contrast to previous works, same feature extraction methods (TokAndCount and NoSpace) and types (N-grams; N ranging from 1 to 5) are used in the comparisons to achieve more comparable results.

As shown, the Ad-hoc Ranking and Bayes Decision Rules and Markov Chain perform best. In case the samples are error-free, only 20 characters are required in combination with the counting method NoSpace to achieve a precision of about 91% when classifying 7 different languages including English, German, Spain, and so on.

The NoSpace feature extraction method is proposed by the present thesis for erroneous samples given by OCR. The approach drops all kind of information that are too error-prone to OCR. Alternatively, a system could use OCR documents to train the classifier. But this would bind the language classifier to a specific OCR algorithm.

Though Cavnar & Trenkle’ approach and Dunning’ approach have practically the same precision, I would recommend the first approach regarding to the memory usage as well as computing time.

As demonstrated in the present thesis, more features won’t necessarily cause higher precisions than less features when dealing with short documents. In future work, additional information filtering can be applied to remove N-grams that occur in all supported languages on almost the same rank. In addition, the precision might increase if bigrams and trigrams are solely used instead of all N-grams, N ranging from 1 to 5, since they are more suitable for short documents as reported in related works.

Bibliography

Artemenko, Olga, and Margaryta Shramko. *Entwicklung eines Werkzeugs zur Sprachidentifikation in mono- und multilingualen Texten.* Master Thesis, Hildesheim: University of Hildesheim, Information Science, 2005.

Cavnar, William B., and John M. Trenkle. "N-Gram-Based Text Categorization." In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 161--175. Michigan: CiteSeerX, 1994.

Damashek, Marc. "Gauging similarity with n-grams: Language-independent categorization of text." *Science* 267 (February 1995): pp. 843 - 848.

Dunning, Ted. *Statistical Identification of Language.* Technical Report, New Mexico State University: CiteSeerX, 1994.

Ferreira da Silva, Joaquim, and Gabriel Pereira Lopes. "Identification of Document Language is Not yet a Completely Solved Problem." In *CIMCA '06: Proceedings of the International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce*, 212. Washington, DC: IEEE Computer Society, 2006.

Grothe, Lena, Ernesto William, De Luca, and Andreas Nürnberger. "A Comparative Study on Language Identification Methods." In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, edited by Nicoletta Calzolari, et al. European Language Resources Association (ELRA), 2008.

Lins, Rafael Dueire, and Paulo Gonçalves Jr. "Automatic language identification of written texts." In *Proceedings of the 2004 ACM symposium on Applied computing (SAC '04)*, 1128 - 1133. New York, NY: ACM, 2004.

Ljubešić, Nikola, Nives Mikelić, and Damir Boras. "Language Identification: How to Distinguish Similar Languages?" *University of Zagreb, Department of Information Sciences.* 2007. http://infoz.ffzg.hr/ljubesic/nlnmdb_iti07.pdf (accessed January 16, 2010).

Padró Cirera, Montserrat, and Lluís Padró Cirera. "Comparing methods for language identification." In *Procesamiento del lenguaje natural*, 155-161. Barcelona: Sociedad Española para el Procesamiento del Lenguaje Natural, 2004.

Poutsma, Arjen. "Applying Monte Carlo Techniques to Language Identification." In *In Proceedings of Computational Linguistics in the Netherlands (CLIN)*, 179--189. SmartHaven, Amsterdam: Rodopi, 2001.

Prager, John M. *Linguini: Language Identification for Multilingual Documents.* Vol. 2, in *Thirty-Second Annual Hawaii International Conference on System Sciences*, 2035. Los Alamitos, CA: IEEE Computer Society, 1999.

Eidesstaatliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich diese Bachelorarbeit selbstständig und ohne unzulässige Hilfe angefertigt habe.

Die verwendeten Quellen sind im Literaturverzeichnis vollständig angegeben.

Berlin, den 8. Juni 2010

Do, Hoang Viet