A Bambara Tonalization System for Word Sense Disambiguation Using Differential Coding, Segmentation and Edit Operation Filtering

Luigi (Yu-Cheng) Liu
ER-TIM, INALCO, 2 rue de Lille, Paris, France
luigi.liu@laposte.net

Damien Nouvel
ER-TIM, INALCO, 2 rue de Lille, Paris, France
damien.nouvel@inalco.fr

Abstract

In many languages such as Bambara or Arabic, tone markers (diacritics) may be written but are actually often omitted. NLP applications are confronted to ambiguities and subsequent difficulties when processing texts. To circumvent this problem, tonalization may be used, as a word sense disambiguation task, relying on context to add diacritics that partially disambiguate words as well as senses. In this paper, we describe our implementation of a Bambara tonalizer that adds tone markers using machine learning (CRFs). To make our tool efficient, we used differential coding, word segmentation and edit operation filtering. We describe our approach that allows tractable machine learning and improves accuracy: our model may be learned within minutes on a 358K-word corpus and reaches 92.3% accuracy.

1 Introduction

Bambara (Bamana, Bamanankan, ISO-369 Bam) is the most widely spoken language of the Manding language group. It is spoken mainly in Mali (and among the considerable Malian diaspora) by 12 to 15 million people. It is not an official language; however it is the major language (besides French) on Malian radio and TV broadcasts, there are newspapers in Bambara, it is broadly used in humanities and in primary schools, and it is taught at several universities around the world. In the Mande language family, Bambara is among the best described: there are numerous works on Bambara language, such as dictionaries (Bailleul, 2007; Dumestre, 2011), description of its grammar (Dumestre, 2003; Vydrin, 1999b,a).

Bambara is a tonal language, with the important drawback that its official orthography does not represent tones. Because several tonalized forms can correspond to some unaccented tokens, it makes word sense more ambiguous in corpus and imposes important challenges to NLP applications. Our goal is to remedy this issue by implementing an automatic tonalizer for Bambara, to improve subsequent NLP processings and facilitate linguistic analysis for Bambara.

2 Bambara Reference Corpus

Our work relies on an annotated corpus, the Bambara Reference Corpus (BRC, in French *Corpus Bambara de Référence*), a linguistically annotated corpus suitable both for linguistic research and for NLP tools development. Building BRC started in 2010 as a project conducted by a small group of specialists in Manding linguistics and computer sciences in Russia. Later on, colleagues from other countries (France, Germany) joined the team to provide more texts, clean the corpus, and automatically preprocess it. The corpus is available online since April 2012, most relevant information about the project may be found online[1] or in publications (Vydrin, 2013, 2014).

Part \ Type	Words (dist.)	Punct.
Non-disamb.	2,160,155 (58,277)	358,659
Disamb.	358,794 (23,875)	61,847

Table 1: Corpus statistics

Table 1 and Figure 1 present overall corpus statistics and characteristics [2]. The corpus consists

[1] http://cormand.huma-num.fr/index.html

[2] Corpus is continuously growing and contains, as of June 2017, 4,113,006 raw words and 903,585 in the disambiguated part.

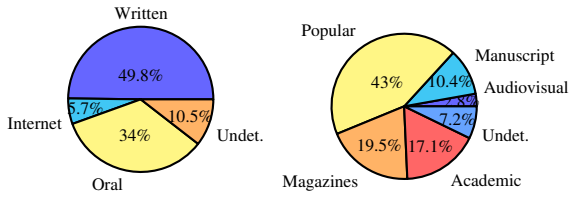

Figure 1: Corpus composition (medium, source)

of two parts: an automatically annotated subcorpus which contains ambiguous interpretations for each token (non-disambiguated), and a manually disambiguated subcorpus which serves as a gold-standard annotated dataset.

2.1 BRC Tools and Resources

The available data within the BRC project includes source text files and an electronic Bambara dictionary. Dedicated tools have been implemented to annotate the corpus. The collection of software tools developed for the BRC (parser, disambiguation interface, various auxiliary scripts) is called Daba (Maslinsky, 2014) and is available online[3]. Among them, a rule-based parser was built to bootstrap the annotation of the corpus.

Thanks to this data and these tools, a gold standard has been created by human annotators proficient in Bambara. This process is described in Figure 2: The parser's output is reviewed, and annotation is done by selecting or editing required information for each wordform.

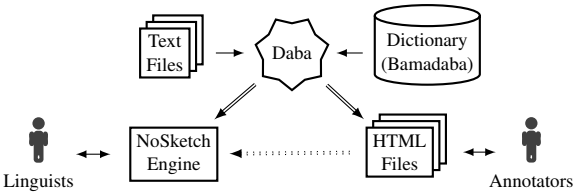

Figure 2: BRC data workflow

2.2 Main Goals for BRC

The project's goal is to disambiguate the whole corpus. Wordform annotation is done for each wordform and for three main features:

- POS Tagging
- Tone Marker Restoration
- Gloss Assignment

[3] https://github.com/maslinych/daba

We are currently working on these tasks by using sequential modeling with various additional techniques. For POS tagging, we use state of the art, Conditional random fields (CRFs; Lafferty et al. 2001) over 23 morpho-syntactic possible tags, our implementation reaches 94% accuracy, which is quite satisfying for such an under-resourced language.

For the tone marker restoration task, we considered using similar methods, but the tag set is much larger: 20,870 distinctive tonal forms are found, which prevents an efficient learning. We have to reduce training complexity by finding an adequate representation that will allow to model tone addition without listing every possible form.

2.3 Tone Markers in Bambara

The word-level tonal information is provided by tonal form or so-called diacritized form in the literature. The tone markers inserted in tonal form are mainly acute accent (´) for low tone, grave accent (`) for high tone, háček (ˇ) for phonetically rising tone and circumflex accent (ˆ) for lexical rising tone. For instance, a non-tonalized token *tugu* can be tonalized as a noun *tùgu* (arms) or a verb *túgu* (to close) that we can find in its quasi-homonym list.

Ideally, the only difference between a tonalized token compared to its non-tonalized form is the presence of tone accents. In reality, annotators also often introduce other modifications, mainly typographic and orthographic correction. As we will see later, this fact makes it necessary to operate filtering on edit operations, in order to properly focus on tonalization operations.

To give an idea of tonal form variety, we propose to compare annotated tokens to their original forms. Related statistics are summarized in Table 2, showing that the probability for modifications other than tonalization during annotation is not negligible : 8.90% modification are related to other modification such as typographic or orthographic correction. We will have to take this into account for our system to focus on tonalization operations.

3 Problem Formulation

Our objective is to design a word sense disambiguation system for Bambara helping to choose the most correct interpretation from the quasi-homonym list provided for each token in the sub-

Operation	Ratio
Tonalization	38.73%
Other	**8.90%**
None	52.35%

Table 2: Distribution of edit operation to annotated forms receiving: only tone markers, other characters, or nothing (i.e., tonalized token is identical to its non-tonalized form)

corpus processed by Daba.

The drawback of modeling sequences of large-scale label set is the expensive computational cost needed to estimate CRF parameters. Indeed, the quasi-Newton method that CRF training employs for solving parameter estimation of CRF model has a computational cost proportional to M^2 where M is the size of label set (Sutton and McCallum, 2010). This makes CRF learning computationally expensive when labels are the set of possible tonalized words.

4 Related Works

4.1 Automatic Diacritization and Sub-word Level Modeling

Simard (1998) presented a method for accent insertion in French using a two-layers Hidden Markov Model (HMM) trained over a POS-tagged corpus while Tufiş and Chiţu (1999) adapted a trigram tagger to this task for Romanian. Focused on word-level modeling, their method requires lexical resources.

Elshafei et al. (2006) use a single-layer HHM for diacritization of Arabic text. Scannell (2011) extended diacritization to uni-codification in African languages by solving it using a Naive Bayes classifier. Their models, trained with trigram features on word and character levels, do not need other resources than an accented corpus, so it is compatible with resource-scarce languages.

Nguyen et al. (2012) implemented a system for accent restoration in Vietnamese based on two different models including the CRFs. The training corpus is preprocessed and annotated to indicate, both at syllable and character levels, the accents to insert to recover diacritic forms. Learning how to infer diacritic forms is indirectly reached by learning their differential representation.

Some hybrid approaches were also proposed for Arabic Diacritization : Said et al. (2013) combined a CRF and a morphological analyzer to improve diacritization accuracy, while Metwally et al. (2016) proposed a three-layer processing composed of a CRF, a HMM and a morphological analyzer.

4.2 Category Decomposition

In the domain of CRF-based tagging, the idea to decompose label set in small pieces to train in order to gain learning performance can be found in (Tellier et al., 2010). They proposed to map total label set to a category tree and experiment to train on labels in a cascade manner or to train independently on each label components which correspond to a sub-category and then recombine them to obtain the total label. They concluded that their method, coined "category decomposition", improves greatly the time-wise efficiency of learning process.

5 Methodology

5.1 Fundamental Definitions

In the following, we will introduce essential elements to our methodology .

Let X, Y, Δ be three discrete random variables: non-tonalized token, tonalized token and code. X and Y take value from Ω (character set with or without diacritics) and Δ is either $\varnothing$ (no modification) or a concatenation of codewords, where a codeword is a triplet containing operation type (insertion or deletion), position for operation and character (if insertion) from Σ:

$$\Sigma \equiv \{+1, -1\} \times \mathbb{N}_{>0} \times \Omega \tag{1}$$

We define E and D, respectively encoder and decoder functions, inverses of each other, such as:

$$\begin{aligned} \Delta &= E(Y; X) \\ Y &= D(\Delta; X) \end{aligned} \tag{2}$$

5.1.1 Entropy Reduction Ratio

r_E is called entropy reduction ratio of E as defined:

$$r_E(\Delta, Y) \equiv \frac{\mathcal{H}(Y)}{\mathcal{H}(\Delta)} \;\; or \;\; \frac{\mathcal{H}(Y)}{\mathcal{H}(E(Y; X))} \tag{3}$$

for any Δ satisfying $\mathcal{H}(\Delta) \neq 0$.

We need to predict tonalization for every non-tonalized token value $X = x$. Instead of predicting on Y, we can predict first on Δ and decode

Δ to obtain Y indirectly by using the second relation in (2). This will allow to greatly reduce the number of possible values of our model: we predict differential codes instead of tonalized words themselves.

In the next subsections, we try to design an efficient encoder E which maximizes r_E so that the resulting code Δ has a reduced set of possible values, it is more easy for a CRF-based modeler to learn on Δ than on Y as explained in the last paragraph of section 3.

5.2 Tonalization as Edit Operations

To maximize r_E (i.e. reduce Δ), we use an algorithm to represent the difference between Y and X by a minimal sequence of basic operations required for recovering Y from X. These basic operations are known as edit operations that we will present in the following.

By edit operation s, we mean a mapping from string a to string b, where a, b are sets of string drawn from the alphabet Ω such that if a is expressed as concatenation of characters as $a_1\, a_2\, ...\, a_{N-1}\, a_N$, then the mapping can be written as

$$b = s(a; \sigma) \tag{4}$$

where $\sigma = (m, p, c) \in \Sigma$ is a parameter tuple, m is an operation type indicator, p is a position parameter, c is a character involved by the operation.

And the mapping output b will be

$$b = \begin{cases} a_1 a_2 ... a_{p-1}\, c\, a_p ... a_N, & m = +1 \\ a_1 a_2 ... a_{p-1}\, a_{p+1} ... a_N, & m = -1 \end{cases} \tag{5}$$

We can see in (5), that when $m = +1$, s represents an insertion operation which consists in adding c just before the p-th character[4] in a, while in the case of $m = -1$, s denotes a deletion operation which removes the p-th character from a.

5.2.1 Use of Wagner-Fischer Algorithm

For any pair of strings (x, y), the Wagner-Fischer algorithm[5] in (Wagner and Fischer, 1974) allows to obtain a minimal sequence $S = (\sigma_1, \sigma_2, ..., \sigma_{M-1})$ such that:

$$\begin{aligned} A_{i+1} &= s(A_i; \sigma_i + (0, l_i, \varnothing)) \\ l_{i+1} &= l_i + m(\sigma_i) \\ A_0 &= x, A_M = y, l_0 = 0 \end{aligned} \tag{6}$$

Where $0 \leq M, 1 \leq i \leq M-1$, position parameters p of σ_i denoted by $p(\sigma_i)$ form an increasing series with respect to index i:

$$p(\sigma_i) \leq p(\sigma_{i+1}) \tag{7}$$

For the edit operations acting on the same position p, insertions must precede[6] deletion as described below:

$$m(\sigma_i) \geq m(\sigma_{i+1}),\ \text{if}\ p(\sigma_i) = p(\sigma_{i+1}) \tag{8}$$

Because it does not make sense to delete a character of a string twice, we assume that, for a given character position, only one deletion may occur:

Thus, $h(\sigma_i)$ below forms an increasing series

$$h(\sigma_i) \equiv 2 \cdot p(\sigma_i) + 0.5 \times [1 - m(\sigma_i)] \tag{9}$$

5.2.2 Encoder

If we set encoder $E(y; x)$ as an application of Wagner-Fischer algorithm on (x, y), and δ as the concatenation of elements of S like

$$\delta = \sigma_1 \sigma_2 ... \sigma_{M-1} \tag{10}$$

Then $E(y; x)$ is specified so that the first relation in (2) is satisfied.

5.2.3 Decoder

Moreover, if we choose $D(\delta; x)$ as an implementation of (6) applied on x with δ as parameter, this specifies a way to use x and the code δ to recovery y, and define a decoder.

To show that the entropy is effectively reduced, we will present experiments in section 8 for evaluation of entropy reduction ratio of r_E.

5.3 Segmentation

Segment decomposition of a string x is a mapping from a string x to strings called segments $x^{(1)} x^{(2)} x^{(3)} ... x^{(L)}$ so that the concatenation of the latter is equal to the original string x.

We can then divide the resulting code δ in L code segments $\delta^{(1)} \delta^{(2)}\, ...\, \delta^{(L)}$ described as:

$$\delta = \delta^{(1)} \delta^{(2)}\, ...\, \delta^{(L)} \tag{11}$$

$\delta^{(i)}$ is the code segment (i.e. a sub-sequence of code δ) associated with $x^{(i)}$:

$$\delta^{(i)} = \sigma_{\min(S_i)} ... \sigma_{\max(S_i)} \tag{12}$$

[4] Or just after the $(p-1)$-th character, when p = N

[5] In this article, we apply Wagner-Fischer algorithm in its special case where there are only 2 available edit operations against 3 edit operations including the substitution as in its general case.

[6] Or we can also state that deletions precede insertions. In this case, the second relation of (6) will be different for that decoder to work properly. So this was our design choice.

where S_i is the set which contains the indices of all edit operations acting on segment $x^{(i)}$.

Furthermore, we define shifted version of $\delta^{(i)}$ as below

$$\delta'^{(i)} = \sigma'_{\min(S_i)} ... \sigma'_{\max(S_i)} \tag{13}$$

where $\sigma'_{\min(S_i)} = \sigma_{\min(S_i)} + (0, -e_i, \varnothing)$, $e_i = \sum_{k=1}^{i-1} |x_k|$ position offset. It can be shown that y can be obtained by concatenation [7] of decoding results of D on shifted code segment $\delta'^{(i)}$ as below

$$y = D(\delta'^{(1)}; x^{(1)}) D(\delta'^{(2)}; x^{(2)}) ... D(\delta'^{(L)}; x^{(L)}) \tag{14}$$

Using the proposed segmentation allows to get the shifted code segment $\delta'^{(i)}$ associated with segment $x^{(i)}$. For obtaining y, it suffices to individually decode code segments $\delta'^{(i)}$ using the same decoder defined in (2). Hence, an encoder - decoder pair is indirectly specified as:

$$\begin{aligned} \delta'^{(i)} &= E'(y, x^{(i)}) \\ y^{(i)} &= D(\delta'^{(i)}, x^{(i)}) \end{aligned} \tag{15}$$

The segmentation technique allows a CRF-based system to learn $\delta'^{(i)}$ individually instead of a total code δ at once. The relation in (14) can then be used to recover y from predicted code segments $\delta'^{(i)}$ produced by CRF taggers.

To show the effect of segmentation on entropy reduction ratio $r_{E'}$, we will evaluate experiments in section 8 with different segmentation settings such as syllabification and fixed-width segmentation.

5.4 Edit Operation Filtering

As previously explained, segmentation on x leads to splitting code δ in several code segments $\delta^{(i)}$ (or its shifted version $\delta'^{(i)}$) because it regroups edit operations $\sigma_j = (m_j, p_j, c_j)$ inside δ by position parameter p_j.

Edit operation filtering allows to split each code segment $\delta^{(i)}$ in some subgroups by operation type m_i and character involved by operation c_i. But as opposed to segmentation, in the filtering process, some edit operations are dropped because of their irrelevance regarding our tonalization goal for Bambara.

[7] Alternatively, it is also possible to obtain the integral code δ from $\delta'^{(i)}$ using relations (11), (12), (13) and to get the underlying tonalized token y by direct decoding on the integral code δ.

First, we recall that the order of edit operations called σ'_k inside the sequence $\delta'^{(i)}$ is partially determined by their elements. It can be shown, more generally that the order restriction in (7), (8) leads directly to

$$p'_k < p'_l \implies k < l \tag{16}$$

$$m'_k > m'_l \wedge p'_k = p'_l \implies k < l \tag{17}$$

where $m'_k \equiv m(\sigma'_k)$, $p'_k \equiv p(\sigma'_k), k \neq l$.

Thus, the order between any two edit operations in $\delta'^{(i)}$, called σ'_k, σ'_l is free only in the case of multiple insertion acting on the same position of a string, i.e. $m'_k = m'_l$ = +1, $p'_k = p'_l$.

5.4.1 Tone Marker Filtering

The aim of tone marker filtering is to remove edit operations irrelevant to tonalization. It consists in removing all insertions of characters which are not considered as tone markers, and keeping only the first of all insertions (also only the first one of all tone deletions) operating on the same position (because neither multiple insertions nor multiple deletions are supposed to be present in pure tonalization).

In addition, the filter contributes to reduce the code so that the order of edit operations in filter output becomes totally determined by edit operation arguments (i.e. p and m) via relations (16)(17). Therefore, the indexing function h defined in 9 forms a strictly increasing series.

5.4.2 Edit Operation Decomposition

We define an edit operation dispatcher F_m as a mapping from input code δ_{in} to its sub-sequence described as:

$$F_m(\delta_{in}; k) = \sigma_{\min(V_k(\delta_{in}))} ... \sigma_{\max(V_k(\delta_{in}))} \tag{18}$$

where $V_k(\delta)$ is the set which contains the indices of all edit operations in δ of type k, $k \in \{-1, +1\}$

For recall, the tone marker filter produces a code result in which $h(\sigma_i), i > 0$ forms a strictly increasing sequence. This property implies that there exists an inverse mapping from $\{F_m(\delta_{in}; k)\}_{k \in \{-1,+1\}}$ to δ_{in} if δ_{in} is a filtered result. We call this underlying mapping edit operation assembler.

6 System Architecture

In this section, we present the architecture of our system for automatic tone marker insertion for Bambara text by detailing its functional blocks and internal data flow.

6.1 Training Stage

At training stage, as shown in Figure 3, the tonalization system uses an encoder to represent the difference of the tonalized token value y relative to its original non-tonalized token value x by the code δ. The code δ as well as the non-tonalized token x are then fed into the code segmenter which breaks down δ in small shifted code segments $\delta'^{(i)}$. The tone marker filter takes the shifted code segments, selects tone markers and sends its output to a dispatcher which distinguishes insertion and deletion operations.

In the following, two CRF models are trained individually on relation $(x, \delta_+'^{(i)})$ and on relation $(x, \delta_-'^{(i)})$ in the reference corpus.

6.2 Tonalization Stage

In tone recovery stage, the two trained CRF models are used to predict the two parts of shifted code segment $\delta_+'^{(i)}$, $\delta_-'^{(i)}$ from tonalized token x. Then they are fed to, as shown in Figure 4, an edit operation assembler to get predicted shifted code segment $\delta'^{(i)}$; at the same time, the non-tonalized token x is re-segmented to get $x^{(i)}$. Then, the non-tonalized segment $x^{(i)}$ as well as the shifted code segment $\delta'^{(i)}$ are sent to be decoded to obtain tonalized segment $y^{(i)}$ in decoder output. In the end, the decoded tonalized segments $y^{(i)}$ are reunited in the code assembler to give the estimated integral tonalized token value y.

7 Running examples

To illustrate more concretely how our system works at training stage, we propose in the following, two running examples: the first one shows how our system models a pure tonalization; the second one shows how the tone marker filter could be helpful when some typographic modifications are introduced in the tonalized form.

7.1 Example 1 (pure tonalization)

In this example, the system models the tonalization from *lakali* to *lákàlí*. Hence, x, y are set as below:

$$x = lakali$$
$$y = lákàlí$$

The tone encoder compares x and y, then outputs the differential representation as tonal code δ below:

$$\delta = [(+1, 2, \acute{}), (+1, 4, \grave{}), (+1, 6, \acute{})]$$

In the above, the 3 codewords that δ contains represent the 3 tone markers inserted into x to model for this tonalization.

$$\delta^{(1)} = [(+1, 2, \acute{})]$$
$$\delta^{(2)} = [(+1, 4, \grave{})]$$
$$\delta^{(3)} = [(+1, 6, \acute{})]$$

The syllabification is examplified below on non-tonalized and tonalized form:

$$x^{(1)} = la, y^{(1)} = lá$$
$$x^{(2)} = ka, y^{(2)} = kà$$
$$x^{(3)} = li, y^{(3)} = lí$$

We observe that the 3 tone markers are all inserted after the second character for each of 3 syllables. Therefore, we got 3 shifted versions of code segments operating at position 2 as below.

$$\delta_+'^{(1)} = \delta'^{(1)} = [(+1, 2, \acute{})]$$
$$\delta_+'^{(2)} = \delta'^{(2)} = [(+1, 2, \grave{})]$$
$$\delta_+'^{(3)} = \delta'^{(3)} = [(+1, 2, \acute{})]$$

As in this example, only the tone marker insertions are to be modeled, the tone marker filter, edit operation decomposition unit do not affect system result at training stage.

7.2 Example 2 (noisy tonalization)

The system models the tonalization from *taanikasegin* to *táa-ká-ségin*. Hence, x, y are set as below:

$$x = taanikasegin$$
$$y = táa - ká - ségin$$

In the following, we show the syllabification result of y presented with its related to non-tonalized

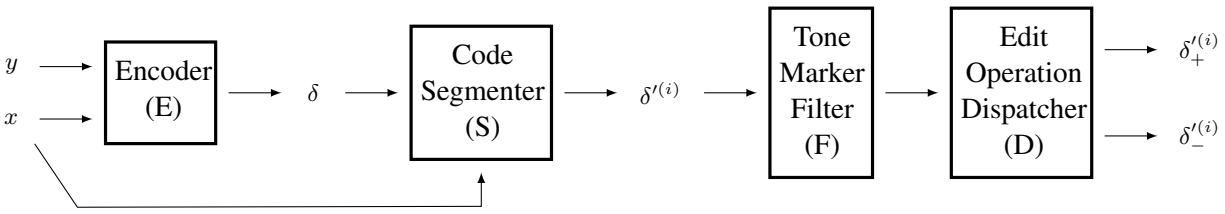

Figure 3: Block diagram for the proposed Bambara tonalization system at training stage

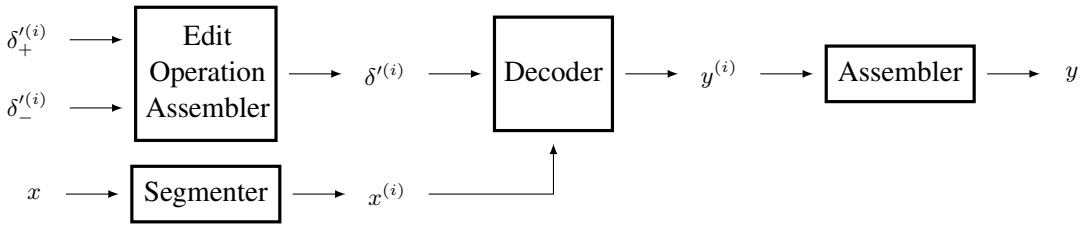

Figure 4: Block diagram for the proposed Bambara tonalization system at tonalization stage

form.

$$x^{(1)} = taani, y^{(1)} = táa-$$
$$x^{(2)} = ka, y^{(2)} = ká-$$
$$x^{(3)} = se, y^{(3)} = sé$$
$$x^{(4)} = y^{(4)} = gin$$

Subsequently, 7 codewords are obtained from differential tone encoding, then departed in 4 code segments corresponding to 4 syllables:

$$\delta^{(1)} = [(+1, 2, \acute{}), (+1, 3, -), (-1, 4, n), (-1, 5, i)]$$
$$\delta^{(2)} = [(+1, 7, \acute{}), (+1, 7, -)]$$
$$\delta^{(3)} = [(+1, 9, \acute{})]$$
$$\delta^{(4)} = \varnothing$$

As a remark, the first syllable in the above is the most concerned by noise : one dash symbol insertion and 2 character removals are observed. The tone marker filter removes all character deletions and insertions in code segments. Therefore, tonalization is modeled by only tone marker insertions as below:

$$\delta'^{(1)}_+ = \delta'^{(1)} = [(+1, 2, \acute{})]$$
$$\delta'^{(2)}_+ = \delta'^{(2)} = [(+1, 2, \acute{})]$$
$$\delta'^{(3)}_+ = \delta'^{(3)} = [(+1, 2, \acute{})]$$
$$\delta'^{(4)}_+ = \delta'^{(4)} = \varnothing$$

Here, we found a tonalization representation similar to the one from the preceding example: insertion of a tone marker at position 2 for some syllables.

8 Experiment

8.1 Experiment Setup

The disambiguated corpus of BRC which contains tonalized tokens is chosen as training data. The train data then is split into training and evaluation data sets with ratio $p : (1 - p)$. By default, p is set to 50% for efficiency reason.

For CRF implementation, we use CRFSuite as open-source library with l-BFGS algorithm specified as training method, Viterbi algorithm as inference method (Okazaki, 2007) .

The segmentation mode is specified by the integer w described in the following : $w = -1$ indicates a syllabification (as obtained by BRC morphological parser), $w = 0$ for no segmentation and $w > 0$ signals a w-width regular segmentation[8].

We denote different configurations of our system by their functional layout at training stage. For specifying the functional layout, the following symbols D, F, S, E are used to indicate respectively four constituent blocks of the system : D for edit operation decomposition block, F for tone marker filter block, S for segmentation block, E for encoder block, and $\circ$ denotes an inter-block connection. For example, $S \circ E$ represents an encoder followed by a segmenter as system.

[8] A regular segmenter forms a segment of every w successive characters, from left to right (i.e. in direction of writing of Bambara), in its input string. By exception, the last segment at output contains the rest of the string which has not yet been segmented so that we allow it to be equal or shorter than a segment of w characters.

8.2 Feature Engineering

The feature patterns we created are the same for all CRF models. The features are generated for each segment[9] for tokens in the training data $x^{(i)}$.

For each segment, features indicate:

- At segment (sub-word) level
  - Previous, current and next segment
  - Substring containing all vowels for current segment
  - Position of current segment (integer) relative to beginning or ending of word
  - Typography of segment: all capital letters, presence of numbers or punctuation marks
- At word level
  - Previous, current and next words
  - Prefix and suffix for previous, current and next words
  - Position of current word (integer) relative to beginning or ending of sentence

8.3 Evaluation Measure

As shown by the example of the section 7.2, the tonalized token y can itself be noisy. On the other hand, we are particularly interested in errors introduced by CRF-based in predicting $\delta_+'^{(i)}$,$\delta_-'^{(i)}$ from x. Consequently, our accuracy evaluation is based on the comparison of two tokens described as below:

- A gold standard according to tonalization filtering (i.e. typographic modification are not expected)
- The token tonalized by our system, resulting from applying $\delta_+'^{(i)}$,$\delta_-'^{(i)}$ (predicted by the CRFs) to the segments $x^{(i)}$

8.4 Experiment Results

8.4.1 Coding and Segmentation

The entropy for the tonalized token Y that we calculated on the disambiguated part of BRC is 8.73 bits of entropy for 20,870 distinctive tonal forms. As shown in Table 3, the tonalization representation produced by tone encoder Δ has only 3.38 bits of entropy (1,273 distinctive codes). In other word, the entropy is reduced by a factor of 2.58 by only using the differential tone encoder. If segmentation is also used, shorter segments lead to greater entropy reduction. The extreme case is to segment by character ($w = 1$), leading to an entropy reduction ratio of 4.82, and implies character-level processing.

8.4.2 Segmentation and Operation Filtering

Table 4 presents an experiment on the impact of system configuration (related to Figure 4) and segmentation mode on tonalization accuracy and training time of our system. For comparison, the accuracy of a baseline based on majority voting method is also provided in the last row.

We note first that segmentation improves accuracy. Syllabification as well as regular segmentation with a fixed width close to averaged syllable length (3.24 characters) gives the best result in accuracy (0.92). Regarding system configuration, accuracy is generally preserved over the four different configurations except of when applying operation decomposition without filtering. This can be explained by the fact that decomposition is not totally recoverable as mentioned in section 5.4.2 for data which is not processed by tone marker filter, it can introduce some errors within assembler at tonalization phase and degrades its accuracy.

Training time is dramatically reduced by segmentation and by edit operation decomposition. This may sometimes be at the cost of accuracy, but syllabification or fixed segmentation with width $w = 2$ is a nice trade-off, preserving accuracy and keeping training time acceptable.

Those experiments allow to state that segmentation is gainful both for accuracy and training time (see last column). Decomposition and filtering do not have a tremendous impact on accuracy, but are unavoidable to be able to train models on the whole dataset in reasonable time.

8.4.3 Effect of Train Size and Error Analysis

Figure 5 shows how the training set size influences the tonalization accuracy and training time. We can see the progress in accuracy with respect to training set size is less considerable when $p \geq 50\%$ than when $p < 50\%$. The training time increases dramatically with respect to training set size.

Table 5 shows distribution of errors which occur in automatic tone insertion (not in tone deletion). The error occurs due to a bad prediction in

[9] If the edit operation decomposition is used, as we use two CRF models, features are generated on segment components (for insertion and for deletion) individually

Entropy \ Width w	-1 (Syllabe)	1	2	3	4	5	6	0
$H(\delta'^{(i)})$	**2.67**	**1.81**	**2.56**	2.86	3.11	3.23	3.29	3.38
$r_E(\delta'^{(i)}, y^{(i)})$	**3.27**	**4.82**	**3.41**	3.05	2.80	2.70	2.65	2.58

Table 3: Encoder entropy $H(\Delta)$ and reduction ratio $r_E(\delta'^{(i)}, y^{(i)})$

Systems \ Width w	-1 (Syll.)	1	2	3	4	5	6	0
D ∘ F ∘ S ∘ E	**0.923**	0.912	**0.923**	**0.923**	0.918	0.911	0.904	0.893
time	19.88	17.62	13.17	15.67	19.62	32.40	44.25	261.83
F ∘ S ∘ E	**0.924**	0.916	**0.923**	**0.923**	0.918	0.911	0.905	0.894
time	57.68	19.42	20.72	35.70	62.85	121.08	168.27	2455.80
D ∘ S ∘ E	0.921	0.911	0.921	**0.922**	0.916	0.910	0.903	0.892
time	28.28	21.27	19.72	29.17	39.90	57.35	70.15	793.85
S ∘ E	**0.923**	0.915	**0.922**	**0.922**	0.917	0.910	0.904	0.893
time	101.63	25.52	42.03	235.35	378.37	169.55	318.23	2683.72
Majority vote	**0.843**							

Table 4: Accuracy for our system trained with four different system configurations and eight segmentation modes (p = 50%)

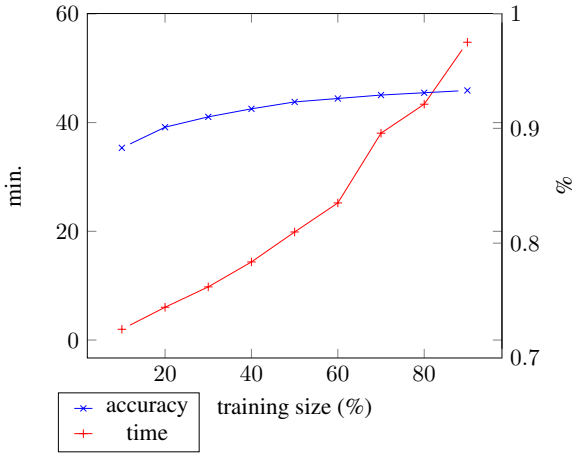

Figure 5: Accuracy and time of training (configured as $D \circ F \circ S \circ E$ using syllabification) with respect to different training size 90%-10%

Error Type	Ratio
Tone Only	**58.52%**
Position Only	1.17%
Tone and Position	0.023%
Silence	**40.08%**

Table 5: Error distribution by type for insertion operations with p = 50%, system = $D \circ F \circ S \circ E$

character to insert but correct prediction in character position is prevalent (58.52%) and the silence (40.08%) which is a false negative case is also very frequent. Prediction error only in character position is as weak (1.17%) as error rate (0.023%) for both character position and symbol error. This shows that our system correctly predicts positions (mainly at beginning of words) and that most of the errors are due to the difficulty to predict if a tone has to be added, and which one.

Actual \ Predicted	´	`	ˆ	ˇ
´	0.9541	**0.0438**	0.0021	0.0000
`	**0.0841**	0.9141	0.0015	0.0003
ˆ	0.0035	**0.0322**	0.9643	0.0000
ˇ	0.0000	**0.0952**	0.0000	0.9048

Table 6: Confusion matrix on prediction of tone markers

Table 6 presents the confusion matrix of the error occurring due to bad prediction in character mentioned previously in Table 5. It shows that these errors are mainly due to the confusion between low tone (´) and high tone (`), which are the two most frequent markers in BRC.

9 Conclusion

In this paper, we presented our Bambara tonalization system for automatic detection of tone markers in Bambara. Our experiments show that using segmentation both increases tonalization accuracy and greatly reduces training time. Our differential encoder reduces entropy of labels to be predicted, making CRF learning efficient and allowing to implement a tone marker filter and edit operation decomposition unit within the tonalization process. The tone marker filter plays the role of normalizator of tonalized token to learn in training phase and in normalizing the token, it leads to reduce training time. The edit operation decomposition unit which follows the filter split the tokens in insertion and deletion of tone markers, allows to accelerate furthermore the training time reduction without altering tonalization accuracy.

Acknowledgments

We thank National Institute for Oriental Languages and Civilizations which supports this work as part of its research project called MANTAL. We acknowledge, in this regard, Valentin Vydrin, Kirill Maslinsky, Davy Auffret, Elvis Mboning and Arthur Provenier for thier contributions, which make possible our work.

References

Charles Bailleul. 2007. *Dictionnaire Bambara-Franais (3e dition corrige)*. Bamako : Donniya.

Grard Dumestre. 2003. *Grammaire fondamentale du bambara*. Paris : Karthala.

Grard Dumestre. 2011. *Dictionnaire bambara-franais suivi dun index abrg franais-bambara*. Paris : Karthala.

Moustafa Elshafei, Husni Al-Muhtaseb, and Mansour Alghamdi. 2006. Statistical methods for automatic diacritization of arabic text. In *The Saudi 18th National Computer Conference. Riyadh*, volume 18, pages 301–306.

John Lafferty, Andrew McCallum, and Fernando CN Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Kirill Maslinsky. 2014. Daba: a model and tools for manding corpora. In *TALN-RECITAL 2014 Workshop TALAf 2014 : Traitement Automatique des Langues Africaines (TALAf 2014: African Language Processing)*, pages 114–122. Association pour le Traitement Automatique des Langues.

Aya S Metwally, Mohsen A Rashwan, and Amir F Atiya. 2016. A multi-layered approach for arabic text diacritization. In *Cloud Computing and Big Data Analysis (ICCCBDA), 2016 IEEE International Conference on*, pages 389–393. IEEE.

Minh Trung Nguyen, Quoc Nhan Nguyen, and Hong Phuong Nguyen. 2012. Vietnamese diacritics restoration as sequential tagging. In *Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on*, pages 1–6. IEEE.

Naoaki Okazaki. 2007. Crfsuite: a fast implementation of conditional random fields (crfs).

Ahmed Said, Mohamed El-Sharqwi, Achraf Chalabi, and Eslam Kamal. 2013. A hybrid approach for arabic diacritization. In *International Conference on Application of Natural Language to Information Systems*, pages 53–64. Springer.

Kevin P Scannell. 2011. Statistical unicodification of african languages. *Language resources and evaluation*, 45(3):375–386.

Michel Simard. 1998. Automatic insertion of accents in french text. In *EMNLP*, pages 27–35.

Charles Sutton and Andrew McCallum. 2010. An introduction to conditional random fields. *arXiv preprint arXiv:1011.4088*.

Isabelle Tellier, Iris Eshkol, Samer Taalab, and Jean-Philippe Prost. 2010. Pos-tagging for oral texts with crf and category decomposition. *Research in Computing Science*, 46:79–90.

Dan Tufiş and Adrian Chiţu. 1999. Automatic diacritics insertion in romanian texts. In *Proceedings of COMPLEX99 International Conference on Computational Lexicography*.

Valentin Vydrin. 1999a. Les parties du discours en bambara : un essai de bilan. 35:73–93.

Valentin Vydrin. 1999b. Manding-english dictionary (maninka, bamana). In *St. Petersburg: Dmitry Bulanin Publishing House*, volume 1.

Valentin Vydrin. 2013. Bamana reference corpus (brc). *Procedia-Social and Behavioral Sciences*, 95:75–80.

Valentin Vydrin. 2014. Projet des corpus écrits des langues manding: le bambara, le maninka. In *Traitement Automatique du Langage Naturel 2014*.

Robert A Wagner and Michael J Fischer. 1974. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173.