Token Identification Using HMM and PPM Models

Yingying Wen[1,2], Ian H. Witten[2], and Dianhui Wang[3]

[1] School of Computer Science and Software Engineering
Monash University, Clayton, Victoria 3800, AUSTRALIA
ywen@csse.monash.edu.au
[2] Department of Computer Science
The University of Waikato, Hamilton, NEW ZEALAND
{yingying,ihw}@cs.waikato.ac.nz
[3] Department of Computer Science and Computer Engineering
La Trobe University, Victoria 3086, AUSTRALIA
dhwang@cs.latrobe.edu.au

Abstract. Hidden markov models (HMMs) and prediction by partial matching models (PPM) have been successfully used in language processing tasks including learning-based token identification. Most of the existing systems are domain- and language-dependent. The power of retargetability and applicability of these systems is limited. This paper investigates the effect of the combination of HMMs and PPM on token identification. We implement a system that bridges the two well known methods through words new to the identification model. The system is fully domain- and language-independent. No changes of code are necessary when applying to other domains or languages. The only required input of the system is an annotated corpus. The system has been tested on two corpora and achieved an overall F-measure of 69.02% for TCC, and 76.59% for BIB. Although the performance is not as good as that obtained from a system with language-dependent components, our proposed system has power to deal with large scope of domain- and language-independent problem. Identification of date has the best result, 73% and 92% of correct tokens are identified for two corpora respectively. The system also performs reasonably well on people's name with correct tokens of 68% for TCC, and 76% for BIB.

1 Related Research

Token identification task is to automatically identify the boundaries of a variety of phrases of interest in raw text and mark them up with associated labels. The systems reported in the Message Understanding Conference are limited to the following tokens: person, organization, location, date, time, money and percent. For us, however, the token identification task has no restriction—tokens are defined by a system designer and could encompass any type of information that is of interest.

Some learning algorithms have been reported such as decision trees, maximum entropy models and hidden markov models.

Sekine [1] and Bennett *et al.* [2] both implemented their token identification systems using decision trees. Their decision trees are based on almost identical features,

such as part-of-speech, character type information and special dictionaries. While the two systems are similar, there are significant differences between them. Another system using decision trees is proposed by Baluja *et al.* [3]. Like the systems described by both Sekine and Bennett *et al.*, they utilized a part-of-speech tagger, dictionary lookups, and word-level features, such as all-uppercase, initial-caps, single-character, and punctuation features.

Borthwick *et al.* [4] described a token identification system based on a maximum entropy framework. The system used a variety of knowledge sources, such as orthographic, lexical, section and dictionary features, to make tagging decisions. For any particular class label, there are four states: *label_start*, *label_continue*, *label_end* and *label_unique*. The first three states are for the case that more than one consecutive words are identified as the same class. The fourth is for the case that only one word is identified in a particular class. In addition, there is a special label—*other*, which indicates that the word is not part of a class. For example, the phrase "Jenny Bentley lives in Hamilton" is marked as "person_start, person_end, other, other, location_unique". One label is assigned to every word in the text. This approach is essentially the same as that described by Sekine [1]. Borthwick *et al.* employed Viterbi's [5] search algorithm to find the highest probability legal path. For example, label_end can only be assigned to a word that follows a word with either label_start or label_continue. The system is a purely statistical one, and contains no hand-generated patterns.

Another system for token identification that uses a maximum entropy model is reported by Mikheev *et al.* [6]. The model uses contextual features of tokens, for example the position of tokens in a sentence, whether they appear in lowercase in general, whether they were used in lowercase somewhere else in the same document and so on. This system makes decisions using the answers provided by the Maximum Entropy model.

IdentiFinder [7] is a well-known system. It uses a variant of a hidden Markov model to identify tokens like names, dates and numerical quantities. Each state of the HMM corresponds to a token class. There is a conditional state for "not a token class". Each individual word is assumed to be either part of some pre-determined class or not part of any class. According to the definition of the task, one of the class labels or the label that represent "none of the classes" is assigned to every word. IdentiFinder uses word features, which are language-dependent, such as capitalization, numeric symbols and special characters, because they give good evidence for identifying tokens.

This paper considers of the effect of the combination of HMMs and PPM on token identification. The system bridges the two well known methods through words new to the identification model. The main characteristics of the proposed system is that it is fully domain- and language-independent.

Two corpora, The Computists' Weekly—formerly known as The Computists' Communique (TCC).[4] and The Collection of Computer Science Bibliographies (BIB).[5], are employed to evaluate the techniques presented in this paper.

The rest of the paper is organized as follows. The next section describes the algorithms of the models. Section 3 demonstrates how the models are used in token iden-

[4] http://www.computists.com

[5] http://liinwww.ira.uka.de/bibliography/index.html

tification. Section 4 evaluates our proposed system. We conclude the paper in the last section.

2 HMMs and PPM

The idea of the system is to identify tokens from word-level to character-level when encounter unknown tokens. It is achieved by bridging HMM and PPM models. Due to limitation of space, we briefly describe the algorithms of the models in this section. Please refer to [8] and [9, 10] for more details.

2.1 HMMs

A hidden Markov model is a finite-state automaton with stochastic state transitions and symbol emissions [8]. It is a particular model based on a sequence of events, and consists of a set of states and a set of output symbols. The automaton generates a sequence of symbols by starting from the initial state, transitioning to a new state, emitting an output symbol, transitioning to another state, emitting another symbol, and so on, until the final state is reached and the last symbol is emitted.

For each member of the set of states, $S = \{S_1, S_2, ..., S_N\}$, there are two probability distributions. One governs the outgoing state transitions, which indicates how likely another state is to follow; the other governs the emission of symbols in the observation vocabulary $V = \{V_1, V_2, ..., V_M\}$, which indicates how likely a symbol is to be generated in the particular state. N and M are the number of states and number of symbols respectively.

We assume that time is discrete, and the model transitions between states at each time unit. In the case of a first-order Markov model, which is used in the undertaken research, the probability of moving from state S_i to state S_j is stored in the *state transition matrix*, $A = \{a_{ij}\}$, where:

$$a_{ij} = \mathtt{Pr}[q_t = S_j | q_{t-1} = S_i], \qquad 1 \leq i, j \leq N. \tag{1}$$

In this and future equations, t refers to the time instant, q_t is the variable that records the state assignment to the t^{th} symbol, and S_j, the j^{th} member of the set of possible states, is the assigned value. In other words, the probability of being in the current state is determined by the previous state.

When the HMM moves between states, it emits an output symbol after each transition. Exactly which output symbol is emitted depends on the *output symbol distribution* B, which defines the probability of emitting a particular symbol in a particular state. For first-order HMM, B is a two dimensional matrix defined as $B = \{b_j(k)\}$, where:

$$b_j(k) = \mathtt{Pr}[o_t = V_k | q_t = S_j], \qquad 1 \leq j \leq N, \quad 1 \leq k \leq M. \tag{2}$$

Here, o_t is the variable that records the t^{th} symbol emission, and V_k, the k^{th} member of the observation vocabulary, is the emitted symbol.

To complete the model we need an initial probability distribution $\pi = \{\pi_i\}$ over states, where:

$$\pi_i = \mathtt{Pr}[q_1 = S_i], \qquad 1 \leq i \leq N. \tag{3}$$

Let us assume that an HMM model has been constructed for a particular kind of sequence, and we are presented with a new example of such a sequence, $O = o_1, o_2, ..., o_T$. The problem of finding the most likely state sequence $Q = q_1, q_2, ..., q_T$ that produces the given symbol sequence is called *decoding*. There are several possible ways of solving this problem. We have used *Viterbi* algorithm [5, 11], which is to recover the state sequence that has the highest probability of having produced the given observation sequence.

2.2 PPM models

Models that take a few immediately preceding symbols into account to make a prediction are called *finite-context* models of order m, where m is the number of preceding symbols used [9]. The PPM technique uses finite-context models of characters [10]. It is a so-called character-level model. It uses the last few characters in the input string to predict the upcoming one. By considering such a context, each character can be better predicted. The prediction is done by using the counts of occurrences of each context. The probabilities associated with each character that has followed the context are used to predict the probability for the upcoming character.

PPM uses fixed-order context models with different values of m, up to a predetermined maximum. The maximum number is a given constant, which is call the *order* of the model. The bigger the order, the more information is considered. But increasing the order does not guarantee better compression, because the contexts become rarer as the order grows.

Several orders are blended together in PPM to obtain a good probability estimate for the current character. The prediction starts with a given maximum order m and checks the occurrence of the order m context. If the order m context has occurred with the upcoming character following it, the corresponding counts are used to predict the probability. If the context has not been seen in the past, the model then uses the order $m - 1$ context.

Consider the case where the context has occurred but never followed by the upcoming character. This is called the *zero-frequency* situation [12]—the character will be predicted using a zero count. In this case, a special transmission called *escape* is used to drop the model down one order, and the order $m - 1$ model is used to make the prediction.

Another possible situation is that the character has never occurred in the past—an unknown character. Then even order 0 cannot be used. This is another instance of the zero-frequency problem. The model then escapes down to a bottom-level model, order -1, that predicts all characters equally.

To illustrate the PPM modeling technique, Table 1[13] shows the four models with order 2, 1, 0 and -1 after the string $tobeornottobe$ has been processed.

In Table 1, c represents the occurrence, esc and p are probabilities for escape and symbol, respectively. They are determined by the following equations. $|\mathrm{A}|$ is the size of the alphabet.

$$esc = \frac{t}{2n}, \tag{4}$$

Order 2				
Prediction			*c*	*p*
be	→	o	1	1/2
	→	*esc*	1	1/2
eo	→	r	1	1/2
	→	*esc*	1	1/2
no	→	t	1	1/2
	→	*esc*	1	1/2
ob	→	e	2	3/4
	→	*esc*	1	1/4
or	→	n	1	1/2
	→	*esc*	1	1/2
ot	→	t	1	1/2
	→	*esc*	1	1/2
rn	→	o	1	1/2
	→	*esc*	1	1/2
to	→	b	2	3/4
	→	*esc*	1	1/4
tt	→	o	1	1/2
	→	esc	1	1/2

Order 1				
Prediction			*c*	*p*
b	→	e	2	3/4
	→	*esc*	1	1/4
e	→	o	1	1/2
	→	*esc*	1	1/2
n	→	o	1	1/2
	→	*esc*	1	1/2
o	→	b	2	3/8
	→	r	1	1/8
	→	t	1	1/8
	→	*esc*	3	3/8
r	→	n	1	1/2
	→	*esc*	1	1/2
t	→	o	2	1/2
	→	t	1	1/6
	→	*esc*	2	1/3

Order 0			
Prediction	*c*	*p*	
→	b	2	3/26
→	e	2	3/26
→	n	1	1/26
→	o	4	7/26
→	r	1	1/26
→	t	3	5/26
→	*esc*	6	3/13

Order –1			
→	A	1	1/|A|

Table 1. PPM after processing the string $tobeornottobe$.

$$p = \frac{2c - 1}{2n}, \tag{5}$$

where t is the distinct number of characters that have followed a particular context, n is the number of times a context has appeared.

The model in Table 1 is used as follows. Suppose the character following $tobeorn$ $ottobe$ is o. Since the order-2 context is be, and the upcoming symbol has already been seen once in this context, the order-2 model is used and the probability is $1/2$. If the next character, instead of o, were t, this has not been seen in the current order. Consequently an order-2 escape probability of $1/2$ is used and the context is truncated to the order-1 context e. Again it has not been seen in this context, so an order-1 escape probability of $1/2$ is used and the context is truncated once more to the null context, corresponding to order 0. Finally the character t is predicted with a probability of $5/26$. Thus the prediction of t is done in three steps, using order 2 to order 0 context respectively, with a probability of $1/2 \times 1/2 \times 5/26$. If the upcoming character had been x instead of t, a final level of escape to order -1 would have occurred with a probability of $3/13$, and x would be predicted with a probability of $1/256$ (assuming that the alphabet $|\mathrm{A}| = 256$).

		Context			
		tobeornottobe		*nottobeorto*	
		Probability	Model used	Probability	Model used
Upcoming character	*o*	½	Order 2	½×½×5/6	Order 0
	t	½×½×5/26	Order 0	½×1/6	Order 1

Table 2. Effect of context and current character with order-2 PPM.

The probabilities predicted by PPM are based on the occurrences of the prior context and the characters that have followed each context every time the context has occurred in the training text. Table 2 shows how the previous context being processed and current character affect the result in terms of the order of model and probabilities by using the same prior contexts to predict different characters and vice versa.

3 Token Identification

3.1 HMM based approach

For the token identification application, the observation sequence is a sequence of words in text. The symbols emitted in each state are words, and the HMM is a word-level model.

In the system, each sequence corresponds to a sentence in text, and each state corresponds to a type of token that the program will identify and mark up. Example token class include people's names, geographical locations, monetary amounts and e-mail addresses. Each type of token will be marked in the text by a unique tag. N, the number of states in the model, is the number of different token classes, and is determined by the training data. Because the system uses a word-level HMM model, M, the size of the output vocabulary, is the number of different words that appear in the training data.

The matrix A in HMM gives the probability that the current word belongs to a particular token type given that the previous word belongs to a particular token type as well. We also call it the *contextual probability*. Distribution B is the probability of the same words being seen in a particular token class. It is token-dependent: different token classes have different probabilities for a certain word. B is also called the *lexical probability*. The initial distribution π is the probability that each type of token starts a sentence.

For instance, in the following sentence, a fragment in annotated version of The Computists' Weekly, 1998,

<o>Polytechnic University</o> in <l>Brooklyn</l> will get
<m>$190M</m> from the <n>Othmer</n> estate, about four
times the school's previous endowment.

The sequence of four words "Polytechnic", "University", "in" and "Brooklyn" contributes to the four-element class sequence <o><o><p><l>. It contributes the probabilities of transitioning from organization (o) to organization, state o to o; organization to plain text (p), o to p; and plain text to location (l), p to l. Thus probabilities are given by the elements of matrix A. The words themselves will be counted as the appearances in the corresponding token class to make up the elements of matrix B, for example, words "Polytechnic" and "University" labeled as organization would increase the counts for their occurrences in this class. "Polytechnic" as part of organization would also increase the probability of token <o> starting a sentence.

The model is trained on the training set of the corpus. The system processes the training data in two passes. The first pass counts the number of token classes, N, the number of different words, M, and the vocabularies for each token type. The second pass counts the number of events and calculates the A and B matrices.

For illustration, Figure 1 is an example of an HMM obtained from part of the training data. It is annotated with transition probabilities and token labels: email address (e), dates (d), people's name (n), sources (s), organizations (o), URLs (u), locations (l), money (m) and plain text (p). The figure shows that it is possible for words in the plain text (p) class to follow words in any other token classes, and these words can also be followed by words in any other class except email (e). It is reasonable that words in all token classes can be surrounded by plain text. The location class (l), monetary class (m)

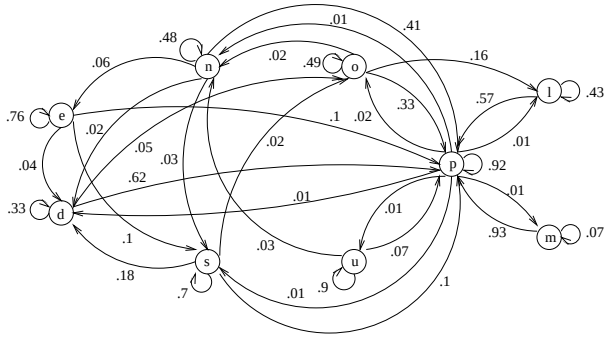

Fig. 1. A HMM obtained from the training data.

and URL (u) class have no direct relationship between each other. They never appear one after another and always have tokens in other classes between them. This is understandable from the grammatical point of view. Probabilities from plain text to some other token class, such as date, source and location, are very low. This is not because the events are rare but because they are overwhelmed by plain text words, which makes the denominator bigger and results in smaller numbers.

The figure indicates that there are no tokens in fax and phone classes in this particular set of training data, because classes, along with vocabularies, depend on the training data.

3.2 Unknown word handling

Unknown words are those that have not been encountered in training data. There are two kinds of unknown word: neologisms, and existing words that happen not to appear in the training data.

One of the main goals of token identification is to choose the correct label in cases where a word can have more than one label assignment. Additionally, a system must deal with words that have not been encountered in the training data, and so are not found in the lexicon.

The lexicon for the HMM is built during training, so the model contains all words. If an unknown word is encountered during decoding, there is no entry in the model. The emission probability in the state transition matrix B is unknown. To ensure that the process continues and works in a proper way, some policy must be adopted to estimate the probability that the current state will emit the given unknown word.

A PPM model is then constructed for each token class, using all tokens in a class in the training data. Whenever a word that has not been encountered in training is seen, and is therefore not contained in the lexicon, the value of $b_j(k)$ is assigned the probability that is predicted by an appropriate PPM model. The more words in a class that occur in the training data, the more likely it is that tokens in the same class can be identified in new text.

4 Performance Evaluation

4.1 Measurement Metric

Three standard measures, *recall*, *precision* and *F-measure* [14, 15], along with *error-rate* are used to evaluate the accuracy of the token identification system. They are calculated by using the corresponding manually marked-up fragment in the training corpus as the gold standard. For easy reference, let us call this gold standard *hand mark-up*. To define them, the following terms are used:

N	Number of tokens occurring in the standard text;
c	Number of tokens correctly marked up by the system;
e	Number of tokens incorrectly marked up by the system;
$n = c + e$	Number of tokens marked up by the system.

The measures take into account two aspects of the mark-up: the label itself, and the boundary where the label is inserted. A token is considered to be correctly marked up when both label and boundaries are correct. For example

> The board has been begging and bribing <n>Steve Jobs</n> to stay on, but he hasn't accepted yet.

"Steve Jobs" is correctly marked as a person's name and it contributes one count to c.

Recall and precision are widely used to assess the quality of an information retrieval system in terms of how many of the relevant documents are retrieved (recall) and how many of the retrieved documents are relevant (precision). In the token identification task, recall is the proportion of the correct tokens which are actually identified by the system, while precision is the proportion of tokens identified by the system which are correct. They are written as:

$$\text{Recall} = \frac{c}{N}, \tag{6}$$

$$\text{Precision} = \frac{c}{n}. \tag{7}$$

The two measures do not always provide an adequate evaluation because there are some extreme situations where one of them is very small while the other is large. For example, if the system identifies few tokens compared to the number of N and they are all correct, recall will be very small whereas precision is 100%. It is better to have a measure that yields a high score only when recall and precision are balanced. A widely used measure is the F-measure [14, 15]:

$$\text{F-measure} = \frac{(\beta^2 + 1) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \tag{8}$$

where values of β between 0 and ∞ give varying weights to recall and precision. In this paper, $\beta = 1$, gives equal importance to recall and precision, therefore, F-measure is the harmonic mean of recall and precision:

$$\text{F-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{9}$$

The measure of error-rate is used just for easy analysis of the result. It is defined as

$$\text{Error-rate} = \frac{e}{N}. \tag{10}$$

This is normally used on its own as an overall indicator of the quality of identification. However, it can give misleading results—an extreme condition is where the system only identifies a single word, leading to a very small error-rate of $1/N$ despite the fact that all tokens but one remain unidentified.

If the system marks up the same number of tokens as the hand mark-up, recall and precision both become equal to one minus the error-rate. A perfect identification system will have an error-rate of zero, and recall and precision of 100%.

4.2 TCC corpus

The first series of evaluations used the TCC corpus. The corpus was first split into sentences because the system processes input sentence by sentence. The available TCC corpus contains 38 issues of the newsletter. 25 issues, which are randomly selected, are used as the training set.

To see how the system works, 5 of the remaining issues are used as the test set with all the tags removed. PPM models are used for the unknown words.

Table 3 shows the results for each of the five test files in terms of recall, precision, F-measure and error-rate. It also gives the average values of these measures.

File	**Recall** (%)	**Precision** (%)	**F-measure** (%)	**Error-rate** (%)
test1	64.86	72.37	68.41	24 76
test2	65.67	74.58	69.84	22.39
test3	68.72	78.34	73.21	18.99
test4	60.96	74.79	67.17	20.55
test5	66.26	66.67	66.46	33.13
Average	65.29	73.35	69.02	23.96

Table 3. Result for five TCC test files.

In the table, the value of precision is always higher than that of recall. This indicates that the system marks less tokens than it should do. Consequently, some tokens are either missed out—left unmarked, or merged into another token by the system.

Figure **??**(a) depicts the proportion of correctly marked tokens (dark gray) and errors (light gray) over the corresponding numbers of tokens in the hand mark-up for each class and overall.

The figure shows that organization class has a very poor result, with only about 20% of correct tokens identified. This is probably due to the lower percentage of words in this class in the training data. As mentioned before, the more words in a class that occur in the training data, the more likely it is that tokens in the same class can be identified in new text. The proportion of words in classes such as fax number and sums of money are

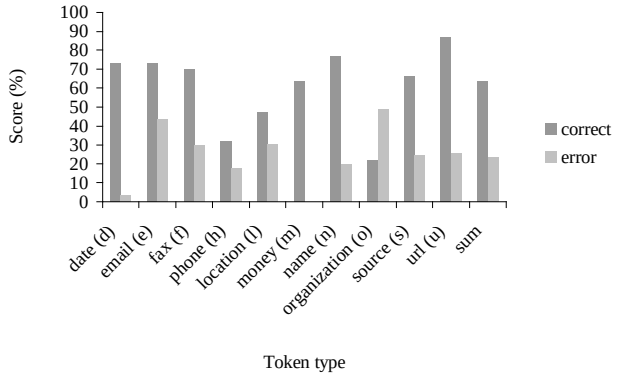

Fig. 2. Detailed result for five TCC test files.

also small, however, they have special so-called indicators, for example “fax” and “$”. This increases the performance for these classes. However, a name of an organization is more likely to comprise diverse words, such as:

<o>Bell Labs</o>
<o>John Benjamins</o>
<o>Holland Academic Graphics</o>

The percentage of correct tokens in date, email and fax classes are about the same. However, the system is more successful on date class because the error rate is very low. This is understandable because date has much less ambiguity than email address and fax number. Most tokens in date class are distinguishable in both words and format. Errors happen in special cases such as “30-<d>40 years</d>” and word stemming. But they are rare. On the other hand, email addresses can be mixed up with URLs due to the definition of a word. Some sources such as “<s>comp.ai.genetic</s>” also increase the error in email address and URL classes. There is no distinction between fax number and phone number except the word “fax/Fax”, which is a strong indicator to distinguish fax number from any other classes.

It is interesting to notice that a token which is a money amount is either marked up correctly or left unmarked. Due to the format of the token in this particular class, there is no marking ambiguities.

Generally speaking, name of a person is relatively harder to identify than other classes. The figure shows that it gets the second best result for both the correct ratio and error ratio. As we have noticed, there are some names that occur repeatedly. For example, “Ken”, “Bill Park” and “Brandon” have occurred several times in the entire corpus and none of them has been found to be identified incorrectly. However, most of the names are not in this special case.

4.3 Bibliographies

The text in bibliographies is more structured than that in the TCC newsletter, which the model takes advantage of. A bibliography entry contains name, title and date. Most

of them have page number(s), and some provide organization, publisher, location and source. In the experiments, the BIB corpus of 2400 references were selected randomly from the bibliography collection. The model is trained on 2200 entries, and the experiments are done by running the system on 100 of the remaining references with labels removed.

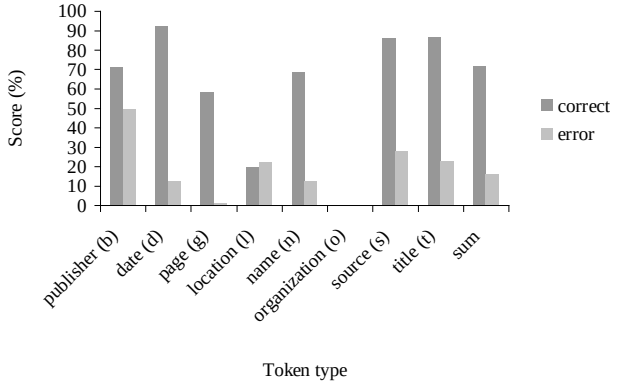

Fig. 3. Detailed result for 100 bibliographic entries.

Figure **??**(b) shows the proportion of correctly marked tokens (dark gray) and errors (light gray). As we can see, tokens in date class have the best identification result. From the figure, not a single token is marked up as an organization whether correct or not. It is because that there are only 4 organizations in the test data. 3 of them are marked up incorrectly in the hand mark-up and one is left unmarked.

Corpus	**Recall** (%)	**Precision** (%)	**F-measure** (%)	**Error-rate** (%)
BIB	72.02	81.78	76.59	16.05
TCC	65.29	73.35	69.02	23.96

Table 4. Average result for 100 bibliographic entries and 5 TCC issues.

Table 4 shows the average value of recall, precision, F-measure and error-rate for 100 bibliographic entries. It also shows the average result of 5 TCC issues for comparison. Overall, the system was successfully applied on the new text. The figure indicates that the performance is 7.57% better in F-measure than that of TCC.

5 Conclusion

The token identification system described in this paper combines two well-known language models through unknown words for developing a domain- and language-independent system. The system has been tested on two data sets without any changes

in the code. The results show that 69.02% of F-measure for TCC and 76.59% for BIB are achieved. Although the performance is not as good as that obtained from a system which includes language-dependent components, our system has power to deal with large scope of domain- and language-independent problem. The performance evaluation is made by accuracy, however, the scope of the system's applicability should be taken into account. There is always a trade-off between this two issues. The more specific the application, the higher accuracy a system provides. This research emphasizes retargetability and generality. It is understandable that the system is degraded because of the lack of language dependence. Experiment results demonstrate that the presented techniques in this paper perform reasonably well on date and people's name for TCC and BIB.

References

1. Sekine, S.: NYU: Description of the Japanese NE system used for MET-2. In: Proceedings of MUC-7 1998. (1998)
2. Bennett, S.W., Aone, C., Lovell, C.: Learning to tag multilingual texts through observation. In: Proceedings of the Second Conference on Empirical Methods in Natural Language Processing, Providence, Rhode Island (1997) 109–116
3. Baluja, S., Mittal, V.O., Sukthankar, R.: Applying machine learning for high performance named-entity extraction. In: Proceedings of the Conference of the Pacific Association for Computational Linguistics, Waterloo, CA (1999) 365–378
4. Borthwick, A., Sterling, J., Agichtein, E., Grishman, R.: Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In: Proceedings of the Sixth Workshop on Very Large Corpora, Montreal, Canada (1998)
5. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theroy **IT-13** (1967) 260–269
6. Mikheev, A., Moens, M., Grover, C.: Named entity recognition without gazetteers. In: Proceedings of EACL, Bergen, Norway. (1999)
7. Bikel, D.M., Schwartz, R., Weischedel, R.M.: An algorithm that learns what's in a name. Machine Learning Journal **34** (1999) 211–231
8. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proceedings of the IEEE **77** (1989) 257–286
9. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes-Compressing and Indexing Documents and Images. 2 edn. ISBN 1-55860-570-3. Morgan Kaufmann Publishers, San Francisco, California (1999)
10. Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Trans on Communications **32** (1984) 396–402
11. Forney, J.G.D.: The Viterbi algorithm. Proceedings of the IEEE **61** (1973) 268–278
12. Witten, I.H., Bell, T.C.: The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. IEEE Transactions on Information Theory **37** (1991) 1085–1093
13. Teahan, W.J., Wen, Y., McNab, R., Witten, I.H.: A compression-based algorithm for Chinese word segmentation. Computational Linguistics **26** (2000) 375–393
14. Van Rijsbergen, C.J.: Information Retrieval. Second edn. Butterworths, London (1979)
15. Lewis, D.D.: Evaluating and optimizing autonomous text classification systems. In: Proceedings of the Eighteenth Annual International ACM Special Interest Group on Information Retrieval. (1995) 246– 254