Natural Language Engineering **1** (1): 1–23. Printed in the United Kingdom

Real-time Automatic Insertion of Accents in French Text

MICHEL SIMARD

ALEXANDRE DESLAURIERS

Laboratoire de recherche appliquée en linguistique informatique (RALI)
Université de Montréal
[simardm,deslaura]@iro.UMontreal.CA

(*Received 7 April 2000*)

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. An AAI method for French is presented, based on a statistical language model. Next, it is shown how this AAI method can be used to do real-time accent insertions within a word-processing environment, making it possible to type in French without having to type accents. Various mechanisms are proposed to improve the performance of real-time AAI, by exploiting on-line corrections made by the user. Experiments show that, on average, such a system produces less than one accentuation error for every 200 words typed.

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 is awkward and slows down reading, 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 text poses 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 text 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 (AAI). Of course, as standards such as *Unicode* (multilingual character-coding standard) and *MIME* (multipurpose Internet mail extensions) gain ground, the accent legacy problem is slowly disappearing. The problem of typing accents, however, is likely to remain. For this reason, we have become interested in methods that would perform AAI 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 AAI method, based on a statistical language model (Section 2).We then examine how this method can be adapted to perform accent insertions on-the-fly within a word-processing environment, and describe a possible implementation (Section 3). Finally, we explore various ways of exploiting user-feedback to improve the performance of the system (Section 4).

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 Automatic Accent Insertion (AAI)

In its simplest form, the AAI problem can be formulated this way: we are given as input an unaccented French text, in the form of a sequence of unaccented words

$u_1 u_2 \ldots u_n$. To every one of these input words u_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 index k_i of the correct word 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 word tokens in French texts carry no accents at all, and that the correct form of more than half of the remaining tokens 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[1].

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 AAI 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.

2.1 AAI Method

We propose an AAI method that proceeds in two steps: first, identify for each input word the list of valid alternatives to which it may correspond (what we call *hypothesis generation*); then select the best candidate in each list of hypotheses (*candidate selection*). This is illustrated in Figure 1.

Hypothesis generation produces, for each word-form u_i of the input, a list of the 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*. All that would be needed

[1] As we will see later on, other problems are caused by unknown words , i.e. words for which the valid accented form is not known.

Input text:					
u_1 = Mais	u_2 =la	u_3 =cote	u_4 =une	u_5 =fois	...
Hypothesis generation:					
$w_{1,1}$ = Mais	$w_{2,1}$ = la	$w_{3,1}$ = cote	$w_{4,1}$ = une	$w_{5,1}$ = fois	...
$w_{1,2}$ = Maïs	$w_{2,2}$ = là	$w_{3,2}$ = coté			
		$w_{3,3}$ = côte			
		$w_{3,4}$ = côté			
Candidate selection:					
$k_1 = 1$	$k_2 = 1$	$k_3 = 3$	$k_4 = 1$	$k_5 = 1$	
$w_{1,1}$ = Mais	$w_{2,1}$ = la	$w_{3,3}$ = côte	$w_{4,1}$ = une	$w_{5,1}$ = fois	...

Table 1. *Automatic accent insertion method*

to perform hypothesis generation then would be a list of possible accentuations for each letter of the alphabet. Instead, 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. When the hypothesis generator encounters word-forms that it does not know, it simply reproduces them verbatim.

Once lists of hypotheses have been identified for each input word, the best candidate of each list must be selected. For this, we rely on a statistical 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 $u_1 u_2 \ldots u_n$, and for each word u_i 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 statistical model we use is a hidden Markov model (HMM), based on parts-of-speech, such as those proposed by Church (Church, 1988) and DeRose (DeRose, 1988) for part-of-speech tagging. Within this kind of model, a text is viewed as the result of two distinct stochastic processes. The first process generates a sequence of abstract symbols, informally called "tags". In our case, each of these tags corresponds to the name of a part-of-speech, augmented with some morpho-syntactic features, *e.g. "common noun, masculine-singular"*, *"verb, present indicative form, third person plural"*. In an N-tag HMM, the production of a tag depends only 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 word-forms in the language.

Hence, 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.

The actual values of individual parameters are learned automatically, by examining large quantities of running text (the "training corpus"). Through a procedure known as "Baum-Welch re-estimation" (Baum, 1972), the parameters are gradually refined, so as to maximize the likelihood of the training corpus with regard to the

model. Whether or not this procedure converges to the overall maximum depends on the initial values of the parameters; the better our initial guess at these values, the better our chances of reaching maximum likelihood and, ultimately, the better our model.

Short of knowing the right initial values, the best approach is to estimate them by observing the frequency of each event in a sample of text in which the hidden layer is visible, i.e. a corpus of text manually annotated with POS tags. Another useful resource in this regard is a dictionary, from which we can determine which parts-of-speech are possible for each word-form, and which are not; otherwise, for words that do not appear in the training material (i.e., most words), we will have to assume *a priori* that all tags are equally likely.

Once the model has been trained, the probability of observing some sequence of words $w = w_1 w_2 \ldots w_n$ can be evaluated. If T is the set of tags, 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 number of possible tag sequences in T^n grows exponentially with n, but fortunately, there exists an algorithm for computing the value of $P(w)$ in polynomial time (see Rabiner and Juang (Rabiner and Juang, 1986), for example).

To find the sequence of accentuation 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 later.

Segments are processed in a left-to-right fashion. In practice, one way of minimizing the negative impact of sub-optimal segmentations is to provide the last few words of the previous segment, as re-accented by the AAI system, as additional context to the current segment. Hence, the system is actually processing overlapping segments of text. Of course, the additional words preprended to each segment are dropped when the final result is pieced together.

2.2 AAI Implementation

The method presented in the previous section was implemented in a program called `Réacc`. This program, given a hypothesis generator, the parameters of a HMM and an input, unaccented French text, produces an accented version of that text on the output[2].

The hypothesis 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.)

For the language model, we used Foster's implementation of HMMs (Foster, 1991) to create a 2-tag model, based on a set of approximately 350 morpho-syntactic tags. The vocabulary and possible POS's for each word-form were first determined using the same French morpho-syntactic electronic dictionary as above. Then, the parameters of the HMM were estimated by direct frequency counts on a 60 000 words, hand-tagged extract of the Canadian Hansard, and finally re-estimated on a 3 million word (untagged) corpus consisting of equal parts of Hansards, Canadian National Defense documents and French press revues (*Radio-France International*).

2.3 AAI 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 three texts: a 35K-word court transcript, a 59K-word extract of Hansard (distinct from the extract used to train the HMM) and a novel by Jules Verne ("De la Terre à la Lune"; 53K words). These texts were chosen because they represent types of document which are respectively highly, moderately and remotely related to the language model's training material. Each document was split in two more or less equal parts; the first halves were kept for various training tasks, while the second halves were used for testing.

Apart from the hypothesis 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 is used in the segmentation heuristic. In what follows, we refer to this parameter as S. The effect of different values of S on the training portion of our texts is shown in Figure 1. All tests were done on a Sparc Ultra 1 computer, with 256 MB of memory.

[2] `Réacc` is commercially available, through Alis Technologies (`http://www.alis.com`), and an on-line demo of is available at `http://www-rali.iro.umontreal.ca/Reacc`

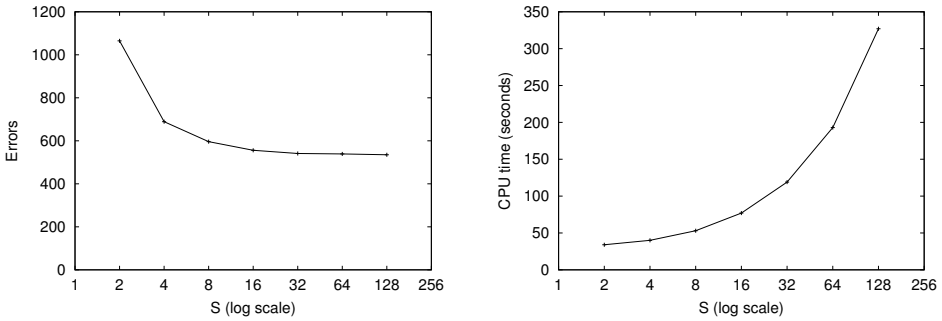

Fig. 1. Effect of AAI segment-size on training corpus.

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 40 000 and 70 000 words per minute.

Table 2 shows the performance of the system on the test portion of our corpus, with $S = 16$. As we can see, the system performs quite differently on different types of text: the average distance between accentuation errors varies between 120 and 200 words (in these results, as in all that follow, what counts as a "word" is any contiguous sequence of alphabetic characters). The overall average is 165 words – this is roughly a dozen sentences[3].

Table 2. *AAI test results.*

Text	Total errors (words)	Average distance between errors (words)
Court transcript	103	167
Hansard	140	208
Jules Verne	233	120
total/avg.	441	165

It is interesting to look at where `Réacc` goes wrong. Table 3 provides a rough classification of accent-restoration errors made by the program on our test corpus

[3] Here, as in all similar tables, the overall average distance between errors is just the average of the figures obtained on specific test files; in other words, we compensate for the fact that some test files are larger than others.

Table 3. *Classification of accent restoration errors.*

Type of error	Number of occurrences	Percentage
-e VS. *-é* ending	155	32.6%
a VS. *à*	145	30.5%
ou VS. *où*	44	9.2%
la VS. *là*	33	6.9%
Unknown words	25	5.3%
Other	74	15.5%

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 categories concern specific grammatical words: the preposition *à*, and *a*, the third person singular present indicative form of the verb *avoir*; coordination *ou* and adverb *où*; and article *la* and adverb *là*.

Finally, a small group of errors stem from inadequacies in the hypothesis generator – i.e. cases in which the generator simply does not know the correct accented form. In most cases, proper nouns are involved, but there are also a few cases of non-French words. In our experience, such a small proportion (5.3%) for this category of errors is atypical for real-life texts – the real figure is usually somewhere around 20-25%. In fact, this is what we observe on the court transcript (20.4%). How is it then that in the comparatively larger sample of Hansard, we find only one such error? The explanation is actually simple: at the time these tests were carried out, our lab had been working with the Hansard data for over a decade. Over the years, most of the vocabulary specific to this corpus (including proper nouns) had been manually added to our lexical databases. Hence the virtual absence of unknown words in this text.

As for the Jules Verne novel, being over a hundred years old, most of its vocabulary is standard dictionary stuffing, and because the story is set in the United States, the few proper names it contains are typically American and devoid of any accents.

2.4 Related Work

El-Bèze et al. (El-Bèze et al., 1994) present an AAI method that is very similar to ours. It also proceeds in two steps: hypothesis generation, which is based on a list of valid words, and candidate selection, which also relies on a POS-based 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 input word u_i, the hypothesis 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 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 by El-Bèze et al. 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 widely 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 by Yarowsky (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 Yarowsky does address the problem of French automatic accentuation, his work is centered 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 a different article (Yarowsky, 1994a), the author compares his method with one based on the statistical part-of-speech tagger of Church (Church, 1988), a method which obviously has a number of points in common with ours. In these 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/preterit 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, 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 Real-time Automatic Accent Insertion (RTAAI)

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 this 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 *real-time automatic accent insertion* (RTAAI) may be done.

3.1 RTAAI Method

How does RTAAI differ from the basic AAI problem? In Section 2, the input was considered to be a static and (hopefully) complete text. In RTAAI, the text is dynamic: it changes with every edit operation performed by the user. Therefore, the RTAAI 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 in Section 2.1.

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 3.2.

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 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 3.2).

Also, because the user can only correct the error that he sees, we will want to limit the scope of the AAI to a small number of words around the location of the last edit. 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 RTAAI method that we propose essentially follows these lines:

- RTAAI 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 RTAAI 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 RTAAI 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* can activate 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 RTAAI 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 RTAAI 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 (detecting user-corrections is also important for taking user-feedback into account, as will be discussed in Section 4). In both cases, it is undesirable that the RTAAI modify the words in question. The character marking 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. 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 RTAAI 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.*).

3.3 RTAAI Evaluation

The ultimate goal of RTAAI is to facilitate the editing of French texts. Therefore, it would be logical to evaluate the performance of an RTAAI 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.

What we can reliably evaluate, however, is the absolute performance of an RTAAI 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 RTAAI system on the "ease of typing", because it suggests how often the user will need to correct accentuation mistakes.

We conducted a number of experiments along this line, to evaluate how an RTAAI 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 2.3. 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 RTAAI 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 RTAAI 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 on the training part of the corpus, with various values for W, the results of which appear in Table 4. In all of these experiments, the segmentation factor S was set at 16.

The conclusion that we can draw from these results is that there is much to be gained in using an AAI window of more than two word. Performance quickly levels

Table 4. *RTAAI simulation results on training corpus, for various AAI window sizes.*

AAI window (W)	Total errors (words)	Average distance between errors (words)
1	599	136
2	596	137
3	547	152
4	546	153
8	551	151
16	550	151

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.

Table 5 shows the system's performance on the test corpus, with $W = 3$. Another pleasant surprise awaits us here: the RTAAI system actually performs better than the basic AAI on which it is based. One possible explanation is that because the RTAAI 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. The major difference, of course, is that since RTAAI recomputes accentuation with every new word, its computational cost is accordingly higher. However, as seen in Section 2.3, our AAI system can process 50 000 words per minute. Since very few typists can enter more than 100 words per minute, even a straightforward RTAAI implementation should be able to handle the required computations in real-time.

Table 5. *RTAAI results on test corpus.*

Text	Total errors (words)	Average distance between errors (words)
Court transcript	101	170.6
Hansard	121	241.6
Jules Verne	222	126.9
total/avg.	444	179.7

Table 6. *RTAAI results on test corpus, with user-corrections.*

Text	Total errors (words)	Average distance between errors (words)	Improvement over baseline (% errors)
Court transcript	99	174.0	2.0%
Hansard	121	241.6	0.0%
Jules Verne	221	127.5	0.5%
total/avg.	441	181.0	0.7%

4 Exploiting User-feedback

One of the expected benefits of RTAAI, 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 actually help improve the performance of the system itself. The intuition is that there must be situations where one accentuation error leads to another, so that we observe some sort of "domino effect". This conjecture is easily verified by modifying the simulated scenario used in Section 3.3 for testing RTAAI: Table 6 shows what happens when the user systematically corrects words that have accentuation errors as soon as they leave the AAI window.

While the net result is only a slight reduction in the total number of errors, we observe on closer examination that the actual errors are not exactly the same. What this means is that, while correcting accent errors as they happen may not improve performance, it does affect the behavior of the system. In this section, we describe our attempts at exploiting the user's contribution, and the results we obtained.

4.1 Dynamic Lexicalization

As pointed out in Section 2.3, a number of AAI errors are caused by unknown words, i.e. words in the correctly accented version of the text which are unknown to the hypothesis generator. This suggests a simple way of exploiting user-feedback: dynamically augment the contents of the hypothesis generator with user-corrected words whose form is not already known.

If we add such a mechanism to our RTAAI system, and if the user corrects the AAI errors as soon as they happen, unknown words will be lexicalized right after their first appearance. Then, the system shouldn't make more than one error per unknown word. With the RTAAI implementation described in Section 3.2, this is easily done: every time re-accentuation is invoked, we examine the last word-form of the context window; if it carries an accent and is unknown to the hypothesis generator, it is automatically added to the system. This way, we not only catch user-corrections, but also words that were originally accented by the user.

Table 7. *RTAAI with dynamic lexicalization.*

Text	Total errors (words)	Average distance between errors (words)	Improvement over baseline (% errors)
Court transcript	83	207.6	17.8%
Hansard	121	241.6	0.0%
Jules Verne	220	128.1	0.9%
total/avg.	424	192.4	4.5%

This mechanism was incorporated into our RTAAI system; its impact on the test corpus can be seen in Table 7, where the last column represents the improvement relative to standard RTAAI (the "baseline"), in percentage points.

As can be seen, dynamically lexicalizing unknown words has a different impact on different documents. The most striking improvement is observed on the court transcripts, which contain a lot of repeated proper nouns (individuals, places), initially unknown to the system. On the Hansard and Jules Verne documents, however, we get almost no improvement, because both texts contain very few "unknown" words (see Section 2.3).

Yet, the net effect is always positive, and all other user-feedback strategies described in this section are actually built on top of dynamic lexicalization.

4.2 Dynamic Language Models

While dynamic lexicalization can be quite effective for dealing with unknown words, it does not solve the problem entirely. Lexicalizing new word-forms makes it possible for the hypothesis generator to propose the correct accentuation when these word-forms re-appear in the text. However, they remain unknown to the language model, which is ultimately responsible for selecting the best accentuation among the proposed hypotheses. For example, suppose the user types in the proper name *René*; because this form is unknown, it will be dynamically added to the system, and will be proposed the next time the unaccented form *Rene* is encountered. However, if the hypothesis generator already had an entry for the form *Rêne* (*reindeer*), then this be proposed as well. At this point, whether the language model favors an unknown word-form over a known one depends on many factors, most notably the training material and the surrounding context[4].

Another problem with our RTAAI system is its sometimes annoying tendency to

[4] In our HMM implementation, unknown words are matched against a small number of regular expressions; all words that match the same expression are then viewed as occurrences of the same "unknown" word, which can be associated to any of a subset of the possible POS tags.

systematically select the most frequent alternative when confronted with syntactically interchangeable words. For example, the two French words *cote* (*quotation*, *rating*, *etc.*) and *côte* (*hillside*, *coast*, *rib*, *etc.*) 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 is much more frequent. Therefore, the RTAAI 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.

This is what Church and Gale refer to as the "bunching-up effect" (Church and Gale, 1995): a word that has appeared once in a text is much more likely to reappear in the near vicinity than its average frequency would suggest. Standard Markovian models are too weak to account for this phenomenon, which calls for some form of dynamic language modeling. In our case, this means a mechanism that takes recently encountered text into consideration when computing the probability of some newly edited segment of text. One approach is to constantly update the parameters of the language model as new text is encountered. For instance, Jelinek et al. (Jelinek et al., 1990) describe one such method to deal with unknown words: every time a new word-form w is encountered, it is added to the language model, and the relevant parameters (in our case, the $P(w|t_i)$) are estimated from the parameters of a set of synonyms. This set is computed automatically using co-occurrence statistics ("Which are the words that appear in a context similar to the one within which w appears?"). Unfortunately, this type of method is computationally quite expensive, and would thus be difficult to adapt to our real-time context.

A more practical approach is to combine local (dynamic) and global (static) information in real-time, without modifying the parameters of the language model. For example, Kuhn and De Mori (Kuhn and Mori, 1990) use such an approach to help a speech recognition application adapt to new context in real-time. Essentially, they suggest using recent data to estimate the parameters of a local (dynamic) language model, and to use this model in conjunction with the global (static) model when decoding new utterances.

To apply this kind of approach to RTAAI, we need to answer two questions:

- What kind of language model is best suited for modeling local phenomena?
- How do we combine the information coming from the local and global models?

In our case, we will want to focus on statistical models, because it makes it much easier to answer the second question. We will also want to restrict ourselves to very simple models, because by definition, "local" information necessarily comes in limited quantities, which quickly raises the problem of reliable parameter estimation if the model is too complex. In the following sections, we describe some variations on the dynamic modeling scheme proposed by Kuhn and De Mori.

4.2.1 Dynamic Unigram Models

The simplest statistical language model that comes to mind is what we could call a "unigram" Markovian model, or a "word-frequency" model, in which the words that make up a sentence are assumed to be independent of one another:

$$P(w_1^n) = P(w_1)P(w_2)...P(w_n)$$

Implementing a dynamic unigram model is trivial: to estimate $P(w_i)$, simply count the number of times word-form w_i appears in local context, and divide by the size of the context. To avoid null probabilities for words that do not appear in context, a simple trick is to count the currently hypothesized content of the AAI window as if it was part of the context.

The output of this dynamic model can then be linearly combined with that of the static (HMM) model:

$$P(w_1^n) = \alpha P_S(w_1^n) + \beta P_D(w_1^n), \tag{1}$$

where P_S and P_D denote the static and dynamic models respectively, and α and β are positive values such that $\alpha + \beta = 1$.

The most straightforward way to integrate this kind of scheme to our RTAAI implementation is to have the editor provide the AAI system with a "buffer" of local context at every re-accentuation round. Unfortunately, this is not practical, because depending on the amount of local context required, it may imply transfering excessively large quantities of information between the two processes. For this reason, we opted for an implementation based on a "cache" mechanism, as proposed by Kuhn and de Mori for their speech processing application: every time re-accentuation is invoked, the last word-form in the context window is appended to a queue, whose capacity is fixed; when this capacity is exceeded, words are dropped from the beginning of the queue. This way, the queue always contains the most recently encountered vocabulary. When the user is typing text in a continuous, left-to-right fashion, this queue is equivalent to a fixed-size window of context to the left of the cursor. In a session where the user hops around from place to place, the content of the queue looks more like a random sampling of words of the text.

Our experiments with this kind of dynamic language model reveal it to be inneffective: all runs on the training corpus to determine the optimal parameter values systematically pointed towards $\beta = 0$, whatever the quantity of local context considered. This would indicate that the unigram dynamic model is too weak to be of any real use to the static HMM: the occasional situation where it does have something interesting to say (such as accounting for bunching-up effects) does not compensate for the ill-informed predictions it makes in most other cases. As a result, it can only degrade the performance of the system.

4.2.2 Improved Unigrams

Clearly, some words are less likely than others to be affected by local factors: this is particularily true of so-called grammatical words. For instance, an occurrence of

Table 8. *Unigram dynamic modeling with stop-list.*

Text	Total errors (words)	Average distance between errors (words)	Improvement over baseline (% errors)
Court transcript	84	205.1	16.8%
Hansard	120	243.6	0.8%
Jules Verne	218	129.2	1,8%
total/avg.	422	192.6	5.0%

the French preposition *à* is probably not an indication that this word is suddenly more likely to appear in the near future. In a dynamic modeling scheme, this is a crucial issue, because modifying the behavior of the system for grammatical words is likely to have catastrophic consequences. One way of compensating for this is to weight each word's contribution in the dynamic model according to its likelihood of being affected by local factors. The dynamic probability of some sequence w_1^n would then be computed as

$$P_D(w_1^n) = W(w_1)P_D(w_1)...W(w_n)P_D(w_n)$$

Intuitively, in such a scheme, grammatical words such as *à* would have weights close to zero, while syntactically interchangeable words such as *cote* would have weights close to 1. Finding the right value for each weight is problematic, however.

An approach that is much simpler, and possibly just as effective, is to filter the input of the cache with a hand-made list of grammatical and high-frequency words. This is actually equivalent to starting with all weights equal to one, and then setting them to zero for all words in the stop-list. We call this the "stop-list" model.

We can also take an opposite approach: use the cache only for words that are very likely to be affected by local factors. One way of attaining this goal is to cache only those words whose accentuation was corrected by the user. The intuition here is that words that do not require user-corrections can be assumed to follow their "usual" statistical distributions. We call this the "correction-only" model.

We experimented with both strategies. The best results on the training portion of the corpus for the stop-list model were obtained with a queue of 200 words and $\beta = 0.16$. For the correction-only model, the optimal queue had 300 words and $\beta = 0.04$. As can be seen in Tables 8 and 9, both models produce small improvements over dynamic lexicalization on the test corpus. However, it is not clear whether these improvements are significant.

Table 9. *Unigram dynamic modeling of user-corrections.*

Text	Total errors (words)	Average distance between errors (words)	Improvement over baseline (% errors)
Court transcript	82	210.1	18.8%
Hansard	121	241.6	0.0%
Jules Verne	211	133.5	5.0%
total/avg.	414	195.1	6.8%

4.2.3 Dynamic Hidden Markov Model

To get more out of local context, therefore, it seems that a more elaborate model is required. An interesting possibility is to replace the unigram model by a hidden Markov model, modified along the following lines so as to deal with data sparseness:

- It is unlikely that local factors have a major impact on the syntax of the language; therefore, "grammatical" parameters can be assumed to be the same locally and globally. In other words, the dynamic model uses the static $P_S(t_i|t_{i-1})$ parameters.
- As was the case with the unigram model, we can assume that the local behavior of grammatical words is not significantly different from their global behavior. With an HMM model, however, there is no need to use a stop-list, because filtering can be based on part-of-speech information instead. For instance, we can assume that $P_D(w|t) \approx P_S(w|t)$ when t is the *Preposition* tag; in other words, the dynamic model can "ignore" prepositions, pronouns and other grammatical POS's.
- Finally, we can merge tags. For instance, all static tags related to common nouns (*Noun-masc-sing*, *Noun-masc-plur*, *etc.*) can be merged into a single tag *Noun*. We then approximate $P_D(w|\textit{Noun-masc-sing}) \approx P_D(w|\textit{Noun})$. In practice, we can select which tags are merged together so as to focus on distinctions that are likely to matter for the AAI task. For example, it will probably be interesting to distinguish between past-participles and other verbal forms, but maybe less so between singular and plural forms of adjectives. In our dynamic HMMs static tags that are not "ignored" by the dynamic model are merged into only 6 dynamic tags: common nouns, proper nouns, adjectives, past participles, other verbal forms and unknown words. Some preliminary experiments with more tags did not appear to produce significantly better results.

Estimating the parameters of such a dynamic HMM is a bit more subtle than for the unigram, because each word in the local context must be associated with one of the dynamic model's tags. In our case, there is an easy way of classifying

context words: we can use the static language model to compute the most probable ("Viterbi") sequence of tags. Once this is done, an estimate of $P_D(w|t)$ is obtained by dividing the frequency of (w,t) pairs by the total number of words associated to tag t in the local context.

In practice, if we combine the results of this dynamic model with those of the static model linearly, as we did with the unigram models, we obtain no significant improvement. To get the most out of this model, we need a tighter combination, that uses distinct pairs of coefficients α_j and β_j for each dynamic tag:

$$P(w_1^n) = \sum_{t \in T^n} \prod_{i=1}^{n} P_S(t_i|t_{i-1}) \prod_{i=1}^{n} \left(\alpha_{t_i} P_S(w_i|t_i) + \beta_{t_i} P_D(w_i|t_i)\right), \tag{2}$$

In other words, rather than combine models, we combine individual parameter estimates.

Also, following Kuhn and de Mori's suggestion once again, instead of having a single queue in the cache, we set each dynamic tag with a separate queue, whose size is parameterizable. As a **consequence**, the amount of local context used by the dynamic model is not necessarily the same for all tags, and actually changes dynamically as context (and the POS distributions) evolves. This way, we use the same amount of data to estimate probabilities for all dynamic tags.

To adjust the relative contribution of each model, instead of just two parameters (cache size and β), we now have a dozen. Kuhn and De Mori suggest that the values for these parameters can be determined by deleted interpolation (Jelinek and Mercer, 1981). In practice, we found that the dynamic model was fairly insensitive to even large variations in cache-size; overall, the best results were obtained with all queues limited to 200 words. As for the α/β parameters, they appeared to be quasi-orthogonal, so that each could be optimized separately by hill-climbing. In general, the optimal contribution of the dynamic model varied between 0.0001 (proper nouns) and 0.4 (past-participles).

Table 10 shows the optimal results on the test corpus. The net effect is an improvement over dynamic lexicalization alone, which we believe to be significant. Overall, the number of accentuation errors passed from 424 to 397, and the average distance between errors from 193 to 206. As a point of reference, this occurrence of the word **consequence** and the previous (both in boldface font) are exactly 206 words apart.

5 Conclusions

We have presented a method for automatically inserting accents into French text, based on a statistical language model. This method was implemented into a C library of functions and program called `Réacc`, which are now commercially available through Alis Technologies[5]. We have also shown how this method can be used to do on-the-fly accent insertions in real-time, within a word-processing environment.

[5] Alis Technologies: `http://www.alis.com`

Table 10. *HMM dynamic modeling.*

Text	Total errors (words)	Average distance between errors (words)	Improvement over baseline (% errors)
Court transcript	76	226.7	24.8%
Hansard	115	254.2	5.0%
Jules Verne	206	136.8	7.2%
total/avg.	397	205.9	10.6%

A prototype RTAAI system was also implemented and integrated into the Emacs editor. Finally, we have shown how dynamic language modeling techniques could be used to exploit user-feedback and further improve the performance of the system.

Text processed with our system contains less than one accent error per 180 words on average, regardless of whether the system is used on its own or within an RTAAI environment. On a Sun Sparc Ultra 1 computer, with 256 MB, the system will process approximately 50 000 words per minute. Within the Emacs RTAAI prototype, because AAI is performed asynchronously, the performance of the editor itself is not affected, and accents are inserted faster than any typist that we know of can type. With dynamic modeling, the error rate is further reduced, to approximately 200 words between errors.

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 RTAAI method to a "real" word-processor, such as Microsoft Word.

Acknowledgments

We are greatly indebted to George Foster, Pierre Isabelle, Philippe Langlais and Guy Lapalme, for their invaluable advice and constructive comments, as well as to Elliott Macklovitch, for helping us translate our 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.

References

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

Church, K. W. (1988). A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proceedings of the 2nd Conference on Applied Natural Language Processing (ANLP)*, Austin, Texas.

Church, K. W. and Gale, W. A. (1995). Inverse Document Frequency (IDF): A Measure

of Deviation from Poisson. In *Proceedings of the 3rd ACL Workshop on Very Large Corpora (WVLC)*, Cambridge, Massachusetts.

DeRose, S. J. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14:31–39.

El-Bèze, M., Mérialdo, B., Rozeron, B., and Derouault, A.-M. (1994). Accentuation automatique de textes par des méthodes probabilistes. *Technique et sciences informatiques*, 13(6):797–815.

Foster, G. F. (1991). Statistical Lexical Disambiguation. Msc thesis, McGill University, School of Computer Science.

Jelinek, F. and Mercer, R. L. (1981). Interpolated estimation of markov source parameters from sparse data. In Gelsema, E. S. and Kanal, L. H., editors, *Pattern Recognition in Practice*, pages 381–397. North Holland, Amsterdam.

Jelinek, F., Mercer, R. L., and Roukos, S. (1990). Classifying Words for Improved Statistical Language Models. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP) 1990*, pages 621–624, Albuquerque, New Mexico. IEEE.

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

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

Yarowsky, D. (1994a). A Comparison of Corpus-based Techniques for Restoring Accents in Spanish and French Texts. In *Proceedings of the 2nd ACL Workshop on Very Large Corpora (WVLC)*, Las Cruces, New Mexico.

Yarowsky, D. (1994b). Decision Lists for Lexical Ambiguity Resolution: Applications to Accent Restoration in Spanish and French. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL)*, Las Cruces, New Mexico.