Truecasing For The Portage System

Akakpo Agbago, Roland Kuhn, George Foster
Institute for Information Technology, National Research Council of Canada
{Akakpo.Agbago, Roland.Kuhn, George.Foster}@nrc-cnrc.gc.ca

Abstract

This paper presents a truecasing technique - that is, a technique for restoring the normal case form to an all lowercased or partially cased text. The technique uses a combination of statistical components, including an N-gram language model, a case mapping model, and a specialized language model for unknown words. The system is also capable of distinguishing between "title" and "non-title" lines, and can apply different statistical models to each type of line. The system was trained on the data taken from the English portion of the Canadian parliamentary Hansard corpus and on some English-language texts taken from a corpus of China-related stories; it was tested on a separate set of texts from the China-related corpus. The system achieved 96% case accuracy when the China-related test corpus had been completely lowercased; this represents 80% relative error rate reduction over the unigram baseline technique. Subsequently, our technique was implemented as a module called Portage-Truecasing inside a machine translation system called Portage, and its effect on the overall performance of Portage was tested. In this paper, we explore the truecasing concept, and then we explain the models used.

1. Introduction

Many natural language processing engines output text that lacks case information – by convention, usually in lowercase. For instance, Portage-Truecasing is incorporated in a machine translation system called Portage whose initial translations are generated in lowercase format. Thus, to complete the translation task, the system needs a truecasing module that will change some of the characters in the initial translation to uppercase. Systems that carry out named entity recognition, spelling correction, and grammar correction may also require truecasing modules to function properly.

To illustrate the use of truecasing, consider the following example. Let us assume that an automatic speech recognition or machine translation system outputs the sentence *"sir john a macdonald drank old covenanter whiskey"*. The sentence is much easier to read and to understand in its truecase form: *"Sir John A MacDonald drank Old Covenanter whiskey"*. In this version, "Sir John A MacDonald" and "Old Covenanter" are clearly understood to be names. (After truecasing, the typical next step is punctuation insertion).

Few people have worked on this problem. The most recent papers are by Chelba and Acero [2] and by Lita *et al.* [5]. Chelba and Acero's technique is based on maximum "a posteriori" (MAP) adaptation of Maximum Entropy Markov Models (MEMMs) to solve this problem. These authors obtained a 35-40% relative improvement for the baseline MEMM over a 1-gram baseline, and a 20-25% relative improvement for the MAP-adapted MEMM over the baseline MEMM (in tests done on Broadcast News data). Lita *et al*. [5] used a truecasing approach based on trigram language modeling. They obtained relative error rate improvement over a unigram baseline of about 50% (from 96% accuracy to 98%) on a news articles from which titles, headlines, and section headers had been excluded, and an even greater relative error rate improvement of about 66% (from about 94% accuracy to about 98%) over the baseline on a test corpus comprising titles, headlines, and section headers. Finally, Mikheev's work [1] targeted the parts of a text where capitalization is expected, such as beginning of sentences and quotations. Similarly, Kim and Woodland [3] used rule-based techniques to generate punctuation and beginning of sentence capitalization for speech recognition outputs.

We began by implementing a unigram baseline system that yielded 19.35% case error; implementation of a trigram-related model similar to that of Lita *et al*. lowered this to 5.24% (relative error rate reduction of 73%). Careful study of the problems seen on a development set showed that many of the errors came from titles, and from "unknown" words – *i.e.*, those encountered during testing but not during training. Thus, we extended the basic approach by incorporating a **title**

detector which attempts to label lines as being either "title" or "non-title". This gives us the option of training separate title and non-title casing models for application at runtime. In addition, we grouped "unknown" words into four classes. For each such class, the case probabilities are determined from the cases of low-frequency words in the training data that fall into that class.

The language models described in this paper were trained using the SRI Language Modeling Toolkit (SRILM). Since one of the goals of this work was to improve the performance of a machine translation (MT) system participating in a NIST MT task, much of the training data was drawn from the 2004 NIST "Large" Chinese-English training corpus. This "C/E" corpus includes texts from a variety of China-related sources. Additional training material was drawn from the Canadian parliamentary Hansard corpus. The test data were the 2004 NIST C/E evaluation data.

The metric employed for the C/E MT NIST task is BLEU (see Papineni et al. [4]), which measures the similarity of the translation system's output with one or more reference translations. In this paper, we measure the performance of the truecasing module both by how accurately it assigns case to normal text that has had case information removed, and by its effect on BLEU. We define "case accuracy" per word - a case error in a single character of a word is counted as a case error for that word. The goal of optimizing performance according to one of these metrics may conflict with optimizing performance according to the other. Suppose that the MT system outputs "elephants in africa mostly has long nose" and the truecasing module converts this to "elephants in Africa mostly has long nose". We might be tempted to add a rule to the truecasing module that imposes uppercase for the first letter in every sentence. Though this rule might help performance according to the "case accuracy" metric, it may hurt the BLEU score. In the example, if the reference sentence were "Most elephants in Africa have long noses", BLEU will assign a higher score to "elephants in Africa mostly has long nose" than to "Elephants in Africa mostly has long nose" (because the form of "elephants" in the reference is all-lowercase).

The layout of this paper is as follows: section 2 will outline the problem, section 3 will describe the statistical models, section 4 will describe the experiments and their results, and section 5 will discuss these results.

2. The Problem of Truecasing

The truecasing problem is not obvious until one faces a real example. Consider the sentence "*indian paratroopers will command a joint alpha-tango military exercise with the special forces of the us pacific command*". In languages employing the Latin alphabet, a sentence typically begins with uppercase. Therefore, "indian" should be "Indian" with little ambiguity. The word "us" could remain lowercase but the word sequence "us pacific command" suggests that the all-uppercase form "US" is more likely. Thus, word context can provide clues to case. In a syntactic approach, some aspects of context could be exploited by means of Part-Of-Speech (POS) tagging. For instance, the tagger might tag "the us pacific command" as "the <noun phrase>" and use the information that "us" is part of a noun phrase to generate "the US Pacific Command".

At the beginning of our work on truecasing, we investigated the distribution of the casing errors of a unigram truecaser. This system, which was used as the baseline in subsequent experiments, assigns to words observed in the training data the most frequent case observed. New words seen in the test data for the first time – the so-called "unknown" words - are left in lowercase. The resulting error distribution is plotted below (with words of similar frequency in the training corpus binned together).

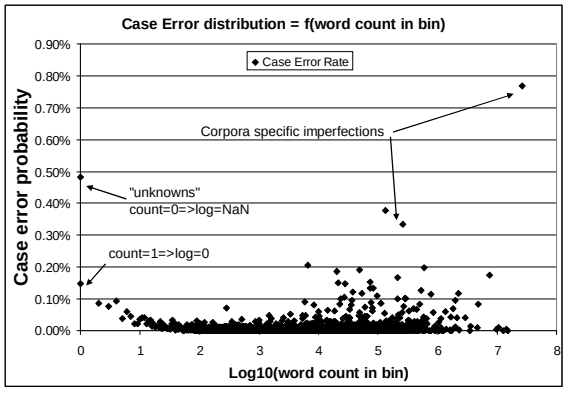

Figure 1: Case error distribution of the baseline truecaser as a function of word count in our training corpus

The point marked "unknowns" appears for convenience on the y axis (though its true x coordinate is not 0 but -∞); it represents words

appearing in the test data that were not seen in the training data. It is not surprising that these words have a higher case error rate than the words of count 1: the baseline system has not learned anything from the training data about the "unknown" words. At the high end of the x axis, we see that a few very frequent words such as "the" also have a high error rate. This is partly because of tokenization problems (*e.g.*, "the" sometimes has a hyphen glued to the end of it, or a quotation mark glued to the front of it) and partly because "the" and similar words often appear in titles, which are particularly tricky.

In the truecasing approach we used (similar to [5]), an N-gram language model (LM) is used to model the contextual information. In the example, if the trigram "US Pacific Command" has often been seen, then the system will be inclined to carry out truecasing correctly. The "case mapping" model smooths the N-gram model. If (for instance) the erroneous sequence "will Command a" occurred once in the training data, this smoothing ensures that an occurrence of "command" preceded by "will" and followed by "a" will still receive the correct all-lowercase form in the system's output.

We need a third kind of model to deal with "unknown" words – *i.e.*, those that were not observed in the training data. In the example, it is quite likely that no form of "*alpha-tango*" (a rare military code word) has been observed. Nevertheless, the "unknown word" model we provide will be capable of converting it to the correct form, "*Alpha-Tango*".

3. Scoring Function and Models

We gave in chapter 2 some motivations for three sub-models. To use the specific contribution of each sub-model, we combine them into a scoring function Ω formulated by Eq.1. The sub-models are:

- An N-gram model called θ_N to capture the contextual information surrounding a word;
- A case mapping model called Φ to capture the probabilities for different cases of a word;
- An "unknown word" model called Л to provide for unseen cases.

$$\Omega = \theta_N \otimes \Phi \otimes \Pi \qquad \text{Eq.1}$$

3.1 Terminology

Let S denote a sequence of words s_i, with case information included. Let C() denote the function that gives only the casing of a string, and L() the function that returns its lowercase form, thus leaving only information about the uncased word sequence. Let AU denote "All Uppercase", FU "First letter Uppercase", AL "All lowercase", and MC "Mixed Case"; for S = "USA is an acronym for United States of America", C(S) = AU AL AL AL AL FU FU AL FU, and L(S) = "usa is an acronym for united states of america". Truecasing is applied when we know L(S) and are trying to obtain C(S). If both the case information C(S) and the word information L(S) are known for a string S, S is completely defined.

3.2 N-Gram model θ_N

One way of estimating the probability that s_i has a particular case $C(s_i)$ would be to assume recursively that we already know the case of the words preceding a particular word s_i in the string S. This line of thought leads to the N-gram component θ_N of the truecaser. For instance, for N=3, let $P_{\theta 3}(C(s_i) \mid L(s_i), s_{i-2}s_{i-1})$ denote the probability that the $C(s_i)$ form of s_i (rather than some other form) occurs after the cased word sequence $s_{i-2}s_{i-1}$. An example: if $L(s_i)$ = "america", and that $s_{i-2}s_{i-1}$ = "States of", the trigram-related probability of "America" is $P_{\theta 3}(C(s_i)=FU \mid L(s_i)=$"america", "States of").

3.3 Case mapping model Φ

Another way of estimating the probability that s_i has a particular case $C(s_i)$ would be to ignore context and rely on the case forms observed in the training data for s_i. This leads to the case mapping model, $P_\Phi(C(s_i) \mid L(s_i))$. For example, the probability of "America" given that some form of "america" has occurred is denoted $P_\Phi(C(s_i)=FU \mid L(s_i)=$ "america"). Using Φ alone would be equivalent to considering the most probable case pattern for a word everywhere. This sub-model is used to smooth the model θ_N.

3.4 Unknown word model Л

Finally, the sub-model called Л deals with words s_i that weren't observed in the training data. It was constructed by defining classes based on the form

of a word – for instance, the presence of non-word symbols (*e.g.*, internal hyphen). It's formulated as

$$P_{\Pi}\left(C(s_i) \mid L(s_i)\right) \approx P\left(C(s_i) \mid Class(L(s_i))\right)$$

The conditional probability on the right side above is calculated from the case statistics for words that belong to the class, and that occur exactly once in the training data. Our assumption is that low-frequency words in a given class tend to follow similar patterns of case.

How should the function *Class(L(s$_i$))* be defined? Depending on the test corpus, the nature of such "unknown" words may vary. They include rare proper names such as "agbago" and mixed alphanumeric expressions such as "$2563US" or "675km" or "220kV". Other forms are compounded name entities and character sequences resulting from words in non-alphabetic languages. This last type of "unknown" word sometimes occurs in the English portion of the C/E corpus when Chinese characters have been inserted in English text (*e.g.*, to clarify the meaning of an English word to Chinese readers).

Based on the characteristics of the C/E corpus, we decided to define the following "unknown" word classes:

1. *quantity words*: "unknown" tokens starting or/and ending with numbers. Example: "us$0.19", "10kV", "rmb0.308".
2. *acronyms*: "unknown" tokens containing a sequence of single letters followed by periods. Example: "u.s.", "u.s.-south".
3. *hyphenated words*: "unknown" tokens made up of at least two components joined by a hyphen, where each component consists of a sequence of alphabetic characters. Example: "belarus-russian", "jong-il".
4. *regular uniform words*: "unknown" tokens consisting entirely of alphabetic characters. Example: "abesie", "badeshire".

These classes are considered in the precedence order just given. Thus, an unknown token is only considered for class 2 if it has been rejected for class 1, and so on (that's why "u.s.-south" is assigned to class 2 and not class 3). "Unknown" tokens not falling into one of these four classes are left in all-lowercase form (an example is the "unknown" token "*cafâ¨â¦*" we observed during our tests, which results from a word that combines alphabetic letters and Chinese characters).

3.5 Scoring function Ω

The θ_N , Φ and Л components of Portage's truecasing module (defined above) are true probabilities. The scoring function Ω combines them in the following way:

$$\Omega\left(C(s_i) \mid L(s_i)\right) \approx \begin{cases} P_{\Pi}\left(C(s_i) \mid L(s_i)\right), & \text{if } s_i \text{ unknown} \\ P_{\theta_N}\left(C(s_i) \mid L(s_i), s_{i-1}s_{i-2}\right) & \\ \quad * P_{\Phi}\left(C(s_i) \mid L(s_i)\right), & \text{else} \end{cases}$$

Although Ω defined in this way is not a probability because of the product term, it has certain advantages (*e.g.*, ease of implementation in the SRILM framework). The way Ω is formulated indicates that at the step i, we already know the case of the words preceding s_i in the string S. To get a sense of how Ω works, consider the following training text:

> *"Akakpo is the son of Agbago. So his name is said and written as Akakpo Agbago in Canada but Akakpo AGBAGO in Togo. Akakpo AGBAGO is unique in Togo. Akakpo is a last name for many. Agbago is a good guy. Agbago is smart. Agbago is kind."*

And the following test text:

> *"Akakpo agbago"*

Let's redefine the θ component slightly so it's based on bigrams rather than trigrams, and let's ignore smoothing and assume the component models use frequencies directly to estimate probabilities.

Then using these training and testing texts, we obtain:

Step i = 1:

$$\begin{aligned} \Omega\left(Akakpo \mid akakpo\right) &= P_{\theta_N}\left(Akakpo \mid Akakpo\right) \\ &\quad * P_{\Phi}\left(Akakpo \mid akakpo\right) \\ &= 1 * 1 \\ &= 1 \quad "akakpo" \text{ is known} \end{aligned}$$

$$\hat{C}\left(s_1\right) = \text{Akakpo}$$

Step i = 2:

$$\Omega(\text{Agbago} \mid \text{agbago, Akakpo}) =$$
$$P_{\theta_N}(\text{Agbago} \mid \text{agbago, Akakpo}) * P_{\Phi}(\text{Agbago} \mid \text{agbago}) = \frac{1}{3} * \frac{5}{7} = \frac{5}{21}$$
$$\Omega(\text{AGBAGO} \mid \text{agbago, Akakpo}) =$$
$$P_{\theta_N}(\text{AGBAGO} \mid \text{agbago, Akakpo}) * P_{\Phi}(\text{AGBAGO} \mid \text{agbago}) = \frac{2}{3} * \frac{2}{7} = \frac{4}{21} \quad \text{"agbago" is known}$$
$$\hat{C}(s_2) = \text{Akakpo Agbago}$$

Thus, the scoring function Ω, if trained on this corpus, would tend to predict "Agbago" rather than "AGBAGO" after "Akakpo". This prediction is incorrect in Togo, but correct in Canada (and most of the English-speaking world) – an example of how slippery the notion of correct casing can get.

We also tried a different approach in which we find the cased form that maximizes the trigram probability, given the lowercase form and the two preceding cased forms. Let S denote the entire cased word sequence, and L the corresponding sequence of lowercased words. By Bayes's Law, we have

$$P(S \mid L) = P(L \mid S) * P(S) / P(L)$$

However, by definition we know the lowercase word sequence L. Thus, we want to maximize

$$P(S \mid L) \propto P(L \mid S) * P(S)$$

Substituting in the trigram estimate of $P(s_i)$, we see that at each step we are trying to maximize

$$P(L(s_i) \mid s_{i-2}, s_{i-1}, s_i) * P(s_i \mid s_{i-2}, s_{i-1})$$

Thus, we search over the cased forms s_i of $L(s_i)$ observed in the training data to find the one that maximizes this expression. For an observed form s_i of $L(s_i)$, $P(L(s_i)|s_{i-2},s_{i-1},s_i)$ will be 1. In initial experiments, this approach yielded inferior performance to that obtained by using the scoring function Ω above.

4. Experiments and Results

We used the SRILM package, along with some code we wrote ourselves, to handle the training (creation of the language models) and the case decoding (also called "disambiguation"). The θ models are produced in the ARPA N-gram LM format and the Φ and Л models in SRILM "V1 to V2" mapping format.

The resources used were as follows:
- Training corpus: contains 366,532,578 tokens (~words).
- Test corpus: contains 451,154 tokens (~words)

Recall that the training corpus comes from the English-language half of the 2004 NIST "Large" Chinese-English (C/E) training corpus (which includes material from Hong Kong Hansard and news sources such as Xinhua News Agency, Associated Press, Agence Française de Press, *etc.)*, supplemented by material from the Canadian Hansard corpus. The test corpus is the 2004 NIST C/E test set.

To the scoring function Ω described above, we added some **heuristics**. These are:

- Junk cleaning: we removed from the training data various special tags; also, all lines in which most words are uppercased (these turned out to be extremely atypical).
- Title detection and processing: titles show unusual casing patterns. Unfortunately, in English there are no explicit rules for casing in titles; frequently, casing is left to the whims of the author. We implemented a title detection module that relied on domain specific aspects of our data, which consisted mainly of newswire data. For instance, the presence of a date, name of a news agency, and "reported by" followed by a personal name was taken to indicate a title. Used on training data, this module makes it possible to train "body only" or "title only" models; used on test, it makes it possible to apply different models or rules to title and body. The best-performing system shown in Figure 2 was trained only on the portion of the training corpus classified as "body" by the title detector. For casing of test text, this body-only system was applied both to portions of the test corpus classified as "body" and as "title". Then, words in the title that were longer than four letters were systematically uppercased. It might seem more logical to use a model trained on titles to assign case to words

in titles, but the main characteristic of titles in the training text is inconsistency in case assignment. Thus, the title detector's usefulness for training is that it enables us to **remove** titles from the training data.

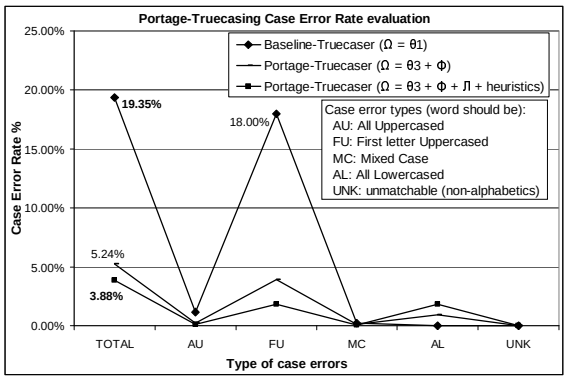

Figure 2: performance of Portage-Truecasing

The performance of Portage-Truecasing is plotted in Figure 2 and shows 80% improvement in relative error rate over the unigram baseline technique, $\Omega = \theta_1$ (from 19.35% to 3.88%). The figure also shows case error by the correct case type – *e.g*., the points above "AU" show error rates for words that **should** be written all-uppercase. From the figure, it is clear that the effort that went into classifying different types of unknown words for the Ɪ model and into developing heuristics (junk cleaning and title detection) was justified: it yielded an improvement of 26% relative (from 5.24% to 3.88% error). If we do **not** include the heuristics but only Ɪ, the improvement is only 13% relative (to 4.55% case error – this point is not shown in the figure 2). Figure 3 provides an analysis of Portage-Truecasing errors by word count, as was done in Figure 1 for the baseline truecaser.

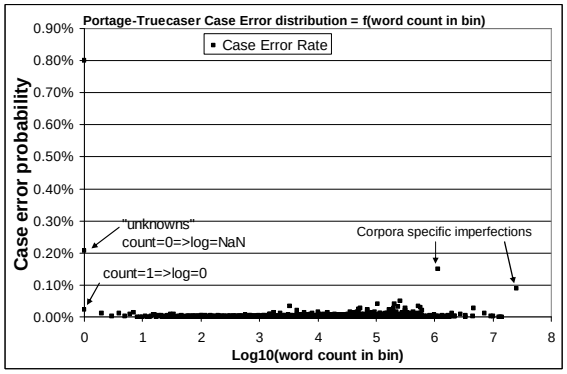

Figure 3: Case error distribution of Portage-Truecaser as a function of word count in our training corpus

Since the module is used for machine translation (MT), we ran it on the output of the MT system and obtained the BLEU results in Table 1. The last two columns show that the unknown word model Ɪ helps performance, while the use of heuristics causes slight deterioration. As noted earlier, it is not surprising that the best-performing system according to case error rate (this system includes the heuristics) is not the same as the best-performing system according to BLEU (this system does not include the heuristics). For most applications, the case error rate is more informative.

Baseline-Truecaser ($\Omega = \theta_1$)	Portage-Truecaser ($\Omega=\theta_3+\Phi$)	Portage-Truecaser ($\Omega=\theta_3+\Phi+$Ɪ +heuristics)	Portage-Truecaser ($\Omega=\theta_3+\Phi+$Ɪ)
17.98%	23.77%	23.74%	23.83%

Table 1: BLEU score

5. Discussion

In this paper, we presented a module designed as part of a machine translation system that uses three statistical language models to assign case to a text. It reduces the case error rate of a unigram baseline truecaser by 80% relative, achieving 96% global accuracy on the test corpus. We designed the system in a manner that allowed us to quickly test different variants; this was fortunate, because the best variant according to case error rate and the best variant according to BLEU turned out to be different.

Some specific problems we encountered were:

- *Inconsistency*: there was a non-negligible proportion of case inconsistencies in training and test corpora. This happens because the corpora are agglomerations of texts written by different people with different formatting styles, competences, and working tools. Furthermore, not much attention is paid to enforcing casing standards, even where these exist. Named entities (e.g., "United States Government" *vs*. "United States government") and titles tend to be subject to casing inconsistency.
- *Portage specific side effects*: errors from other components of the system, particularly the tokenizer, had a strong negative impact on performance.

For future work, we could consider turning scoring function Ω into a true probability by interpolating the θ_N and Φ terms instead of multiplying them – *i.e.*, defining it as:

$$\Omega(C(s_i)\,|\,L(s_i)) \approx \begin{cases} P_{\Pi}(C(s_i)\,|\,L(s_i)), & \text{if } s_i \text{ unknown} \\ \lambda * P_{\theta_N}(C(s_i)\,|\,L(s_i), s_{i-1}s_{i-2}) + & \\ (1-\lambda) * P_{\Phi}(C(s_i)\,|\,L(s_i)), & \text{else} \end{cases}$$

Alternatively, we could approximately keep Ω in its current form, but incorporate power terms α and β that would depend on the frequency of a word (where K is a normalization factor):

$$\Omega(C(s_i)\,|\,L(s_i)) \approx \begin{cases} P_{\Pi}(C(s_i)\,|\,L(s_i)), & \text{if } s_i \text{ unknown} \\ P_{\theta_N}(C(s_i)\,|\,L(s_i), s_{i-1}s_{i-2})^{\alpha} & \\ \quad * P_{\Phi}(C(s_i)\,|\,L(s_i))^{\beta} / K, & \text{else} \end{cases}$$

It is interesting to think about how one would build a truecaser optimized for MT (*i.e.*, to maximize the BLEU score). MT output is not exactly the same as regular text. One might consider training the truecaser on output from the MT system whose words have been assigned case in some other way (*e.g.*, by "pasting" onto them case patterns from corresponding words in reference data).

References:

[1] A. Mikheev, "*A knowledge-free method for capitalized word disambiguation*", 37th conference on Association for Computational Linguistics, pp. 159 – 166, 1999, College Park, Maryland, ISBN.

[2] C. Chelba and A. Acero, "*Adaptation of Maximum Entropy Capitalizer: Little Data Can Help a Lot*", Conf. on Empirical Methods in Natural Language Processing (EMNLP) July 2004, Barcelona, Spain.

[3] J.H. Kim and P. C. Woodland, "*Automatic Capitalization Generation for Speech Input*", Computer Speech and Language, 18(1):67–90, January 2004.

[4] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "*BLEU: A method for automatic evaluation of machine translation*", IBM Technical Report RC22176, Sept. 2001.

[5] L. Lita, A. Ittycheriah, S. Roukos, and N. Kambhatla, "*tRuEcasIng*", Proceedings of ACL 2003, pp. 152–159, Sapporo, Japan.