Automatic Insertion of Accents in French Text

Michel Simard
Laboratoire de recherche appliquée en linguistique informatique (RALI)
Université de Montréal
simardm@iro.UMontreal.CA

Abstract

Automatic accent insertion (*AAI*) is the problem of re-inserting accents (diacritics) into a text where they are missing. Unaccented French texts are still quite common in electronic media, as a result of a long history of character encoding problems and the lack of well-established conventions for typing accented characters on computer keyboards. We present an AAI method for French, based on a stochastic language model. This method was implemented into a program and *C* library of functions, which are now commercially available. Our experiments show that French text processed with this program contains less than one accent error per 130 words. We also show how our AAI method can be used to do on-the-fly accent insertions within a word-processing environment, which makes it possible to write in French without having to type accents. A prototype of such a system was integrated into the Emacs editor, and is now available to all students and employees of the Université de Montréal's computer science department.

1 Introduction

Even in this era of flashy, high-speed multimedia information, unaccented French texts (i.e texts without diacritics) are still routinely encountered in electronic media. Two factors account for this: first, the computer field has long suffered from a lack of sufficiently widespread standards for encoding accented characters, which has resulted in a plethora of problems in the electronic transfer and processing of French texts. Even now, it is not uncommon for one of the software links in an E-mail distribution chain to deliberately remove accents in order to avoid subsequent problems. Secondly, when using a computer keyboard that is not specifically designed for French, keying in French accented characters can turn out to be a laborious activity. This is a matter of both standards and ergonomics. As a result, a large number of French-speaking users systematically avoid using accented characters, at least in informal communication.

If this situation remains tolerable in practice, it is essentially because it is extremely rare that the absence of accents renders a French text incomprehensible to the human reader. Cases of ambiguity do nonetheless occur: for instance, "Ce chantier ferme a cause des emeutes" could be interpreted as "Ce chantier *ferme à cause* des émeutes" ("This work-site is closing because of the riots") or "Ce chantier *fermé a causé* des émeutes" ("This closed work-site [more naturally put, this work-site closure] has caused riots"). From a linguistic point of view, the lack of accents in French simply increases the relative degree of ambiguity inherent in the language. At worst, it slows down reading and proves awkward, much as a text written entirely in capital letters might do.

The fact remains, however, that while unaccented French text may be tolerated under certain circumstances, it is not acceptable in common usage, especially in the case of printed documents. Furthermore, unaccented texts pose serious problems for automatic processing: NLP-based applications such as information retrieval, information extraction, machine translation, human-machine conversation, speech synthesis, as well as many others, will usually require that French texts be properly accented to begin with.

Actually, for human readers, unaccented texts is probably the most benign of a more general class of ill treatments to which French texts are subjected. For example, it is not uncommon for older programs that are not "8-bit clean" to "strip" the eighth bit of each character, thus irreversibly mapping French characters onto the basic ASCII set. When this treatment is applied to an ISO-Latin text, '*é*' becomes '*i*', '*è*' becomes '*h*', etc. Other programs will simply delete accented characters, or replace them with a unique character, such as a question mark. The texts that result rapidly become unreadable.

All of the above factors prompted the initial interest in methods of *automatic accent insertion* (or *AAI*). Of course, as standards such as *Unicode* (multilingual character-coding standard) and *MIME* (multipurpose Internet mail extensions) gain ground, the accent legacy problem slowly disappears. The problem of typing accents, however, is likely to remain. For this reason, we have become interested in meth-

ods that would perform automatic accent insertion *on-the-fly*, in real time. It appears to us that such a tool would be a valuable addition to any word-processing environment, equally useful for native and non-native speakers of French.

In what follows, we first present a general automatic accent insertion method, based on a stochastic language model. This method was implemented into a program called Réacc, which is now commercially available through Alis Technologies[1]. We then examine how this method can be adapted to perform accent insertions on-the-fly within a word-processing environment. As we go along, we describe the various experiments we designed to evaluate the performance of the system in different contexts, and present the results obtained. Finally, we briefly describe how a prototype "on-the-fly accentuation" (*OTFA*) system was implemented within the Emacs text-editor.

Although our research focuses on unaccented French texts, we believe that our approach could be adapted to other languages that use diacritical marks, as well as to other types of text corruption, such as those mentioned above. The AAI problem and the solutions that we propose are also related to the more general problems of word-sense disambiguation and spelling and grammar checking.

2 Basic Automatic Accent Insertion

In its simplest form, the automatic accent insertion problem can be formulated this way: we are given as input an unaccented French text, in the form of a sequence of unaccented words $w_1 w_2 \ldots w_n$. To every one of these input words w_i may correspond any number of valid words (accented or not) $w_{i1} \ldots w_{im}$: our task is to disambiguate each word, i.e. to select the correct words w_{ik_i} at every position in the text, in order to produce a properly accented text.

An examination of the problem reveals that the vast majority (approximately 85%) of the words in French texts carry no accents at all, and that the correct form of more than half of the remaining words can be deduced deterministically on the basis of the unaccented form. Consequently, with the use of a good dictionary, accents can be restored to an unaccented text with a success rate of nearly 95% (i.e., an error in accentuation will occur in approximately every 20 words). The problems that remain at this point mostly revolve around *ambiguous* unaccented words, i.e. words to which more than one valid form may correspond, whether accented or not[2].

Obviously, for many such ambiguities in French, a simple solution is to systematically select the most frequent alternative. For instance, the most frequent word in most French texts is usually the preposition *de*, which turns out to be ambiguous, because there is also a French word *dé*, meaning either *dice* or *thimble*. If we simply ignore the latter form, we are likely to produce the correct form over 99% of the time, even in texts related to gambling and sewing! This general strategy can be implemented by determining *a priori* the most frequent alternative for each set of ambiguous words in a dictionary, by means of frequency statistics extracted from a corpus of properly accented French text. Using this simple method, we achieve a success rate of approximately 97%, i.e. roughly one error per 35 words.

Clearly, to attain better performances than these, an automatic accent insertion system will need to examine the context within which a given ambiguous word appears, and then resort to some form of linguistic knowledge. *Statistical language models* seem to be particularly well fit to this task, because they provide us with quantitative means of comparing alternatives.

We propose an automatic accent insertion (*AAI*) method that proceeds in two steps.

1. **Hypotheses generation**: identify for each input word the list of valid alternatives to which it may correspond;
2. **Candidate Selection**: select the best candidate in each list of hypotheses.

This is illustrated in Figure 1.

2.1 Hypotheses Generation

Hypotheses generation produces, for each word w_i of the input, a list of possible words $w_{i1} \ldots w_{im}$ to which it may correspond. For example, the form *pousse* may correspond to either *pousse* or *poussé*; *cote* to *cote*, *côte*, *coté* or *côté*; the only valid form for *francais* is *français* (with a cedilla), and *ordinateur* is its own unique correct form. In theory, nothing precludes generating *invalid* as well as valid hypotheses at this stage: for instance, for *cote*, also generate *cötè* and *çote*. But to limit the number of possibilities that the system must consider, hypotheses are produced using a list of known French word-forms, indexed on their unaccented version. On the other hand, when the hypotheses generator encounters word-forms that it does not know, it simply reproduces them verbatim.

2.2 Candidate Selection

Once lists of hypotheses have been identified for each input word, the best candidate of each list must be identified. For this, we rely on a stochastic language model, which can assign a score to any sequence of words, corresponding to the probability that the model generate this sequence. Given an input sequence of words $w_1 w_2 \ldots w_n$, and for each word w_i

[1] Alis Technologies: http://www.alis.com

[2] As we will see later on, other problems are caused by *unknown* words , i.e. words for which *no* valid forms are known.

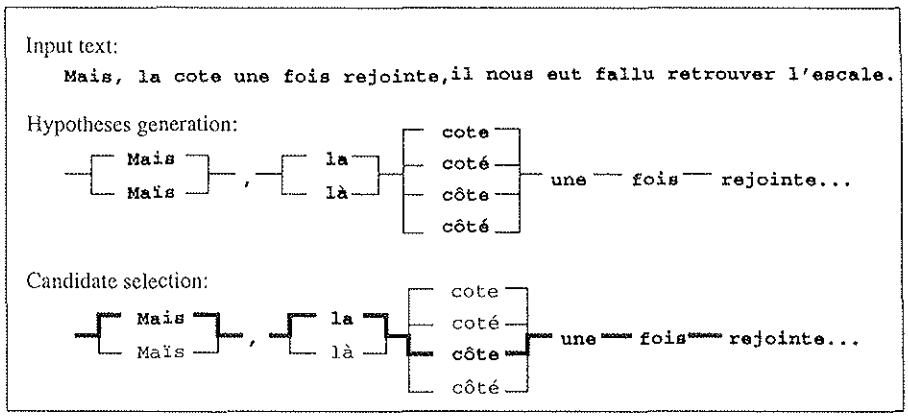

Figure 1: Automatic accent insertion method

in the sequence, a list of hypotheses $(w_{i1}, \ldots, w_{im})$, our goal can be reformulated as finding the sequence of hypotheses $w_{1k_1} w_{2k_2} \ldots w_{nk_n}$ that maximizes the overall likelihood of the output sequence.

The stochastic model we use is a *Hidden Markov Model* (HMM), within which a text is viewed as the result of two distinct stochastic processes. The first process generates a sequence of abstract symbols. In our case, these symbols correspond to *morpho-syntactic tags*, e.g. *"common noun, masculine-singular"*, *"verb, present indicative form, third person plural"*. In an N-tag HMM, the production of a tag depends on the $N-1$ preceding tags, so that the probability of observing a given tag t_i in a given context follows a conditional distribution $P(t_i|t_{i-N} \ldots t_{i-1})$.

Then, for each tag in this first sequence, a second stochastic process generates a second symbol: in our case, these symbols correspond to actual words in the language.

The parameters that define the model are:

- $P(t_i|h_{i-1})$: the probability of observing tag t_i, given the previous $N-1$ tags (h_{i-1} designates the series of $N-1$ tags ending at position $i-1$);
- $P(w_i|t_i)$: the probability of observing word w_i given the underlying tag t_i.

Given these parameters, the probability of generating some sequence of words $w = w_1 w_2 \ldots w_n$ can be evaluated. If T is the *tag alphabet*, and T^n denotes the set of all possible sequences of n tags of T, then:

$$P(w) = \sum_{t \in T^n} \prod_{i=1}^{n} P(t_i|h_{i-1}) P(w_i|t_i)$$

The direct calculation of this equation requires a number of calculation that is exponential in the length of the sequence. However, there exists an algorithm that computes the value of $P(w)$ in polynomial time (Rabiner and Juang, 1986).

To find the sequence of hypotheses that maximizes the probability of the text, each individual combination of hypotheses is examined. Because the number of possible combinations grows exponentially with the length of the text, we will want to segment the text into smaller pieces, whose probabilities can be maximized individually. Sentences are usually considered to be syntactically independent, and so we may assume that maximizing the probability of each sentence will yield the same result as maximizing the whole text. Even within sentences, it is sometimes possible to find subsegments that are "relatively" independent of one another. Typically, the inner punctuation of sentences (semicolons, commas, etc.) separates segments that are likely to be independent of one another. In the absence of inner punctuation, it is still possible to segment a sentence around regions of "low ambiguity".

Our AAI method relies on a heuristic segmentation method, which cuts up each sentence into a number of segments, such that the number of combinations of hypotheses to examine in each segment does not exceed a certain fixed threshold, while minimizing dependencies between segments. This segmentation strategy effectively guarantees that the accent-insertion can be done in polynomial time. But we sometimes end up segmenting the text at "sub-optimal" locations. This will have consequences on performance, as we will see in the next section.

Segments are processed in a left-to-right fashion. In practice, we have realized that one way of minimizing the negative impact of sub-optimal segmentations is to prepend to each segment the last few words of the previous segment, as output by the AAI system. This seems to have the effect of "priming" the model. The prepended words are then simply dropped when the

final result is pieced together.

2.3 Implementation

The method presented in the previous section was implemented in a program called `Réacc`. This program, given a hypotheses generator, the parameters of a HMM and an input, unaccented French text, produces an accented version of that text on the output.

The hypotheses generator we used was produced from a list of over 250 000 valid French words, extracted from our French morpho-syntactic electronic dictionary. Such a large dictionary is probably overkill, and in fact, it may even be the case that it uselessly slows down processing, by proposing extremely rare (although probably valid) words. (The only francophones we met that had heard of a *lé* were crossword puzzle addicts.)

The language model used is a 2-tag HMM, based on a set of approximately 350 morpho-syntactic tags. The parameters of the HMM were first estimated by direct frequency counts on a 60 000 words, hand-tagged extract of the Canadian Hansard. The parameters were then refined, using Baum-Welch reestimation (Baum, 1972), on a 3 million word (untagged) corpus consisting of equal parts of Hansards, Canadian National Defense documents and French press revues (*Radio-France International*).

2.4 Performance Evaluation

One of the interesting properties of the AAI problem is that the performance assessment of a given program is a very straightforward affair: all we need is a corpus of correctly accented French text, and a "de-accentuation" program. Performance can be measured by counting the number of words that differ in the original text and its re-accented counterpart.

For the purpose of our evaluation, we used a test corpus made up of various types of text. It contains Hansard, National Defense and RFI documents (distinct from those used in training), but also United Nations documents, court transcripts, computer manuals as well as some literary texts. The whole corpus contains 57 966 words (as counted by the standard `wc` UNIX program).

Apart from the hypotheses generator and the language model parameters, a number of parameters affect the performance of the program. The most important of these is the maximum number of combinations per subsegment, that it used in the segmentation heuristic. In what follows, we refer to this parameter as S. The results obtained for different values of S are presented in Table 1. All tests were done on a SparcSTATION 10 computer, with 32 MB of memory.

A cursory look at the results reveals that there is much to be gained by allowing the system to work on longer segments. However, beyond a certain limit, the quality of the results tends to level off, while the running time increases radically. Depending on the context of application of the program and the resources available, it would seem that acceptable results can be obtained with S set at around 16 or 32. In this setting, the system will process anywhere between 10 000 and 20 000 words per minute.

It is interesting to look at where `Réacc` goes wrong. Table 2 provides a rough classification of accent-restoration errors made by the program on our test corpus with S set at 16. The largest category of accentuation errors includes a rather liberal grouping of errors that have a common feature: they are the result of an incorrect choice pertaining to an acute accent on a final *e*. In most cases (although not all), this corresponds to an ambiguity between a finite and participle forms of a verb, e.g. *aime* as opposed to *aimé*. The next group of errors are those that stem from inadequacies in the hypotheses generator – i.e. cases in which the generator simply does not know the correct accented form. In most cases (nearly half), proper nouns are involved, but, especially in more technical texts, there are also many abbreviations, non-French words and neologisms (e.g. *réaménagement*, *séropositivité*). The next category concerns a unique word pair: the preposition *à*, and *a*, the third person singular present indicative form of the verb *avoir*.

2.5 Related Work

El-Bèze et al. (1994) present an AAI method that is very similar to ours. It also proceeds in two steps: hypotheses generation, which is based on a list of valid words, and candidate selection, which also relies on a Hidden Markov Model. The main difference between their method and ours is how the HMM is used to score competing hypotheses. While we segment the text into "independent segments" and maximize the probability of these segments, their program processes the text from left to right, using a fixed width "sliding window":

- For each word w_i, the hypotheses generator produces a list of possible *word/tag* alternatives: $(w_{i1}, t_{i1}), \ldots, (w_{ik}, t_{ik})$;
- Candidate Selection proceeds by selecting a specific pair (w_{ij}, t_{ij}) at each position; the goal is to find the sequence of *word/tag* pairs whose probability is maximum according to the model:

$$\prod_{i=1}^{n} P(w_{ij_i}|t_{ij_i}) P(t_{ij_i}|t_{i-1j_{i-1}}, t_{i-2j_{i-2}})$$

- To avoid combinatorial problems, instead of computing this product for all possible sequences, the system finds at each position i in the sequence the pair (w_{ij}, t_{ij}) that *locally* maximizes that part

Max. no. of combinations per segment (S)	Running time (seconds)	Total number of errors (words)	Average distance between errors (words)
2	68	821	70
4	85	560	103
8	132	466	124
16	169	441	130
32	277	429	134
64	429	425	136
128	731	420	137

Table 1: Results of AAI Experiments on 58K-word Test Corpus

Type of error	Number of occurrences	Percentage
-e VS. *-é* ending	171	38.8%
Unknown words	111	25.2%
a VS. *à*	69	15.7%
Other	90	20.4%
Total	441	100.0%

Table 2: Classification of Accent Restoration Errors ($S = 16$)

of the global computation within which it is involved:

$$P_i \times P_{i+1} \times P_{i+2}$$

where $P_i = P(w_{ij_i}|t_{ij_i})P(t_{ij_i}|t_{i-1j_{i-1}}, t_{i-2j_{i-2}})$.

- These computations proceed from left to right, so that the optimal tag found for position i will be used in the computation of the optimal *word/tag* pairs at positions $i + 1$ and $i + 2$.

The experimental results reported in El-Bèze et al. (1994) indicate success levels slightly superior to ours. This may be explained in part by the use of a better language model (their HMM is three-tag, ours is two-tag). It must be said, however, that their test-corpus was relatively small (in all, a little over 8000 words), and that the performances varied wildly from text to text, with average distances between errors varying between 100 and 600 words.

A method which exploits different sources of information in the candidate selection task is described in Yarowsky (1994b): this system relies on local context (e.g., words within a 2- or 4-word window around the current word), global context (e.g. a 40-word window), part-of-speech of surrounding words, etc. These are combined within a unifying framework known as *decision lists*. Within this framework, the system bases its decision for each individual candidate selection on the single most reliable piece of evidence.

Although the work described in Yarowsky (1994b) does address the problem of French automatic accentuation, it mostly focuses on the Spanish language. Furthermore, the evaluation focuses on specific ambiguities, from which it is impossible to get a global performance measure. As a result, it is unfortunately not currently possible to compare these findings with ours in a quantitative way.

In Yarowsky (1994a), the author compares his method with one based on the stochastic part-of-speech tagger of Church (1988), a method which obviously has a number of points in common with ours. In Mr Yarowsky's experiments, this method is clearly outperformed by the one based on decision lists. This is most apparent in situations where competing hypotheses are "syntactically interchangeable": pairs of words with identical morpho-syntactic features, or with differences that have no direct syntactic effects, e.g. present/preterite verb tenses. Such ambiguities are better resolved with non-local context, such as temporal indicators. As it happens, however, while such situations are very common in Spanish, they are rare in French. Furthermore, Mr Yarowsky's language model was admittedly quite weak: in the absence of a hand-tagged training corpus, he based his model on an *ad hoc* set of tags.

3 On-the-fly Automatic Accent Insertion

As mentioned earlier, the existence of unaccented French texts can in part be explained by the lack of a standard keying convention for French accents: conventions vary from computer to computer, from keyboard to keyboard, sometimes even from program to program. Many users type French texts without accents simply because they are unfamiliar with the conventions in a particular environment, or because these conventions are too complicated (e.g. hitting three keys in sequence to type a single accented character).

Clearly, in some situations, automatic accent insertion offers a simple solution to this problem: type the text without accents, run an AAI program on the text, and revise the output for accentuation mistakes. Of course, such a solution, if acceptable for one-time production of short texts, is not very practical in general. If a text is subjected to a number of editions and re-editions, or if it is produced cooperatively by several authors working in different environments, then it may need to go through a series of local re-accentuations. This process, if managed by hand, is error-prone and, in the end, probably more laborious than typing the accents by hand.

If, however, the accents are automatically inserted on-the-fly, as the user types the text, then accent revision and corrections can also be done as the text is typed. If such an *on-the-fly accentuation* (*OTFA*) system is capable of producing acceptable results in real-time, it may become a realistic alternative to the manual insertion of accents. In what follows, we examine how this may be done.

3.1 Method

How does OTFA differ from the basic AAI problem? In Section 2, the input was considered to be a static and (hopefully) complete text. In OTFA, the text is dynamic: it changes with every edit operation performed by the user. Therefore, the OTFA method that is conceptually the simplest is to re-compute the accentuation of the whole text after each edit, i.e. repeatedly apply to the entire text an AAI method such as that proposed earlier.

Of course, such a method is impractical, mainly because it will likely be computationally excessively expensive. It is also overkill, because changes in one region of the text are unlikely to affect the accentuation of the text in more or less distant regions. In fact, if we use the AAI method of Section 2, changes in one location will have no effects outside the sentence within which the edit occurs, because sentences are all treated independently. Because sentences are themselves sub-segmented, it is tempting to think that the effect of a given edit will be even further restricted, to the segment of the sentence within which it takes place. This, however, is not generally true, firstly because an edit is likely to affect the sub-segmentation process itself, and also because changes in one segment can have cascading effects on the subsequent segments, as the last words of each segment are prefixed to the following segment as additional context.

So a more practical solution is to process only the sentence within which the latest edit occurred. There are still problems with this approach, however. While the user is editing a sentence, chances are that at any given time, this sentence is "incomplete". Furthermore, although modern text-editors allow insertions and deletions to be performed in any order and at any position of the text, in a normal text-editing context, given the natural tendency of humans to write in a beginning-to-end fashion, the majority of the edits in a French text will be left-to-right insertions at the end of sentences. This means that at any given time, the text to the left of the latest edit is likely to constitute relevant context for the AAI task, while the text to the right is likely not to be relevant. In fact, taking this text into consideration could very well mislead the AAI process, as it may belong to a completely different sentence.

This suggests a further refinement: after each edit, process only that part of the current sentence that lies to the left of the location where the edit took place.

Also, it seems that there is no real need to take any action *while* the user is modifying a given word, and that it would be wiser to wait until all edits on that particular word are finished before processing it. By doing so, we will not only save computational time, we will also avoid annoying the user with irrelevant accentuations on "partial" words. Notice, however, that detecting the exact moment when the user has "finished" typing or modifying a word can be a tricky business. We will deal with this question in Section 3.4.

One of the potential benefits of performing accentuation on-the-fly, as opposed to *a posteriori* AAI, is that the user can correct accent errors as they happen. In turn, because accentuation errors sometimes cascade, such on-the-fly corrections may help the AAI "stay on the right track".

If we want to capitalize on user-corrections, we will need to:

1. *somehow distinguish "corrections" from other types of edits*: the reason is that we don't want to override the user's decisions when performing AAI. This question will also be dealt with when we discuss implementation details (Section 3.4).
2. *limit the scope of the AAIs to a small number of words around the location of the last edit*: the user can only correct the error that he *sees*; in theory, the effect of AAI after each edit is limited to the current sentence, but sentences come in all sizes. If a given "round" of AAI affects text too far away from the site of the last edit, which is usually also the focus of the user's attention, then he is likely not to notice that change. For this reason, it seems reasonable to restrict the actual scope of the AAI process to just a few words: intuitively, three or four words would be reasonable. Note that this doesn't imply restricting the amount of *context* that we provide the AAI with, but only limiting the size of the region that it is allowed to modify.

To summarize, the OTFA method that we propose essentially follows these lines:

- OTFA is performed by repeatedly applying an AAI method (such as that of Section 2) on the text.
- AAI rounds are triggered every time the user finishes editing a word.
- The scope of AAI (which we call the *AAI window*) is limited to a fixed number of words to the left of the last word edited.
- If this can be useful to the AAI process, more context can be given, in the form of additional words belonging to the same sentence to the left of the AAI window (what we call the *context window*).

3.2 Performance Evaluation

The ultimate goal of OTFA is to facilitate the editing of French texts. Therefore, it would be logical to evaluate the performance of an OTFA system in those terms. Unfortunately, the "ease of typing" is a notion that is hard to quantify. In theory, typing speed would seem to be the most objective criterion. But measuring performance using such a criterion would obviously require setting up a complex experimental protocol. On the other hand, the number and nature of parameters involved prohibits a "theoretical" evaluation in these terms.

What we can reliably evaluate, however, is the absolute performance of an OTFA system, in terms of the number of accentuation errors, for a given editing "session". Such a measure gives us an intuitive idea of the impact of the OTFA system on the "ease of typing".

We conducted a number of experiments along this line, to evaluate how an OTFA system based on the AAI system of Section 2 would perform. All experiments were done by simulation, using the same corpus that was used in Section 2.4. The editing "session" we simulated followed a very simple scenario: the user types the whole test corpus, from beginning to end, without typing accents, without making errors, and without correcting those made by the OTFA system.

As was the case with the Réacc program, several parameters affect the quality of the results and the computation time required. The only parameter that is specific to our OTFA method, however, is the size of the AAI window. This parameter, which we refer to as W, is measured in words. We conducted distinct experiments with various values for W, the results of which are summarized in Table 3. In all of these experiments, the segmentation factor S was set at 16.

The first conclusion that we can draw from Table 3 is that there is much to be gained in using an AAI window of more than one word: setting $W = 2$ allows to cut down the number of errors by almost 60%. Performance quickly levels off, however, so that near-optimal results are obtained with a three- or four-word window. This is encouraging, because it seems reasonable to assume that the user can effectively monitor a window of that size, and therefore detect accentuation errors when they occur.

Another point that is very encouraging, and perhaps surprising, is that with $W = 3$, the performance of our OTFA system rivals with that of the basic AAI experiments reported in Section 2.4. One possible explanation is that because the OTFA works with only a small number of words at each round (i.e. only the words in the AAI window), the system never has more than $S = 16$ combinations to examine, and therefore never needs to segment sentences into smaller pieces. In the end, both ways of proceeding are probably more or less equivalent, although more experimentation would be required to determine this for sure. The major difference, of course, is that since OTFA recomputes accentuation with every new word, its computational cost is accordingly higher. However, as seen in Section 2.4, our AAI system can process 20 000 words per minute. Since very few typists can enter more than 100 words per minute, even a straightforward OTFA implementation should be able to handle the required computations in real-time.

3.3 User-feedback

We mentioned earlier that one of the expected benefits of OTFA, as opposed to applying AAI on a text *a posteriori*, is that the user can spot accent errors as soon as they happen, and correct them right away. In fact, we believe that this form of *user-feedback* can even be further exploited, to improve the performance of the system itself. As pointed out in Section 2.4, about a quarter of AAI errors are caused by *unknown words*, i.e. words in the correctly accented version of the text which are unknown to the hypotheses generator. This suggests an easy way of exploiting user-feedback: systematically add to the hypotheses generator all user-corrected words whose form is unknown.

In principle, if we add such a mechanism to our OTFA system, and if the user corrects the AAI errors as soon as they happen, unknown words will be lexicalized right after their first appearance, and the system should only make one error per unknown word. In preliminary experiments with this idea, the average distance between errors passed from 138 to 156 words, a reduction of almost 12% on the total number of errors. Our test corpus being heterogeneous by design, unknown words do not repeat very often. We suspect that even better improvements would be observed on homogeneous texts of similar size.

This idea of exploiting user-feedback to modify the parameters of the OTFA dynamically can actually be pushed further. One of the current problems with

AAI window (W)	Total errors (words)	Average distance between errors (words)
1	1125	52
2	461	126
3	420	138
4	417	139
8	417	139
16	417	139

Table 3: OTFA Simulation Results

our OTFA system is its sometimes annoying tendency to systematically select the most frequent alternative when confronted with syntactically interchangeable words. For example, the two French words *cote* and *côte* have similar morpho-syntactic features (common noun, feminine singular) and so, from a grammatical point of view, are totally interchangeable. It so happens, however, that in the language model's training corpus, the second form, which is highly polysemous, is much more frequent. Therefore, the OTFA will systematically produce that form rather than the other. If the user of the system is writing about the stock market for example, he is likely to want to use the first form *cote*, and therefore to react negatively to the system's insistence on putting a circumflex accent where none should appear.

To solve this problem, some form of *dynamic language modeling* is required. We have begun experimenting with an approach initially proposed by Kuhn and Mori (1990) to solve a similar problem in speech recognition applications. Essentially, they suggest using local context to estimate the parameters of a unigram Markov model, and to use this model in conjunction with the static HMM to evaluate competing alternatives. Preliminary results with this approach are encouraging, although much work remains to be done.

3.4 Implementation

As mentioned earlier, the AAI method presented in Section 2 has been implemented as a program and *C* function library. Based on this implementation, a prototype OTFA system was developed and integrated to the *Emacs* text-editor. Although *Emacs* is not generally viewed as a true word-processing environment, it was a natural choice for prototyping because of its openness and extendibility.

In our implementation, the user of *Emacs* has access to a special editing mode called *Réacc-mode* (technically speaking, a *minor-mode*). When in this mode, the user has access to all the usual editing functions: he can move the cursor around, insert, delete, etc. The main difference with the normal "fundamental" mode is that now, accents are automatically inserted as words are typed, without the user having to explicitly type them.

The implementation follows the general lines of the OTFA method presented in Section 3.1: every time a new word is inserted, the system identifies the AAI window, submits the words that fall within this window to the AAI system, and replaces the content of the window with the newly accented words.

In practice, Emacs and the AAI program run as separate processes, and communicate asynchronously: when a new word is typed, Emacs sends the AAI window to the AAI process, along with other relevant information (context, position, etc.), and returns the control to the user. The AAI program processes the "accentuation request" in the background, and sends the results back to Emacs as soon as they are ready. When this happens, Emacs interrupts whatever it was doing, and replaces the original contents of the AAI window with the newly arrived words. This way, user-interaction is not significantly slowed down by the AAI process, because time-consuming computations typically take place during the editor's idle time, between keystrokes.

It is the editing process' responsibility to initiate AAI rounds, and therefore to determine when a new word has been typed. After experimenting with various strategies, we opted for a relatively simple method, based on the possibility to mark individual characters of the text with specific "properties" in Emacs. When words are processed by the AAI program and re-inserted into the text, they are systematically marked as *auto-accented*. By contrast, characters typed by the user do not carry this mark. Every time the user types a space or newline character, we examine the word immediately preceding the cursor: if all its characters are unmarked, then a new AAI round must be initiated.

We mentioned earlier that it was important for an OTFA system not to override the user's decisions. Two situations are particularly important to consider: when the user manually types an accent within a new word, and when the user corrects the accentuation of a word. In both cases, it is undesirable that the OTFA modify the words in question. The character mark-

ing capabilities of Emacs are also used to detect these situations. The first case (new word with accents) will be identified easily by the presence of accented characters within an unmarked word. The second situation (accent corrections) is more difficult to detect, but in general, a mix of marked and unmarked characters within a single word is a good indicator that corrections have taken place.

When these two situations occur, not only do we not initiate an AAI round, we also inhibit any further re-accentuations on these words, by marking their characters as *user-validated.* Words bearing this mark will never be touched by AAI. This type of marking is not limited to user-inserted accents and user-corrections: when the user turns *Réacc-mode* on, all existing text is initially marked that way. Later on, when AAI rounds are initiated and the system locates the AAI window, all text outside this window is also marked as *user-validated.* This way of proceeding, while allowing the OTFA system to do its work during simple text insertions, limits the possibility of "unpleasant surprises" when more complex interactions take place (deletions, corrections, cut-and-paste operations, etc.).

4 Conclusion

We have presented a method for automatically inserting accents into French text, based on a stochastic language model. This method was implemented into a program and *C* library of functions, which are commercially available from Alis Technologies. We have also shown how this method can be used to do on-the-fly accent insertions within a word-processing environment. A prototype OTFA system was also implemented and integrated into the Emacs editor.

Text processed with our system contains less than one accent error per 130 words on average, regardless of whether the system is used on its own or within an OTFA environment. On a Sun SparcSTATION 10 computer, with 32 MB, the system will process approximately 20 000 words per minute. Within the Emacs OTFA prototype, because AAI is performed asynchronously, the performance of the editor itself is not affected, and accents are inserted faster than this typist can type[3].

The program has been made available to students and employees of the Université de Montréal's computer science department, and initial feedback has been positive. We are currently examining the possibility of integrating our OTFA method to a "real" word-processor, such as Microsoft Word.

Acknowledgments

I am greatly indebted to Guy Lapalme, George Foster and Pierre Isabelle for their invaluable advice and constructive comments, as well as to Elliott Macklovitch, for helping me translate my thoughts into readable English. Many thanks also go to all the members of the RALI who contributed to the development of the Réacc system, as well as François Yergeau of Alis Technologies.

[3]These performance figures were obtained with a segmentation factor S set at 16.

References

L. E. Baum. 1972. An Inequality and Associated Maximization Technique in Statistical Estimations of Probabilistic Functions of Markov Processes. *Inequalities*, 3:1–8.

Kenneth W. Church. 1988. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proceedings of the Second Conference on Applied Natural Language Processing.* ACL.

Marc El-Bèze, Bernard Mérialdo, Bénédicte Rozeron, and Anne-Marie Derouault. 1994. Accentuation automatique de textes par des méthodes probabilistes. *Technique et sciences informatiques*, 13(6):797–815.

Roland Kuhn and Renato De Mori. 1990. A Cache-based Natural Language Model for Speech Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(6).

L. R. Rabiner and B. H. Juang. 1986. An Introduction to Hidden Markov Models. *IEEE ASSP Magazine*, pages 4–16, jan.

David Yarowsky. 1994a. A Comparison of Corpus-based Techniques for Restoring Accents in Spanish and French Texts. In *Proceedings of the Second Annual Workshop on Very Large Corpora*, Kyoto, Japan.

David Yarowsky. 1994b. Decision Lists for Lexical Ambiguity Resolution: Applications to Accent Restoration in Spanish and French. In *Proceedings of ACL-94*, Las Cruces, New Mexico.