# Unsupervised learning of word separators with MDL

Aris Xanthos, François Bavaud

University of Lausanne – CH-1015 Lausanne – Suisse

## Abstract

This paper describes a novel algorithm for the unsupervised learning of word separators in raw text. The algorithm requires no language-specific knowledge regarding the text being processed. It relies solely on distributional properties of the text and uses the *minimum description length* (*MDL*) principle in order to partition characters into two subsets that correspond well with the traditional notion of letters and separators. The distinction between these types of characters emerges as an optimal solution to the problem of simultaneously compressing two elements: the lexicon that is obtained by tokenizing the text using the hypothesized separators, and the representation of the text under this lexicon. The performance of the proposed algorithm is evaluated on the basis of electronic text in English, French and German.

**Keywords:** unsupervised learning, word separators, tokenization, minimum description length.

## 1. Introduction

Text tokenization is the process by which a finite sequence of characters is converted into a sequence of higher-level units, typically words (Webster and Kit, 1992). Although it is a necessary step in virtually any form of computerized text analysis, tokenization is not firmly established as a proper area of natural language processing. Indeed, it has long been considered part of the heterogeneous set of procedures referred to as *preprocessing*, along with such tasks as deleting markup or converting sentence-initial capital letters, for instance.

To some extent, this ancillary status originates in the commonly held belief that tokenization is a trivial problem. Indeed, in many written languages, word boundaries are explicitly marked by *separators*, i.e. special characters such as whitespace, carriage return, punctuation marks, and so on. In these so-called *segmented* languages, knowledge of the set of characters that may function as separators is sufficient for correctly identifying a large proportion of word boundaries, and hence of word tokens [1]. However, even in such languages, most separators sometimes lose this status, as illustrated by whitespace in multi-word expressions, line-ending hyphens, punctuation marks in abbreviations and numbers and so on (see e.g. Mikheev, 2002). In other words, tokenization is plagued with the same kind of ambiguities that pervade all aspects of natural language processing.

Besides the mere knowledge of potential separators, tokenizers are often enriched with additional information in order to handle ambiguous cases. Such information ranges from contextual rules to full-fledged lexicons (Grefenstette and Tapanainen, 1994; Grefenstette, 1999). In general,

[1] Since the focus of this paper is on discovering word separators, we will not delve into the issues related to the tokenization of Asian languages such as Chinese and Japanese, whose written form does not include systematic separators (see e.g. Teahan et al., 2000).

these resources aim at characterizing the conditions under which a separator should rather be treated as an ordinary character – or, less frequently, the other way round. These conditions are sometimes very general, in the sense that they apply to a lot of texts of different origin; for example, many languages share the convention that a period immediately surrounded by digits should not be treated as a separator. In other cases, the conditions are specific to a particular language, if not to a particular register, genre, field, etc. within a language. Thus, while providing a tokenizer with more knowledge usually makes it more accurate, it makes it less general in terms of the range of data that it can successfully handle.

Unsupervised approaches to text tokenization take this idea to the extreme. In this framework, the main issue is to determine the minimal amount of knowledge that is needed in order to tokenize a text. This question is both of practical and scientific relevance. From a practical point of view, it is obviously desirable to design tokenization systems that can be applied to the broadest possible range of data. Such versatility is particularly important for handling noisy data that include a lot of errors and inconsistencies, and for which maintaining updated disambiguation resource may not be feasible (Wrenn et al., 2007). From a scientific point of view, unsupervised methods for word segmentation are frequently offered as evidence of the learnability of certain aspects of lexical knowledge, in particular since the influential work of Saffran et al. (1996) on infants' statistical learning abilities.

Most unsupervised tokenization methods have been designed for processing orthographic or phonetic transcriptions which do not include explicit separators–symbolic approximations to the continuous stream of spoken language (see e.g. Olivier, 1968; Wolff, 1977; Elman, 1990; Brent and Cartwright, 1996; de Marcken, 1996; Bavaud and Xanthos, 2002; Xanthos, 2004). However, a few attempts to apply unsupervised tokenization to segmented languages have been reported. In some of these studies, the tokenization process does not lead to an explicit distinction between separators and other characters. Rather, the resulting tokens are more or less consistently decorated with separators at their edges (de Marcken, 1996; Nevill-Manning, 1997, Kit and Wilks, 1999). Other studies investigate specific heuristics for identifying separators *after* performing some form of unsupervised tokenization: Hutchens and Alder (1998) propose to label as separators those characters whose frequency in token-final position is higher than some arbitrary threshold; Wrenn et al. (2007) attempt to remove sentence-ending punctuation from the right edge of tokens by inserting new boundaries before characters whose transition probability is lower than some threshold.

To the best of our knowledge, the proposal of Kempe (1999) has been the only attempt to identify separators *prior to* tokenization (based on the assumption that character transitions are less predictable when moving from separator to adjacent word than within words). Kempe concludes his paper by encouraging the "search for criteria (based on the corpus itself and on the obtained result) to evaluate the 'quality' of segmentation" (p.12). The present contribution is an attempt to provide such a criterion. In particular, we seek to identify separators based on their "usefulness" for tokenizing the text. In this context, a character's usefulness is equated with its ability to produce a compact representation of the text when it is added to the set of separators. This intuitive notion of usefulness is given a rigorous content on the basis of information-theoretic principles, and in particular it is expressed in the specific framework of *minimum description length* (*MDL*) inference (Rissanen, 1989). We describe a greedy algorithm that examines the consequences of labelling each character as a separator and incrementally expands the set of separators by selecting the most "useful" one at each stage. The algorithm is evaluated on the basis of raw text in English, French and German.

The rest of this paper is organized as follows. In the next section, we describe the proposed algorithm, along with the data and methods used for its evaluation. The evaluation results are presented and discussed in section 3. Section 4 summarizes our findings and discusses ways in which this line of research can be pursued.

## 2. Method

### *2.1. Learning algorithm*

Let $T \in A^*$ denote the text to be processed, i.e. a finite sequence of *characters* taken from set $A$. Define $\mathsf{H}(A)$ as the set of all possible bipartitions of $A$ into a non-empty set of *letters* [2] and a possibly empty set of *separators* (there are $|\mathsf{H}(A)| = 2^{|A|} - 1$ such partitions). For a given partition $h \in \mathsf{H}(A)$, we denote by $L^h$ the set of letters in $h$, and by $\overline{L^h}$ the set of separators. The goal of the proposed algorithm is to select a partition in $\mathsf{H}(A)$ with no prior information regarding the "actual" use of separators in $T$. This is achieved by means of a model selection process based on the *minimum description length* (*MDL*) principle (Rissanen, 1989).

In the MDL framework, $T$ is conceived as an observation which should be explained. It is further assumed that the burden of $T$'s explanation can be shared by two distinct components: (i) a *model* which specifies the (possibly infinite) set of texts that could have been observed in place of $T$ and (ii) a *representation* of $T$ under this model, whose responsability is to uniquely identify $T$ among all texts that are compatible with the model. The MDL principle dictates that the selection of a model for explaining the data should be based on two criteria: on the one hand, the model should be simple, in the sense that it should be possible to specify it with a small number of statements; on the other hand, the model should make it possible to describe the observed data with a small number of statements.

In general, these two objectives (simplicity of the model and simplicity of the data representation under the model) act as contradictory forces in the process of model selection. It is usually possible to make the representation of the data very simple by crafting a very complex model, which in turn will fail to generalize to unseen data. On the other hand, an excessively simple model will place most of the burden of explanation on the data representation, and thus it will fail to account for the inherent structure of the observations. The fundamental insight of MDL inference is to seek a balanced distribution of information between model and data representation by *simultaneously* minimizing their length.

Putting this program into practice requires the rigorous definition of a coding scheme for each possible model and associated data representation. Indeed, such a scheme is needed in order to obtain a quantitative evaluation of the cumulated length of these two elements – which we attempt to minimize. Obviously, this evaluation depends crucially on the choice of a particular coding scheme. In order to reduce this lurking arbitrariness, it is desirable to design the coding scheme in such fashion that the length of the encoded model and data representation is as strongly compressed as possible, and in particular as close as possible to the minimal bounds set by information and coding theory.

[2] For the sake of readability, the term *letters* is used to refer to all characters that are assumed not to function as word separators (whether alphabetic, numeric, or other).

The detailed presentation of this approach (as applied to the problem of text tokenization) proceeds in three steps. First we describe a set of conventions which, given a text $T \in A^*$, uniquely associate each partition $h \in \mathsf{H}(A)$ to a probabilistic *model* (essentially a pair of lexicons) and a *representation* of $T$ under this model (which can be thought of as a series of instructions for generating $T$ on the basis of the lexicons). Then we define a way of compactly encoding each possible pair of model and data representation in binary format, and derive an expression of the length of the resulting binary string. Finally we present a greedy algorithm which attempts to find the partition $h \in \mathsf{H}(A)$ minimizing the cumulated length of the corresponding model and data representation (in compressed binary format).

### *2.1.1. Partition, model, and data representation*

Given text $T \in A^*$, each partition $h \in \mathsf{H}(A)$ into letters $L^h$ and separators $\overline{L}^h$ can be associated with a unique segmentation of $T$ into a series of *words*, i.e. contiguous sequences of letters in $(L^h)^*$, in strict alternation with *separator words*, i.e. contiguous sequences of separators in $(\overline{L}^h)^*$. Based on this segmentation, it is straightforward to establish a list of distinct words $W^h$ and a list of distinct separator words $\overline{W}^h$. In what follows, we will call $W^h$ the *lexicon* (associated to $h$), and $\overline{W}^h$ the corresponding *separator lexicon*.

The pair $M^h := (W^h, \overline{W}^h)$ can be thought of as a *model* which generates $T \in A^*$, along with other texts. To that effect, we make the following assumptions:

1. Generation starts by choosing one of the two lexicons in $M^h$ at random. This lexicon becomes the current lexicon.
2. A word or separator word in the current lexicon is randomly emitted.
3. If $W^h$ has already been the current lexicon once, generation has a random chance of stopping. Otherwise it proceeds by switching the current lexicon and returns to step 2.

This process generates an infinite set of texts, which form a subset of all possible texts on $A$. The difference between this subset and $A^*$ is the knowledge about $T$ that is encoded in $M^h$.

In order to uniquely identify $T$ among all texts that can be generated by $M^h$, some extra information is needed. This information can be expressed as a series of instructions for $M^h$ to generate $T$. By analogy with what precedes, we assume that it consists of:

1. A bit indicating whether to start with a word or separator word.
2. A sequence of $n^h + \overline{n}^h$ integers corresponding to each of the $n^h$ word tokens and $\overline{n}^h$ separator word tokens in $T$, as segmented under $h$. Each integer is the index of an element of $W^h$ (or $\overline{W}^h$), and the lexicon is assumed to switch at each step.

These data form the *representation of T under* $M^h$, or simply *under h*. We write it $T^h := (t_0^h, t_1^h, \ldots, t_{n^h+\overline{n}^h}^h)$, where $t_0^h \in \{0,1\}$ and $t_1^h, \ldots, t_{n^h+\overline{n}^h}^h$ are positive integers.

Taken together, $T^h$ and $M^h$ give a full account of $T \in A^*$. They encode it in a way that can be uniquely decoded–under the assumptions described in this section. These assumptions are obviously arbitrary, but there is no way of using a code without sharing certains assumptions between encoder and decoder. The main point is that each partition $h \in \mathsf{H}(A)$ can be conventionally associated with a unique model $M^h$ and data representation $T^h$, and these constitute a full specification of $T \in A^*$.

### *2.1.2. Coding scheme and compressed length*

Following the minimum description length (MDL) principle, the aim of the proposed algorithm is to find the partition $h^0 \in \mathrm{H}(A)$ which minimizes the cumulated length of the corresponding model $M^{h^0}$ and data representation $T^{h^0}$. However, as mentioned earlier, length evaluation depends on the specific way in which models and data representations are encoded. So it would seem that this is by no means an objective criteria for selecting a partition.

Information theory provides us with a way of dealing with this issue. In this framework, it has been established that a message can be encoded in a way that is optimal, in the sense that its length is as small as possible (under certain constraints). Thus, rather than simply minimizing the length of models and data representations as expressed in any arbitrary code, it is desirable to minimize their *compressed* length as expressed in an optimal code. Note that we are not interested in actually encoding models and data representations: our aim is merely to evaluate their compressed length. Therefore we will use certain results in information theory without worrying about how to effectively construct codes with the desired properties.

In what follows, we introduce a coding scheme which translates each possible pair $(M^h, T^h)$ into a unique binary string in $\{0,1\}^*$. This string can be decoded unambiguously provided that the coding scheme is known. The scheme takes advantage of the redundancies in $(M^h, T^h)$ and provides a computable approximation of the compressed length of $(M^h, T^h)$.

Formally, a code is a function mapping every item in a given set to a *code-word* in a second set. In the MDL framework, it is both customary and convenient to use codes whose code-words are binary strings in $\{0,1\}^*$. Codes are also traditionally required to be *instantaneous*, in the sense that it should always be possible to tell that an entire code-word has been read before reading the message any further. This condition is also known as the *prefix-free* property, because it is satisfied if and only if no code-word is the prefix of another code-word.

Suppose that *C* is a prefix-free code mapping the elements of some set *B* to binary strings. Consider a sequence $s \in \{0,1\}^*$ of code-words in this code. Since *C* is prefix-free, there is no need for code-words in *s* to be explicitly separated; the decoder always knows when an entire code-word has been read. Suppose now that *s* is the beginning of a longer binary string, and that the rest of this string is encoded with some other code *C'*, possibly using the same binary strings as code-words. In such a case, how does the decoder know when to switch to the new code?

This problem can be solved by prepending *s* with a prefix-free binary code-word (in yet another code) which stands for the number *n* of code-words in *s*. Thus, after reading this initial code-word, the decoder knows how many code-words must be read using prefix-free code *C* before switching to *C'*. This can be thought of as the consolidation of *s* into a *list* of *n* items, and the added length of the initial code-word is the cost for specifying the list structure (see e.g. Goldsmith, 2001). There exists a number of prefix-free codes which associate each integer *n* with a compact binary code-word; the one we will use has code-words of length $\lambda(n) := \lfloor \log(n+1) \rfloor + 2\lfloor \log(\log(n+1)+1) \rfloor + 1$ bits (see e.g. Li and Vitányi, 1997) [3].

The proposed coding scheme makes extensive use of this notion of list structure. We will also draw on the following theorem: given a finite set of items with probabilities $p_1, \ldots, p_m$, there exists an optimal prefix-free binary code which has an expected code-word length (in

[3] The notation log denotes the base 2 logarithm and $\lfloor x \rfloor$ stands for the largest integer not greater than *x*.

bits) approximately equal to the *entropy* $H(p_1,\ldots,p_m) := -\Sigma_i\, p_i \log p_i$ over the probability distribution (Shannon, 1948). This code is optimal because it assigns the shortest code-words to the most frequent items.

For the sake of explanation, suppose that we wish to encode a message consisting in a pair $(M^h,T^h)$ where $M^h := \{w_1,\ldots,w_m\}$ is a single lexicon and $T^h := (t_1,\ldots,t_n)$ is a series of integers corresponding to indices of words in $M^h$ (that is, $t_k \in \{1,\ldots,m\}$ for $1 \le k \le n$). Consider first the question of encoding $T^h$. By Shannon's theorem, we know that there is a code which associates each integer in $\{1,\ldots,m\}$ with a binary code-word of approximate expected length $H(p_1,\ldots,p_m)$ bits, where $p_i$ is the probability of *i* (as estimated by its relative frequency in $T^h$). Thus the compressed length of $T^h$ can be evaluated as $L(T^h) := \lambda(n) + nH(p_1,\ldots,p_m)$ bits, where $\lambda(n)$ is the cost of consolidating the sequence into a *list* of code-words.

The encoding of $M^h$ is slightly more complicated. As a first approximation, we may assume that each word $w_i \in M^h$ is a sequence of $l(w_i)$ characters, and each character is encoded in 8 bits (as in the ASCII code, for instance). Thus the representation of word $w_i$ is a sequence of $8l(w_i)$ bits. In general, the set of all such sequences is not prefix-free, since there may be words which are the prefix of other words. In order to encode the sequence corresponding to $w_i$ in a prefix-free fashion, we turn it into a *list* of characters using the method described earlier. With this convention, we may encode $w_i$ with a prefix-free code-word of length $\lambda(l(w_i)) + 8l(w_i)$ bits. The compressed length of $M^h$ is evaluated as the sum, over all words, of the length of the corresponding code-word, plus the cost of specifying that $M^h$ is a list of *m* elements:

$$L(M^h) \quad := \quad \lambda(m) + \sum_i \left[\lambda(l(w_i)) + 8l(w_i)\right] \tag{1}$$

The total size compressed size of $(M^h,T^h)$ is then simply calculated as $L(M^h) + L(T^h)$.

This first scheme efficiently compresses the redundancies in $T^h$, but it fails to compress those in $M^h$. A better scheme may be designed by replacing the 8-bit string corresponding to each character in the lexicon with a binary code-word in an optimal prefix-free code. Let *A* denote the set of *l* distinct characters in words of $M^h$. Based on Shannon's theorem, we know that there exists an optimal prefix-free code which associates each integer in $\{1,\ldots,l\}$ with a binary code-word of approximate expected length $H(q_1,\ldots,q_l)$, where $q_r$ stand for the probability of *r* (as estimated by the relative frequency of the *r*-th character of *A* in $M^h$). This makes it possible to compress the representation of each word in $M^h$ from $\lambda(l(w_i)) + 8l(w_i)$ to $\lambda(l(w_i)) + l(w_i)H(q_1,\ldots,q_l)$ bits:

$$L(M^h) \quad := \quad \lambda(m) + \sum_i \left[\lambda(l(w_i)) + l(w_i)H(q_1,\ldots,q_l)\right] \tag{2}$$

With this new convention, it is in principle necessary to transmit the ordered list of characters types (in 8-bit or other encoding) even before transmitting the lexicon itself. However, since this list is constant over all pairs $(M^h,T^h)$ [4], its compressed length contributes a constant term to the overall compressed length of $(M^h,T^h)$; therefore it has no influence on the selection of a pair $(M^h,T^h)$ and will be ignored in the computation.

[4] Recall from previous section that the set of characters depends only on the text *T* that is being tokenized.

We may eventually return to the original question, i.e. how to evaluate the compressed length of the pair $(M^h, T^h)$ associated with a partition $h \in \mathrm{H}(A)$, given text $T \in A^*$. The following formulas take into account the fact that $M^h$ actually consists of two lexicons (words $W^h$ separator words $\overline{W}^h$), each of which is based on a separate set of characters (letters $L^h$ and separators $\overline{L}^h$). We assume that $(M^h, T^h)$ is encoded as a single binary string containing the two lexicons followed by the representation of the data under the model, and each of these three components is encapsulated in a list structure. Putting the bits together and using all assumptions stated previously, the compressed length of $(M^h, T^h)$ is finally evaluated as:

$$
\begin{aligned}
L(M^h,T^h) &:= L(M^h) + L(T^h) \\
L(M^h) &:= \lambda\left(\left|W^h\right|\right) + \sum_{i=1}^{|W^h|}\left[\lambda(l(w_i^h)) + l(w_i^h)H\left(q_1^h,\ldots,q_{|L^h|}^h\right)\right] \\
&\quad + \lambda\left(\left|\overline{W}^h\right|\right) + \sum_{i=1}^{|\overline{W}^h|}\left[\lambda(l(\overline{w}_i^h)) + l(\overline{w}_i^h)H\left(\overline{q}_1^h,\ldots,\overline{q}_{|\overline{L}^h|}^h\right)\right] \\
L(T^h) &:= 1 + \lambda(n^h) + n^h H\left(p_1^h,\ldots,p_{|W^h|}^h\right) + \lambda(\overline{n}^h) + \overline{n}^h H\left(\overline{p}_1^h,\ldots,\overline{p}_{|\overline{W}^h|}^h\right)
\end{aligned}
\tag{3}
$$

where:

- $q_r^h$ and $\overline{q}_{r'}^h$ denote the probability of the *r*-th letter in $L^h$ and *r'*-th separator in $\overline{L}^h$ respectively, estimated by their relative frequency in $W^h$ and $\overline{W}^h$.
- $p_i^h$ and $\overline{p}_{i'}^h$ denote the probability of the *i*-th word in $W^h$ and *i'*-th separator word in $\overline{W}^h$ estimated by their relative frequency in $T^h$.
- $w_i^h$ and $\overline{w}_{i'}^h$ denote the *i*-th word in $W^h$ and *i'*-th separator word in $\overline{W}^h$.
- $n^h$ and $\overline{n}^h$ denote the number of word tokens and separator word tokens in $T^h$.

### *2.1.3. Search procedure*

Given text $T \in A^*$, we apply the following greedy algorithm in order to search for the partition $h^0 \in \mathrm{H}(A)$ which minimizes the compressed length $L(M^{h^0}, T^{h^0})$ as defined in formula (3):

1. The algorithm first evaluates the compressed length of the partition in which the set of letters $L^h$ is the whole set of characters in *A*, and the set of separators $\overline{L}^h$ is empty. This partition becomes the current partition.
2. We construct each partition resulting from turning a single letter in the current partition to a separator. The compressed length of each modified partition is computed.
3. If none of the modified partitions leads to a decrease in compressed length (or if $\left|L^h\right| = 1$), the algorithm returns the current partition and stops.
4. Otherwise the modified partition that has minimal compressed length becomes the current partition and the algorithm returns to step 2.

This very simple procedure is not guaranteed to find the partition that globally minimizes compressed length, but it is guaranteed to find a local minimum in less than $|A|$ iterations.

## ***2.2. Evaluation***

### *2.2.1. Data*

The performance of the proposed algorithm is evaluated on the basis of three texts in three languages: English (Aaberg, 1945), French (d'Abbadie, 1868) and German (Abbe, 1989). Each text was downloaded from Project Gutenberg website (http://www.gutenberg.org), and was selected by retrieving the first author in each language (in alphabetical order). All texts are in ISO-8859-1 encoding and are submitted to the algorithm without any form of preprocessing [5]. Tab. 1 summarizes their main properties (*WS* stands for whitespace and *CR* for carriage return).

| *Language* | *# characters* | | *Non alphanumeric characters* |
|---|---|---|---|
| | *Types* | *Tokens* | |
| English | 87 | 346’779 | WS CR . , : ; ? ! - ‘ ( ) [ ] “ $ % @ # * _ / |
| French | 109 | 1’061’657 | WS CR . , : ; ? ! - ‘ ( ) [ ] “ $ % @ # * _ / « » ° |
| German | 107 | 966’055 | WS CR . , : ; ? ! - ‘ ( ) [ ] “ $ % @ # * _ / « » { } < > + = & · \| § \ ~ |

*Table 1: Main properties of texts in the test set*

It is worth noting that besides language variation, this corpus displays considerable genre variation. For instance, the German text contains ASCII-formatted tables which contribute to the relatively large diversity of non-alphanumeric characters in this text, which in turn is likely to affect the results of the algorithm.

### *2.2.2. Evaluation metrics*

There are various ways of assessing the performance of a tokenization method (see Habert et al., 1998 for a discussion). In this study, we perform a boundary-based comparison between the segmentation associated with the partition selected by the learning algorithm and the segmentation associated with a reference partition. By convention, the reference is taken to be the partition $h^\alpha$ in which all alphanumeric characters belong to the set of letters and all other characters to the set of separators.

Given text $T \in A^*$, reference partition $h^\alpha$, and a hypothesized partition $h \in \mathcal{H}(A)$, the evaluation process examines each potential boundary in *T* (i.e. each pair of consecutive characters *c* and *c'*). At each step, we increment (i) the count of *true* boundaries whenever *c* is a letter and *c'* is a separator (or the other way round) in $h^\alpha$; (ii) the count of *positive* boundaries whenever *c* is a letter and *c'* is a separator (or the other way round) in *h*; (iii) the count of *true positives* whenever *c* is a letter in both partitions and *c'* is a separator in both partitions (or the other way round). *Precision* is then calculated as the ratio of true positives to positive boundaries, and *recall* as the ratio of true positives to true boundaries.

Precision and recall are computed at each iteration of the algorithm described in section 2.1.3 above, when a new letter is moved to the set of separators. The result is a series of precision-recall pairs that document the performance of the algorithm throughout the process of separator learning. The curve for each text in our test set is reported in the next section.

## 3. Results

Figures 1 to 3 represent the evaluation results for each language. On each figure, the horizontal axis displays the 20 first separators that have been discovered by the algorithm in chronological order (*WS* stands for whitespace and *CR* for carriage return). Besides precision and recall, the compression rate obtained by adding each successive separator is also displayed (based on the compressed length evaluation defined in section 2.1.2.).

[5] In particular, each text retains its standard Project Gutenberg header and footer in English.

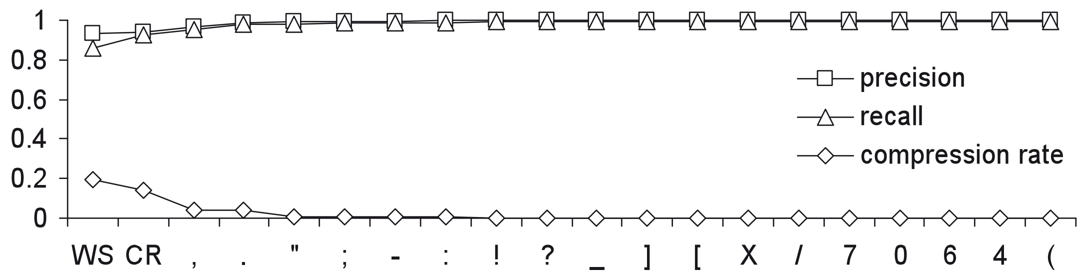

*Figure 1: Evaluation results for English data*

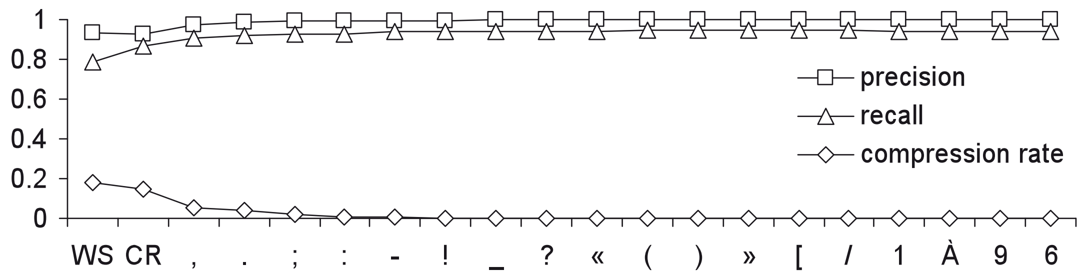

*Figure 2: Evaluation results for French data*

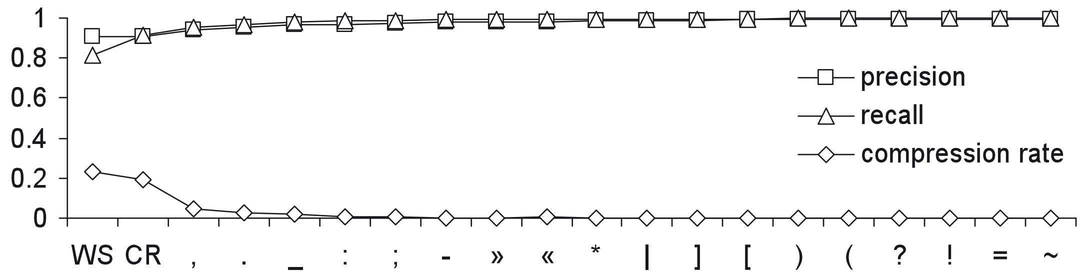

*Figure 3: Evaluation results for German data*

Learning dynamics display striking similarities in the three texts. In particular, the first four separators that the algorithm discovers are the same: WS, CR, comma and period. In each language, they are learned in the same order and yield approximately the same compression rate. WS and CR are the only separators which entail a large compression rate (about .2 and .15 respectively), and the rate falls below .01 after 6 or 7 separators (although it remains positive for another 20 to 30 separators).

Precision is greater that .9 after learning whitespace, and it does not change much when learning carriage return. At this point, the difference to unity can be explained by the fact that WS and CR are very useful for detecting left word boundaries (in these languages), but less so for right word boundaries. Indeed, there are often intervening separators, such as punctuation marks, between right boundaries and WS or CR. This interpretation is confirmed by the fact that comma and period are the next separators to be learned and their discovery leads to a considerable increase in precision (about .05).

The next separators to be learned include other punctuation marks (colon, semi-colon and, in the English and French texts, exclamation and question mark), along with hyphen and underscore. Paired separators such as guillemets, parentheses and square brackets follow closely (in particular in French and German). After a dozen separators have been learned, the compression rate is less than or equal to .001, and while the algorithm keeps finding new separators, their impact on precision and recall becomes hardly noticeable. At this point, the metrics are very close to their values at the end of the learning process, reported in Tab. 2 (except for precision in the German text, which keeps increasing slowly but steadily until the end).

| *Language* | *Precision* | *Recall* | *Separators selected by the algorithm* |
|---|---|---|---|
| English | 0.997 | 0.991 | WS CR , . “ ; - : ! ? _ ] [ X / 7 0 6 4 ( ) 9 @ $ % # |
| French | 1 | 0.941 | WS CR , . ; : - ! _ ? « ( ) » [ / 1 À 9 6 8 0 2 5 3 ] 4 7 “ @ $ # % ° |
| German | 1 | 1 | WS CR , . _ : ; - » « * \| ] [ ) ( ? ! = ~ / § } < > “ % { @ # ‘ · $ \ |

*Table 2: Precision, recall and separator set at the end of the learning process*

The first obviously erroneous separators are capital ‘X’ in the English text (15th separator) and capital ‘À’ in the French text (19th separator). The former has only 13 occurrences, out of which 10 are in roman numerals; in this context, ‘X’ appear only in word-initial or -final position, and therefore it constitutes a good candidate for inclusion in the set of separators. There are 137 occurrences of ‘À’ in the French text, all of which are in the single letter preposition *À*. Since ‘À’ never occurs word-internally, adding it to the set of separators does not lead to the incorrect segmentation of other words (as adding most other letters would). On the other hand, it makes it possible to reduce a number of [ separator word + *À* + separator word ] sequences into a single separator word, which is the most likely explanation for the observed decrease in compression length.

In the English and French text, the algorithm labels part or all of digits as separators. This is a clear divergence to the reference partition adopted for this experiment, but it must be recalled that letters (in the traditional sense) and digits tend to form separate tokens in the languages of our sample. From that point of view, classifying digits with letters rather than with separators was a debatable decision in the first place. A better solution would be to upgrade the model in order to account for three or more character classes. Note that in the German text, where the number of occurrences of digit sequences (2’739) is much larger than in the English and French text (358 and 403 respectively), digits have not been classified as separators.

Finally, while recall increases throughout the learning process and reaches very high values, there are a few separators that the algorithm fails to retrieve: apostrophe and asterisk in English and French; ampersand and plus sign in German. With the exception of the apostrophe, these are all rare characters which are used in rather specific contexts, and their misclassification has little influence on recall. The case of the apostrophe is more problematic, especially in the French text, where it is quite frequent (10’320 occurrences, versus 429 in the English text and 9 in the German text). Failure to classify the apostrophe as a separator accounts for most of the 6% of true boundaries that the algorithm does not retrieve in the French text.

## 4. Conclusion

In this paper, we have described a novel algorithm for the unsupervised learning of word separators in text. The algorithm requires no prior knowledge regarding the text to be processed. Following the minimum description length (MDL) principle, it attempts to find an optimal partition of characters into letters and separators. Each partition is conventionally associated with a model (i.e. a pair of lexicons) and a representation of the text under this model, and the partition’s optimality is evaluated on the basis of the conciseness of the associated model and data representation. Information and coding theory provide useful tools for reducing arbitrariness in the assessment of conciseness. The search for an optimal partition is done by means of a greedy algorithm which gradually expands the set of separators until there is no (direct) way to further compress the model and data representation.

Experiments conducted on English, French and German text show that the proposed algorithm is very efficient at learning word separators from these data. The most significant problem encountered–and arguably the only one–is the algorithm's failure to classify the apostrophe as a separator, which in the case of the French text reduces the boundary-based recall by more than 5 percents. There is also some variation across languages as regards the classification of digits as letters or separators. This suggests that a model with more than two classes of characters might be more appropriate.

The most important limitation of this approach is that it classifies character *types* into letters and separators–rather than character *tokens*. As discussed in the introduction, most separators can lose this status in certain contexts, and this is not accounted for by the present algorithm. In further research, we intend to investigate the possibility of applying MDL inference to the problem of learning contextual rules for disambiguating the function of separators in texts (such a rule might state that a period should not be treated as a separator when it is surrounded by digits, for instance).

Beyond the particular case of text tokenization, our aim in this paper has been to illustrate the use of MDL inference for solving a moderately complex problem of textual data analysis. We believe that this approach to unsupervised learning has the potential to be useful for a wide array of problems in natural language processing, and the present case-study is offered as an attempt to foster this line of research.

## References

Aaberg J.C. (1945). *Hymns and Hymnwriters of Denmark*. Des Moines (IA): Committee on Publication of the Danish Evangelical Lutheran Church in America. Retrieved October 30, 2009, from Project Gutenberg: http://www.gutenberg.org/etext/29666.

d'Abbadie A. (1868). *Douze ans de séjour dans la Haute-Éthiopie, tome premier*. Paris: Librairie de L. Hachette et Cie. Retrieved October 30, 2009, from Project Gutenberg: http://www.gutenberg.org/ etext/18812.

Abbe E. (1989). *Gesammelte Abhandlungen III*. Hildesheim: Georg Olms Verlag. Retrieved October 30, 2009, from Project Gutenberg: http://www.gutenberg.org/etext/19755.

Bavaud F. and Xanthos A. (2002). Thermodynamique et statistique textuelle: concepts et illustrations. In *Actes de JADT 2002*, vol. 2, Saint-Malo, 13-15 marzo, pp. 101-111.

Brent M.R. and Cartwright T.A. (1996). Distributional regularity and phonological constraints are useful for segmentation. *Cognition*, 61: 93-125.

Elman J.L. (1990). Finding Structure in Time. *Cognitive Science*, 14: 179-211.

Gammon E. (1969). Quantitative approximations to the word. In *Proc. of COLING-69 (3rd International Conference on Computational Linguistics)*, Sång-Säby, 01-04 settembre.

*Goldsmith* J.A. (*2001*). *Unsupervised* learning of the morphology of a natural language. *Computational Linguistics*, 27(2), 153-198.

Grefenstette G. (1999). Tokenization. In Van Halteren, H., editor, *Syntactic Wordclass Tagging*, Dordrecht: Kluwer Academic Publishers, pp. 117-133.

Grefenstette G. and Tapanainen P. (1994). What is word, what is a sentence? Problems of text tokenization. *Proc. of 3rd Conference on Computational Lexicography and Text Research* (*COMPLEX'94*), pp. 79-87.

Habert B., Adda G., Adda-Decker M., de Marëuil P.B., Ferrari S., Ferret O., Illouz G. and Paroubek P. (1998). Towards tokenization evaluation. In Rubio, A., Gallardo, N., Castro, R., Tejada, A., editors, *Proc. of First International Conference on Language Resources and Evaluation, volume I*, pp. 427-431.

Harris Z.S. (1955). From phoneme to morpheme. *Language*, 31: 190-222.

Hutchens J. and Alder M. (1998). Finding structure via compression. In *Proc. of CoNLL-98 (2nd International Conference on Computational Natural Language Learning)*, pp. 79-82.

Kempe A. (1999). Experiments in unsupervised entropy-based corpus segmentation. In *Proc. of CoNLL-99 (3nd International Conference on Computational Natural Language Learning)*, pp. 7-13.

Kit C. and Wilks Y. (1999). Unsupervised learning of word boundary with description length gain. In *Proc. of CoNLL-99 (3nd International Conference on Computational Natural Language Learning)*, pp. 1-6.

Li M. and Vitányi P. (1997). *An Introduction to Kolmogorov Complexity and Its Applications (2nd edition)*. New York: Springer-Verlag.

de Marcken C.G. (1996). *Unsupervised Language Acquisition*. Unpublished doctoral dissertation, Massachusetts Institute of Technology.

Mikheev A. (2002). Text Segmentation. In Mitkov, R., editor, *The Oxford Handbook of Computational Linguistics*, Oxford: Oxford University Press, pp. 201-218.

Nevill-Manning C.G. and Witten I.H. (1997). Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7: 67-82.

Olivier D.C. (1968). *Stochastic Grammars and Language Acquisition Mechanisms*. Unpublished doctoral dissertation, Harvard University.

Rissanen J. (1989). *Stochastic Complexity in Statistical Inquiry*. Singapore: World Scientific Publishing Co.

Saffran J.R., Aslin, R.N. and Newport, E.L. (1996). Statistical learning by 8-month old infants. *Science, 274:* 1926-1928.

Shannon C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27: 379-423 and 623-656.

Teahan W.J., Wen Y., McNab R.J. and Witten I.H. (2000). A compression-based algorithm for Chinese word segmentation. *Computational Linguistics*, 26(3): 375-393.

Webster J.J. and Kit C. (1992). Tokenization as the initial phase in NLP. In *Proc. of COLING-92 (14th International Conference on Computational Linguistics)*, pp. 1106-1110.

Wolff J.G. (1977). The discovery of segments in natural language. *British Journal of Psychology*, 68: 97-106.

Wrenn J.O., Stetson P.D. and Johnson S.B. (2007). An unsupervised machine learning approach to segmentation of clinician-entered free text. In *Proc. of AMIA Annual Symposium 2007*, pp. 811-815.

Xanthos A. (2003), Du k-gramme au mot: variation sur un thème distributionnaliste. In *Bulletin de linguistique et des sciences du langage (BIL)*, 21.

Xanthos A. (2004). Combining utterance-boundary and predictability approaches to speech segmentation. In *Proc. of the Psycho-computational Models of Language Acquisition Workshop at COLING 2004*, pp. 93-100.