Automatic Word Spacing Using Probabilistic Models Based on Character *n*-grams

Do-Gil Lee, Hae-Chang Rim, and Dongsuk Yook, *Korea University*

These models regard automatic word spacing as a classification problem. By generalizing hidden Markov models, they consider a broader context and estimate more accurate probabilities.

On the Internet, information is largely in text form, which often includes such errors as spelling mistakes. These errors complicate natural language processing because most NLP applications aren't robust and assume that the input data is noise free. Preprocessing is necessary to deal with these errors and meet the growing need for automatic text processing.

One kind of such preprocessing is automatic word spacing. This process decides correct boundaries between words in a sentence containing spacing errors, which are a type of spelling error. Except for some Asian languages such as Chinese and Japanese, most languages (including English and Korean) have explicit word spacing. In these languages, word spacing is crucial to increase readability and to accurately communicate a text's meaning. For example, if the Korean sentence *a-beo-ji-ga bang-e deul-eo-ga-syeoss-da* (Father entered the *room*) is written as *a-beo-ji ga-bang-e deul-eo-ga-syeoss-da* (Father entered the *bag*), its meaning changes significantly. Automatic word spacing plays an important role not only as a spell-checker module but also as a preprocessor for a morphological analyzer, which is a fundamental tool for NLP applications. Furthermore, automatic word spacing can serve as a postprocessor for optical-character-recognition systems and speech recognition systems.

Although researchers have overlooked automatic word spacing for a long time, word-spacing errors are common, especially in Korean text. Automatic word spacing of Korean is important and difficult for two reasons. First, Korean spelling rules are sometimes confusing. For example, Korean written text allows both splitting a compound word into several words and concatenating words to make a compound word. However, some exceptions exist, and remembering all of them is difficult. So, even native Korean speakers often make writing mistakes. Second, Korean text has no hyphenation. That is, if a whole word doesn't fit at the end of a line, the word's first part appears at the line's end, without a hyphen, and the rest continues on the next line. These difficulties have renewed interest in automatic word spacing for Korean in recent years.

We regard the word-spacing problem as a classification problem similar to part-of-speech tagging. Hidden Markov models[1–3] are a widely used probabilistic method to solve such NLP problems as POS tagging. We've applied HMMs to create models for automatic word spacing that use character *n*-grams—that is, a sub-sequence of *n* characters in a given character sequence. For example, a character bigram is any combination of two characters with or without a space. This technique lets us use raw texts as training data. By generalizing the HMMs, our models can consider a broader context and estimate more accurate probabilities.

Basic approaches

Previous approaches for automating Korean word spacing have been rule based or statistical. Table 1 summarizes the two approaches' characteristics.

1541-1672/07/$25.00 © 2007 IEEE
Published by the IEEE Computer Society

Table 1. Characteristics of rule-based and statistical approaches to automatic word spacing.

Approach	Information source	Weaknesses	Strengths
Rule based	Handcrafted linguistic knowledge	• Knowledge-dependent • High computational complexity • Heavy demand on human effort	• A more linguistic perspective
Statistical	Automatically extracted statistics from raw corpora	• Data dependent • A less linguistic perspective	• Robust for unknown words • No demand on human effort

The rule-based approach

This approach sequentially recognizes words in a sentence from left to right or from right to left, and then inserts a space between the words. This approach usually uses lexical information and heuristic rules.[4–7]

The lexical information consists of *Josa* and *Eomi* lists, a word lexicon, and so on. *Josa* is a Korean morpheme that is attached to a verbal root; *Eomi* is a Korean morpheme that is attached to a noun. (Morphemes are the minimal meaningful language units. A Korean word consists of one or more morphemes; for example, the word *hag-gyo-e* consists of two morphemes: *hag-gyo* and *e*.)

The heuristic rules consist of either the longest-match-preference rules or shortest-match-preference rules, morphological rules, and error patterns. (When the rule-based approach tries to recognize words in the input sentence, a word can be a substring of another longer word in a lexicon. The shortest-match-preference rule splits the input string as soon as it finds a matching word in the lexicon. The longest-match-preference rule delays the split until it can't find any longer matching word in the lexicon.)

Although the rule-based approach takes a linguistic perspective, it has two basic disadvantages. First, it requires higher computational complexity than the statistical approach. Second, it incurs a considerable cost for constructing and maintaining lexical information. Because Korean morphology has high agglutinativity (that is, so many combinations of morphemes are possible), there are too many possible word forms to list in the dictionary. So, most rule-based systems for Korean word spacing use a morphological analyzer to recognize word boundaries. Morphological analysis breaks a word into morphemes and assigns a POS tag to each morpheme.

The morphological analyzer can introduce other disadvantages. Automatic-word-spacing results depend highly on the morphological analyzer's results. If morphological analysis fails owing to unknown words, the word-spacing system won't detect the correct word boundary. Also, the more analyses that are possible because of lexical ambiguity, the larger the search space is. This situation might result in frequent backtracking and error propagation. The most serious disadvantage is that if the system derives a word that's actually erroneous owing to overgeneration, the system won't even detect the word-spacing error. (Overgeneration means that, owing to the text's ambiguity, the system produces too many analyses.) Besides, if the system is acting as a morphological analyzer's preprocessor, the repetition of the same process for morphological analysis is unavoidable.

The statistical approach

This approach uses character statistics extracted from many corpora (large, structured sets of text) to decide whether to place a space between two adjacent characters.[8–12] Unlike the rule-based approach, this approach doesn't require constructing and maintaining statistics by hand; they can be automatically acquired from raw corpora. It's more robust against unknown words than rule-based approaches using a morphological analyzer because it doesn't use a word list.

Most statistical systems use word-spacing probability estimated from every character bigram in the corpora. To decide whether to insert a space between two characters, these systems first calculate the probability by combining local evidences for the character's surrounding context. They then compare the probability with a certain threshold. If the probability is higher than the threshold, the system inserts a space.

As far as we know, the state of the art in automatic word spacing is a statistical method that Seung-Shik Kang and Chong-Woo Woo proposed (see the sidebar). However, their method doesn't consider all possible interpretations of a sentence. We developed our models to overcome this limitation.

Probabilistic models for automatic word spacing

For a given sentence of characters $C = c_{1,n}$, to find the most likely sequence of word spacing tags $\hat{U}$ among all possible word-spacing tags $U = u_{1,n}$, we use

$$\hat{U} = \arg\max_{U} P(U|C)$$

where $P(U|C)$ is the probability of having the word-spacing tag U when the sentence C without spacing is given.

The word-spacing tag indicates whether the current and next character should have a space between them. The tag "1" means that there should be a space; "0" means that there shouldn't. For example, if we attach tags to *gong-bu-hal su iss-da.*, which means "I can study.", it becomes *gong*/0 + *bu*/0 + *hal*/1 + *su*/1 + *iss*/0 + *da*/0 + ./1.

We define the word-spacing function $\Gamma(C)$ as

$$\Gamma(C) = \arg\max_{U} P(U|C) \quad (1)$$

$$= \arg\max_{U} \frac{P(U)P(C|U)}{P(C)} \quad (2)$$

$$= \arg\max_{U} P(U)P(C|U) \quad (3)$$

$$= \arg\max_{U} P(U,C) \quad (4)$$

$$= \arg\max_{u_{1,n}} P(u_{1,n}, c_{1,n}) \quad (5)$$

Using Bayes' rule, equation 1 becomes equation 2. Because $P(C)$ is a constant for U, equation 2 is transformed into equation 3. By the multiplication rule, we transform equation 3 into equation 4. Finally,

Kang and Woo's Method for Automated Word Spacing

Seung-Shik Kang and Chong-Woo Woo's method[1] for automated Korean word spacing defines the word-spacing probability, $P(x_i, x_{i+1})$, between two adjacent characters, x_i and x_{i+1}, as

$$\begin{aligned} P(x_i, x_{i+1}) = {} & \alpha \times P_R(x_{i-1}, x_i) \\ & + \beta \times P_M(x_i, x_{i+1}) \\ & + \gamma \times P_L(x_{i+1}, x_{i+2}) \end{aligned}$$

where $\alpha + \beta + \gamma = 1$. If the probability is greater than a predefined threshold, the method inserts a space. P_R, P_M, and P_L denote the probability of a space being inserted to the right, in the middle, and to the left of the two characters:

$$P_R(x_{i-1}, x_i) = \frac{freq(x_{i-1}, x_i, space)}{freq(x_{i-1}, x_i)}$$

$$P_M(x_i, x_{i+1}) = \frac{freq(x_i, space, x_{i+1})}{freq(x_i, x_{i+1})}$$

$$P_L(x_{i+1}, x_{i+2}) = \frac{freq(space, x_{i+1}, x_{i+2})}{freq(x_{i+1}, x_{i+2})}$$

In these equations, *freq*(*x*) denotes a frequency of a string of characters *x* from the training data, and *space* denotes a space between two characters.

Kang and Woo reported that the performance is sensitive to the training data; it varies according to the similarity between the input document and training data. In addition, this method has a crucial problem because it doesn't consider the previous spacing state. For example, consider the sentence *gong-bu-hal-su-iss-da*, for which the correct word spacing is *gong-bu-hal su iss-da*. According to Kang and Woo's probability formula, the word-spacing probability of *su* and *iss* is

$$\begin{aligned} P(su, iss) = {} & 0.25 \times P_R(hal, su) \\ & + 0.5 \times P_M(su, iss) \\ & + 0.25 \times P_L(iss, da) \end{aligned}$$

The probability $P_R(hal, su)$ is

$$P_R(hal, su) = \frac{freq(hal, su, space)}{freq(hal, su)}$$

Because the correct sentence has a space between *hal* and *su*, the probability formula should have used *freq*(*space*, *su*, *space*) instead of *freq*(*hal*, *su*, *space*) to get the correct word-spacing probability. To alleviate this problem, we can consider the previous spacing state that the system decided. However, this approach could propagate errors if that spacing state is incorrect. To avoid such errors, the automatic word-spacing system must generate all possible interpretations of a given sentence and choose the best one.

Reference

1. S.-S. Kang and C.-W.Woo, "Automatic Segmentation of Words Using Syllable Bigram Statistics," *Proc. 6th Natural Language Processing Pacific Rim Symp.*, Information Processing Soc. of Japan, 2001, pp. 729–732.

we use the tag sequences to reexpress equation 4 as equation 5. So, as equation 5 shows, the word-spacing model seeks to find the tag sequence $u_{1,n}$ for maximizing the joint probability $P(u_{1,n}, c_{1,n})$.

Output-first and input-first models

Consider a character to be an input variable, and a word-spacing tag to be a hidden variable and an output variable.

From equation 5, we derive the equations

$$\begin{aligned} & P(u_{1,n}, c_{1,n}) \\ & = \left(P(u_1) \times P(c_1 | u_1)\right) \\ & \quad \times \left(P(u_2 | u_1, c_1) \times P(c_2 | u_{1,2}, c_1)\right) \\ & \quad \times \begin{pmatrix} P(u_3 | u_{1,2}, c_{1,2}) \\ \times P(c_3 | u_{1,3}, c_{1,2}) \end{pmatrix} \\ & \quad \times \cdots \\ & \quad \times \begin{pmatrix} P(u_n | u_{1,n-1}, c_{1,n-1}) \\ \times P(c_n | u_{1,n}, c_{1,n-1}) \end{pmatrix} \end{aligned} \tag{6}$$

$$= \prod_{i=1}^{n} \begin{pmatrix} P(u_i | u_{1,i-1}, c_{1,i-1}) \\ \times P(c_i | u_{1,i}, c_{1,i-1}) \end{pmatrix} \tag{7}$$

$$\approx \prod_{i=1}^{n} \begin{pmatrix} P(u_i | u_{i-K,i-1}, c_{i-J,i-1}) \\ \times P(c_i | u_{i-L,i}, c_{i-H,i-1}) \end{pmatrix} \tag{8}$$

where K and L refer to the number of tags relative to the number of characters J and H in a sequence.

The joint probability becomes equation 6 by a chain rule. We call equations 6 through 8 the *output-first model* because the chain rule expands the output variable (word-spacing tag) first. In equation 8, we call the first probability *word-spacing tag probability* (or just *tag probability*) and the second *character probability*. The Markov assumption (conditional independence) we use in the tag probability is that the current tag u_i conditionally depends on only the previous K tags and the previous J characters. The Markov assumption we use in the character probability is that the current character c_i conditionally depends on only the previous L tags, the current tag, and the previous H characters.

Similarly, by expanding the input variable (character) first, we get equation 9 from equation 5:

$$P(u_{1,n}, c_{1,n}) = \prod_{i=1}^{n} \begin{pmatrix} P(c_i | c_{1,i-1}, u_{1,i-1}) \\ \times P(u_i | c_{1,i}, u_{1,i-1}) \end{pmatrix} \quad (9)$$

$$\approx \prod_{i=1}^{n} \begin{pmatrix} P(c_i | c_{i-K,i-1}, u_{i-J,i-1}) \\ \times P(u_i | c_{i-L,i}, u_{i-H,i-1}) \end{pmatrix} \quad (10)$$

We call equations 9 and 10 the *input-first model*.

Regardless of which variable we expand first, the models are identical because the chain rule is symmetrical. In fact, there's no difference between equations 7 and 9. However, because of the assumptions we use, the models become quite different. We think that the input-first model of equation 10 intuitively seems more natural than the output-first model. To distinguish between the two models, we denote equation 8 as $O(K, J, L, H)$ and equation 10 as $I(K, J, L, H)$. The HMMs usually used for POS tagging are special cases of the output-first model. (For example, the bigram HMM is $O(1, 0, 0, 0)$ and the trigram HMM is $O(2, 0, 0, 0)$.)

The larger the values of K, J, L, and H are, the more context we can consider.

However, selecting the proper values is important to avoid data sparseness and an excessive increase in the number of parameters, which would increase the memory the model requires.

To efficiently compute the most probable sequence of word-spacing tags, we use the Viterbi algorithm, as in the case of HMMs.[3]

Parameter estimation and smoothing

The output-first and input-first models can estimate the probabilities simply by using the *maximum-likelihood estimator*, which we can derive from the relative frequencies in the training data. To calculate the character and tag probabilities of $I(2, 2, 1, 2)$ (the model that performed best in our experiments) from equation 10 using the MLE, we use these equations:

$$\hat{P}(c_i | c_{i-2,i-1}, u_{i-2,i-1}) = \frac{freq(c_{i-2}u_{i-2}c_{i-1}u_{i-1}c_i)}{freq(c_{i-2}u_{i-2}c_{i-1}u_{i-1})}$$

$$\hat{P}(u_i | c_{i-1,i}, u_{i-2,i-1}) = \frac{freq(u_{i-2}c_{i-1}u_{i-1}c_iu_i)}{freq(u_{i-2}c_{i-1}u_{i-1}c_i)}$$

Because the MLE suffers from sparse data, probability estimates of low-frequency events might be inaccurate. To compensate, we use a linear interpolation from $I(2, 2, 1, 2)$, such as the equations

$$\begin{aligned} P(c_i | c_{i-2,i-1}, u_{i-2,i-1}) &= \alpha_1 \cdot \hat{P}(c_i | c_{i-2,i-1}, u_{i-2,i-1}) \\ &+ \alpha_2 \cdot \hat{P}(c_i | c_{i-1}, u_{i-2,i-1}) \\ &+ \alpha_3 \cdot \hat{P}(c_i | c_{i-1}, u_{i-1}) \\ &+ \alpha_4 \cdot \hat{P}(c_i | u_{i-1}) \end{aligned}$$

$$\begin{aligned} P(u_i | c_{i-1,i}, u_{i-2,i-1}) &= \beta_1 \cdot \hat{P}(u_i | c_{i-1,i}, u_{i-2,i-1}) \\ &+ \beta_2 \cdot \hat{P}(u_i | c_{i-1,i}, u_{i-1}) \\ &+ \beta_3 \cdot \hat{P}(u_i | c_i, u_{i-1}) \\ &+ \beta_4 \cdot \hat{P}(u_i | c_i) \end{aligned}$$

where α_i and β_i are nonnegative constants such that $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$ and $\beta_1 + \beta_2 + \beta_3 + \beta_4 = 1$. To calculate α_i and β_i directly, we use Thorsten Brants' algorithm.[13]

Experiments

For training, we used the 21st Century Sejong Project's (www.sejong.or.kr/english/eng_intro/intro_b7.htm) raw corpus of 26 million words. For evaluation, we used the ETRI (Electronics and Telecommunications Research Institute) POS tagged corpus of 288,269 words. To evaluate automatic word spacing, we modified the corpus to remove any word boundary information. Figure 1 presents the test data statistics.

We used three evaluation measures. *Character-unit precision* is

$$P_{\text{char}} = \frac{\text{the number of correctly spaced characters}}{\text{the total number of characters in the test data}}$$

Word-unit recall is

$$R_{\text{word}} = \frac{\text{the number of correctly spaced words}}{\text{the total number of words in the test data}}$$

Word-unit precision, which measures how accurate the system's results are, is

$$P_{\text{word}} = \frac{\text{the number of correctly spaced words}}{\text{the total number of words produced by the system}}$$

(Table 2 lists the abbreviations for these measures and some other abbreviations used later in this article, for quick reference.)

As we mentioned before, in Korean you can frequently either split a compound word into several words or concatenate several words into a compound word. For example, you can write the compound noun *jeong-bo-geom-saeg* (information retrieval) either as a whole or as two separated nouns *jeong-bo* (information) and *geom-saeg* (retrieval). To take into account such cases, we also performed evaluations that considered the compound nouns, which is the most prominent part of such cases.

Number of characters not spaced	638,546
Number of characters spaced (total number of words)	288,291
Total number of characters (token)	926,837
Number of unique characters (type)	2,456
Percentage of characters not spaced	68.90
Percentage of characters spaced	31.10

Figure 1. Test data statistics.

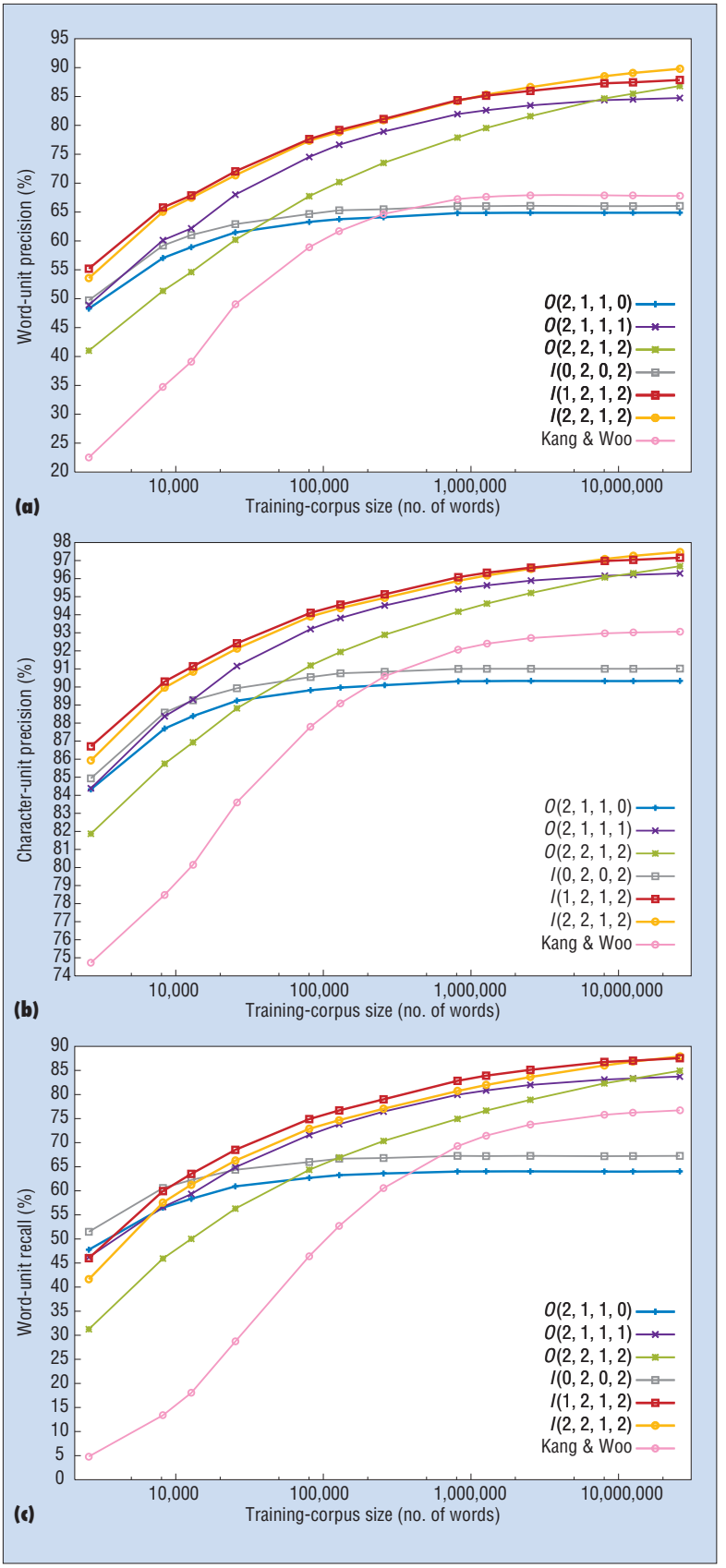


Table 2. Some notations used in this article.

Notation	Description
P_{char}	Character-unit precision
R_{word}	Word-unit recall
P_{word}	Word-unit precision
f_0	Number of characters where merge error occurred
f_1	Number of characters where segment error occurred
t_0	Number of characters successfully merged
t_1	Number of characters successfully segmented

General results

To investigate every model, we performed experiments for different values of K, J, L and H. Here, we restrict the values to $0 \leq K, J, L, H \leq 2$, which in turn restricts the number of possible distinct models to 81 (3^4). However, we didn't use the case of $(K, J) = (0, 0)$. Because we didn't yet know which model was best, we had to determine this through experiments. As a result, we actually tested 144 models, 72 for the output-first model and 72 for the input-first model.

Table 3 shows the results. For a fair comparison of the models, we didn't use the smoothing method in this experiment; instead, we used the MLE to estimate the models' parameters. (We show the results with smoothing later.) Owing to space restrictions, we don't list the results for all models; we show the best performers of the models using the same character n-grams. The table's last row (Kang & Woo) presents the results for Seung-Shik Kang and Chong-Woo Woo's statistical method (see the sidebar). For the evaluation that didn't consider compound nouns, we regarded the results as correct only when the system outputs were exactly the same as the original test data. For the evaluation that considered compound nouns, we allowed both splitting compound nouns and concatenating words into compound nouns.

The well-known HMMs performed poorly, as you can see for both $O(1, 0, 0, 0)$ (the bigram HMM) and $O(2, 0, 0, 0)$ (the trigram HMM). This suggests that choosing suitable models depending on a given problem is better than just using the standard HMMs.

On the basis of our results, we're sure that more con-

Figure 2. Accuracy according to the training-corpus size, without considering compound nouns: (a) character-unit precision, (b) word-unit recall, and (c) word-unit precision. *O* indicates an output-first model, *I* indicates an input-first model, and "Kang & Woo" indicates Seung-Shik Kang and Chong-Woo Woo's statistical method, which is described in the sidebar.

Table 3. Experimental results of some output-first (*O*) and input-first (*I*) models, with a comparison to Seung-Shik Kang and Chong-Woo Woo's statistical method (Kang & Woo), described in the sidebar. Bold numbers indicate the results for the best-performing model.

Model	No. of *n*-grams	Consider compound nouns: No P_{char}	No R_{word}	No P_{word}	Yes P_{char}	Yes P_{word}
O(1, 0, 0, 0)	1	85.75	47.05	48.96	87.41	51.61
O(2, 0, 0, 0)	1	86.18	50.25	51.42	87.73	53.80
O(2, 1, 1, 0)	1	90.34	64.04	64.90	91.71	67.41
O(2, 1, 1, 1)	2	96.29	83.73	84.74	97.25	88.01
O(2, 2, 1, 2)	3	96.69	84.93	86.82	97.59	89.78
O(2, 2, 2, 2)	3	96.04	82.05	83.96	96.88	87.08
I(0, 2, 0, 2)	1	91.02	67.25	66.06	92.28	69.02
I(1, 2, 1, 2)	2	97.16	87.56	87.85	98.02	91.49
I(2, 2, 1, 2)	3	**97.48**	**87.89**	**89.79**	**98.33**	**93.06**
I(2, 2, 2, 2)	3	97.39	87.69	89.31	98.25	92.57
Kang & Woo	2	93.06	76.71	67.80	94.01	71.22

Table 4. The 10 characters with the highest error rates, for the best-performing model, *I*(2, 2, 1, 2).

Character	Error rate (%)	f_0	f_1	t_0	t_1	Frequency
long	46.58	34	0	32	7	73
~	41.38	6	6	10	7	29
neg	40.00	0	4	6	0	10
=	36.84	12	2	10	14	38
ggwo	35.00	4	3	10	3	20
5	34.41	207	6	393	13	619
P	31.62	37	0	78	2	117
leung	30.00	3	0	6	1	10
0	29.73	298	13	718	17	1,046
R	28.21	10	1	28	0	39

texts a model considers, the better it performs. The best output-first model was *O*(2, 2, 1, 2), and the best input-first model was *I*(2, 2, 1, 2). However, the models using the largest context (*I*(2, 2, 2, 2) and *O*(2, 2, 2, 2)) weren't the most accurate. This discrepancy seems due to data sparseness. Our models significantly outperformed Kang and Woo's method, except for those using character unigrams. The input-first models performed better than the output-first models. When the experiments considered compound nouns, character-unit precision improved up to 2 percentage points and word-unit precision improved up to 5.8 percentage points.

The effect of training-data size

We wanted to answer two questions about word-spacing models:

- How much training data do we need to obtain a given model's best performance?
- Which model best fits a given training-data set?

To answer these questions, we compared various models' accuracy for different sizes of training data. This experiment didn't consider compound nouns. Figure 2 shows the results for character-unit precision, word-unit recall, and word-unit precision. *O*(2, 1, 1, 0) and *I*(0, 2, 0, 2) used character unigrams; *O*(2, 1, 1, 1), *I*(1, 2, 1, 2), and Kang & Woo used character bigrams; and *O*(2, 2, 1, 2) and *I*(2, 2, 1, 2) used character trigrams.

As the figure shows, the models using character unigrams converged quickly with small amounts of training data. The models using character bigrams converged with much larger amounts of training data. The models using character trigrams didn't converge at all. So, the possibility exists of improving the performance of the models using trigrams on larger training data. *O*(2, 2, 1, 2) performed worse than other models with small amounts of training data, because of data sparseness.

For the same character *n*-gram, input-first models consistently outperformed output-first models, regardless of the training-data size. This tells us that for the word-spacing problem, the input-first model used a more stable Markov assumption. Kang and Woo's model performed poorly in the word-unit evaluations because it considered only the surrounding character contexts and not the surrounding tag contexts. In contrast, because our proposed models used the Viterbi algorithm to find the most probable tag sequence, they effectively resolved this problem.

Error analysis

For the best-performing model, *I*(2, 2, 1, 2), table 4 lists the 10 characters with the highest error rates. Most errors were due to sparse data. Symbols, digits, Roman alphabet characters, and low-frequency characters (especially Chinese characters) had a high error rate. In addition, unknown words, and especially characters that were part of a foreign word, usually had high error rates. Of the 17 characters with 100-percent error rates, 13 appeared only once in the test data.

Table 5. The 10 most frequent characters, for the best-performing model, *I*(2, 2, 1, 2).

Character	Error rate (%)	f_0	f_1	t_0	t_1	Frequency
.	0.62	123	90	10,032	23,964	34,209
i	2.18	383	277	16,675	12,894	30,229
da	0.64	94	86	27,031	1,068	28,279
neun	0.68	103	53	1,798	20,996	22,950
e	0.85	89	50	6,588	9,673	16,400
eul	0.69	47	62	402	15,366	15,877
ui	0.75	78	36	2,084	13,019	15,217
ga	1.30	101	91	5,963	8,583	14,738
go	1.35	86	113	3,611	10,890	14,700
“	1.37	34	161	7,225	6,767	14,187

Table 6. Results of the best-performing model (*I*(2, 2, 1, 2)) with and without smoothing.

Consider compound nouns	Without smoothing			With smoothing		
	P_{char}	R_{word}	P_{word}	P_{char}	R_{word}	P_{word}
No	97.48	87.89	89.79	97.65	88.63	90.31
Yes	98.33	—	93.06	98.44	—	93.46

Table 5 lists the 10 most frequent characters for *I*(2, 2, 1, 2). High-frequency characters had error rates lower than the 2.52 percent average.

Many of the system’s errors also frequently occur in Korean text, and some are even allowed by the current spelling rules. For example, *do-wa-ju-neun* and *do-wa ju-neun* occurred three times each in the test data. In the case of the character *ggwo* in table 5, both concatenating and separating it are considered correct, such as *ba-ggwo-ju-da* or *ba-ggwo ju-da* and *ba-ggwo-beo-li-da* or *ba-ggwo beo-li-da*, so we think such a case isn’t an error in the strictest sense. To reflect these phenomena properly requires a less strict error-counting scheme, such as the one we used in the evaluation that considered compound nouns.

Improving the best-performing model with smoothing

Even though *I*(2, 2, 1, 2) was the best-performing model, it still had obvious trouble with data sparseness. To improve its performance, we applied our smoothing method. Table 6 compares this model with and without smoothing.

Considering the parameter space

Given that we use tag *m*-gram and character *n*-gram information, our models require T^nV^m parameters. The tag set size, *T*, is equal to 2 for the word-spacing problem, so it doesn’t much affect the parameter size. On the other hand, the character vocabulary size, *V*, greatly affects the parameter size. In theory, 11,172 pure Korean characters are possible. In practice, about 2,300 Korean characters are used. If we consider all the characters that appear in Korean texts (digits, symbols, Chinese characters, and so on), about 7,500 characters are possible. So, it’s implausible to use character *n*-grams that are larger than trigrams for Korean word spacing.

If we apply our models to a language with a small character set, such as English, using character *n*-grams that are larger than trigrams is possible and will provide better performance. This process would still require, as did our experiments, finding the best-performing model among the possible models in a given environment, such as language and training-data size.

Although we performed experiments only on Korean, our models are language independent. For languages having word spacing, the models need no modification. In languages without explicit word spacing, you can use the models for word segmentation, if you train them with annotated corpora instead of raw corpora.

We plan to improve the system’s performance for characters with high error rates and low frequency. Postprocessing by rules and clustering the characters should help.

The most difficult problem in word spacing is how to handle semantic ambiguities. The current models, which use only character *n*-gram frequency, can’t solve this problem. A more sophisticated analysis for semantic ambiguities remains a subject for further research. ■

Acknowledgments

This work was supported partly by grant R01-2006-000-11162-0 from the Korea Science & Engineering Foundation’s Basic Research Program and partly by the second stage of the BK-21 project.

References

1. E. Charniak et al., “Equations for Part-of-Speech Tagging,” *Proc. 11th Nat’l Conf. Artificial Intelligence* (AAAI 93), AAAI Press, 2003, pp. 784–789.
2. B. Merialdo, “Tagging English Text with a Probabilistic Model,” *Computational Linguistics*, vol. 20, no. 2, 1994, pp. 155–172.
3. L. Rabiner, “An Introduction to Hidden Markov Models,” *IEEE ASSP Magazine*, vol. 3, no. 1, 1986, pp. 4–16.
4. J.-H. Choi, “Automatic Korean Spacing Words Correction System with Bidirectional Longest Match Strategy,” *Proc. 9th Conf. Hangul and Korean Information Processing*, Korea Information Science Soc., 1997, pp. 145–151.
5. S.-S. Kang, “Automatic Word-Segmentation for Hangul Sentences,” *Proc. 10th Conf. Hangul and Korean Information Processing*, Korea Information Science Soc., 1998, pp. 137–142.
6. S.-S. Kang, “Eojeol-Block Bidirectional Algorithm for Automatic Word Spacing of Hangul Sentences,” *J. Korea Information Science Soc.*, vol. 27, no. 4, 2000, pp. 441–447.
7. K.-S. Kim, H.-J. Lee, and S.-J. Lee, “Three-Stage Spacing System for Korean in Sentence with No Word Boundaries,” *J. Korea Information Science Soc.*, vol. 25, no. 12, 1998, pp. 1838–1844.
8. Y.-M. Chung and J.-Y. Lee, “Automatic Word-Segmentation at Line-Breaks for Korean Text Processing,” *Proc. 6th Conf. Korea Soc. for Infor-*

mation Management, Korea Soc. for Information Management, 1999, pp. 21–24.

9. N.-Y. Jeon and H.-R. Park, "Automatic Word-Spacing of Syllable Bi-Gram Information for Korean OCR Postprocessing," *Proc. 12th Conf. Hangul and Korean Information Processing*, Korea Information Science Soc., 2000, pp. 95–100.

10. S.-S. Kang and C.-W.Woo, "Automatic Segmentation of Words Using Syllable Bigram Statistics," *Proc. 6th Natural Language Processing Pacific Rim Symp.*, Information Processing Soc. of Japan, 2001, pp. 729–732.

11. K. Shim, "Automated Word-Segmentation for Korean Using Mutual Information of Syllables," *J. Korea Information Science Soc.*, vol. 23, no. 9, 1996, pp. 991–1000.

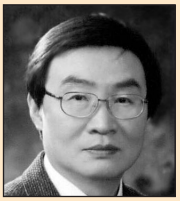
12. J.-H. Shin and H.-R. Park, "A Statistical Model for Korean Text Segmentation Using Syllable-Level Bigrams," *Proc. 9th Conf. Hangul and Korean Information Processing*, Korea Information Science Soc., 1997, pp. 255–260.

13. T. Brants, "TnT—A Statistical Part-of-Speech Tagger," *Proc. 6th Applied Natural Language Processing Conf.* (ANLP 00), Assoc. for Computational Linguistics, 2000, pp. 224–231.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.

The Authors

Do-Gil Lee is a research engineer at the NHN Corporation. His research interests include natural language processing, machine learning, and information retrieval. He received his PhD in computer science from Korea University. Contact him at Venture Town Bldg., 25-1 Jeongja-dong, Bundang-gu, Seongnam-si, Gyeonggi-do, Korea 463-844; dglee@nhncorp.com.



Hae-Chang Rim is a professor in Korea University's Department of Computer Science. His research interests are natural language processing, Korean language engineering, and information retrieval. He received his PhD in computer science from the University of Texas at Austin. He's a member of the Association for Computational Linguistics and a former president of the Korean Information Science Society's Korean Language Processing Special Interest Group. Contact him at the Dept. of Computer Science and Eng., Korea Univ., 1, 5-ka, Anam-dong, Seoul, 136-701, Korea; rim@nlp.korea.ac.kr.

Dongsuk Yook is a professor in Korea University's Department of Computer Science. His research interests are speech recognition, spoken-language understanding, machine learning, and AI. He received his PhD in computer science from Rutgers University. He's a member of the IEEE. Contact him at the Dept. of Computer Science and Eng., Korea Univ., 1, 5-ka, Anam-dong, Seoul, 136-701, Korea; yook@voice.korea.ac.kr.