Comparing Winnow and RIPPER in Thai Named-Entity Identification

Boonserm KIJSIRIKUL
Department of Computer Engineering Chulalongkorn University
Phayathai Road, Bangkok, 10330, Thailand.
email: boonserm@cp.eng.chula.ac.th

Paisarn CHAROENPORNSAWAT and **Surapant MEKNAVIN**
National Electronics and Computer Technology Center
22nd Gypsum Metropolitan Tower, 539/2 Sriayudhaya Road,
Rajthevi, Bangkok, 10400,Thailand.
email: pcharoen@notes.nectec.or.th, surapan@nectec.or.th

Abstract

This paper presents an application of two machine learning algorithms, i.e., Winnow and RIPPER, and their comparison on the task of Thai named-entity identification. While most of previous works on this task are based on hand-coded rules, we use learning algorithms to help automate the development of named-entity system. Since Thai language has no explicit word boundary, Thai name is much more difficult to identify than in other languages such as English. Given an input text, we first generate candidates for names by segmenting the text into several different sequences of words and non-words. This is done by using dictionary and statistical information; candidates for names can be non-words not found in the dictionary, or words that appear in the dictionary but tend to be names according to statistical information. These candidates are then determined whether they are names by learning algorithms previously trained on a corpus. The experimental results showed the effectiveness of Winnow and RIPPER, and demonstrated that Winnow is superior to RIPPER in our task.

1 Introduction

Named-entity identification is important to Natural Language Processing (NLP) tasks, such as information retrieval, part-of-speech tagging, etc. Most of previous works on named-entity systems are based on hand-coded rules (Appelt et al., 1995; Weischedel, 1995; Gaizauskas et al., 1995; Iwanska et al., 1995) which require linguistic expertise and consume a lot of effort to develop. In recent years, there has been increasing interest in application of machine learning techniques to NLP. The techniques have been shown to be successfully applied to various NLP tasks such as Thai word segmentation (Meknavin, et al., 1997), part-of-speech tagging (Brill, 1995), parser construction (Zelle and Mooney, 1996), etc. An advantage of the techniques is that they are able to reduce human effort by helping automate the development of NLP systems through training from the data.

This paper presents an application of two learning algorithms on the task of Thai named-entity identification. To identify names in languages that have no explicit word boundary such as Thai, Japanese, etc., it is necessary to segment the input text into words and non-words. If a name is not correctly segmented, its occurrence can make the surrounding words incorrectly recognized and itself may be wrongly identified.

We divide Thai names into three types; (1) *explicit names*, (2*) partly hidden names*, and (3) *fully hidden names*. Based on each type, we provide an algorithm to generate candidates for a name. These candidates are then classified whether they are names by learning algorithms. The learning algorithms applied in our task are Winnow and RIPPER. To train the algorithms, we employ two types of features that are *context words* and *collocations*. The idea is to have the algorithms learn the features that characterize the contexts in which each named-entity tends to occur.

The paper is organized as follows. Section 2 states the characteristics of Thai named-entity. Section 3 gives the algorithms for generating name candidates. Section 4 briefly describes Winnow and RIPPER. In Section 5, the overview of our named-entity identification system is given. Section 6 shows the preliminary experiment comparing Winnow and RIPPER. We then conclude the paper in Section 7.

2 Characteristics of Thai Named-Entities

The Named-Entity task in MUCs consists of three subtasks for recognizing entity names, temporal expressions and number expressions (Grishman, 1995). In this paper we focus on the entity names which are categorized into person, organization and location names.

While the Named-Entity tasks for several languages such as English, Japanese, etc. have been conducted for a long period, the Named-Entity task for Thai was started as an experimental task in 1997. There are many characteristics in Thai that make the task difficult:

- Unlike most of European languages, word forms provide no direct cue in identifying Thai names. For example, there is no capitalization in Thai.
- A large portion of Thai names are general Thai words in the dictionary. For example, นก (bird), ประดิษฐ์ (invent) and อภิสิทธิ์ (exclusive right) are common names in Thai. So checking for the nonexistence in the dictionary alone is not an effective method to detect Thai names.
- There is neither word boundary nor sentence boundary delimiter. This complicates almost every process related to text processing, and makes finding the range of named-entities extremely difficult.

UKU	*UKK*	*KUKK*	*UK*	*KU*	*KKK*

U = an unknown string, *K* = a known word

Figure 1 :Various forms of Thai names

Figure 1 illustrates various forms of Thai names that are formed by the combination of known words and unknown strings (non-words). According to these forms, Thai names are classified into three categories.

I. Explicit names. An explicit name is a name of which no any substring is in a dictionary. Examples of explicit names are listed below.

- กพ. (abbreviation of an organization name)
- สุนีย์ (a woman name)
- กฤษณ์ (a man name)

II. Partially hidden names. A partially hidden name is composed of known words and unknown strings. For example,

- ไมโครซอฟต์ → ไม **โค** ร **ซอ** ฟต์
  (Microsoft) ไม **(ox)** ร **(fiddle)** ฟต์
- สุรพันธ์ → สุร **พัน** ธ์
  (Surapant) สุร **(one thousand)** ธ์
- นูสกิน → นูส **กิน**
  (NuSkin) นูส **(to eat)**

where "→" means "is composed of", bold strings are known words, and the words in the parentheses are the corresponding English words. These are examples of person and company names.

III. Fully hidden names. A fully hidden name is composed solely of one or more known words such as

- บุญเสริม → **บุญ** **เสริม**
  (Boonserm) **(merit)** **(to strengthen)**
- กนกพร → **กนก** **พร**
  (Kanokporn) **(gold)** **(a gift)**
- ไพศาล → **ไพศาล**
  (Paisarn) **(wide)**

Table 1 shows the distribution of each name type from our 25K-words corpus. As can be seen, nearly 50% (908/1887) of names are fully hidden names. This means that simple dictionary checking can detect only half of Thai names. Moreover, among those that can be detected, only 13% (130/(130+849)) are explicit names that need no further processing to identify the correct name boundaries. Consequently, if the simple method is employed that detects names by finding unknown strings not found in a dictionary and just considers those strings as names, only 7% recall (130/1887) and 13% precision (130/(130+849)) will be obtained. This suggests that other better methods for name detection and boundary identification are needed.

To cope with these characteristics of Thai names, we propose the methods to generate name candidates below.

Table 1: Statistics of Thai names in 25K-words corpus

	Explicit Name	Partially Hidden Name	Fully Hidden Name	Total
Person	57	491	293	841
Organization	1	273	452	726
Location	72	85	163	320
Total	130	849	908	1887

3 Generating Candidates of Named-Entity

We propose two heuristics to generate candidates for names according to their types. After all candidates are generated, the best candidate will be selected by Winnow or RIPPER. Winnow and RIPPER will be described in the next section.

3.1 Handling Explicit & Partially Hidden Names

In the case that a string does not exist in the dictionary, candidates of the name will be created by merging the words around the unknown string and the string itself into a new string. All combinations of +/- k words around the unknown string are used to generate candidates. In the following experiment, k is set to 3. We can define the equation for creating name candidates as in Figure 2.

In case that there are many unknown strings, the nearby unknown strings will be grouped into a single unknown string for being used as a candidate, if they are separated by a word having less than three characters.

$$\text{Text} = w_1 w_2 ... w_a U w_b ... w_n$$
where $w_i \in$ Dictionary, $U \notin$ Dictionary
n = number of words in the text.

$$NM = \{ \alpha U \beta \mid \alpha \in A, \beta \in B \}$$
where NM = set of candidates for names,
$$A = \{ w_{a-i,a}, i \in [0,k] \} \cup \{\varepsilon\},$$
$$B = \{ w_{b,b+i}, i \in [0,k] \} \cup \{\varepsilon\},$$
$$w_{i,j} = w_i \ldots w_j : i \le j,$$
ε = null string, k = constant value.

Figure 2: Equation for generating explicit and partially hidden name candidates

3.2 Handling Fully Hidden Names

On the other hand, if every word is in the dictionary, it is more difficult to detect names.

Let Text = $w_1 w_2 \ldots w_n$ be an input text, w_i be a word in the text, and t_i be the part of speech of the word w_i. The word that will be selected as a name candidate is:

- the word that has $P(w_i|t_i)$ less than a threshold, or
- the word that has $P(t_i|t_{i-1},t_{i-2})$ less than a threshold.

In case that $P(w_i|t_i)$ is less than a threshold, w_i will be considered as a name candidate. In case that the probability $P(t_i|t_{i-1},t_{i-2})$ of w_i is less than a threshold, not only w_i but also w_{i-1} and w_{i-2} will be considered as name candidates because the less-than-threshold probability of w_i may come from w_{i-1} or w_{i-2}. We can define the equation of creating name candidates as in Figure 3.

$$\text{Text} = w_1 w_2 ... w_a ... w_{n-1} w_n$$
where $w_i \in$ Dictionary,
n = number of words in the text,
w_a is a word whose probability is less than the threshold.

$$NM = \{ \alpha W \beta \mid \alpha \in A, \beta \in B \}$$
where NM = set of name candidates,
$$A = \{ w_{a-i,a-1}, i \in [1,k] \} \cup \{\varepsilon\},$$
$$B = \{ w_{a+1,a+i}, i \in [1,k] \} \cup \{\varepsilon\},$$
$$w_{i,j} = w_i \ldots w_j : i \le j,$$
$$W = w_a : P(w_a \mid t_a) < \theta \quad \text{or}$$
$$W \in \{w_{a-2}, w_{a-1}, w_a\} : P(t_a \mid t_{a-1}, t_{a-2}) < \theta,$$
ε = null string, θ is the threshold,
k = constant value.

Figure 3: Equation for generating fully hidden name candidates

4 Winnow & RIPPER Algorithms

This section briefly describes learning algorithms Winnow and RIPPER used in our tasks, and defines features for training the algorithms.

4.1 Winnow

Winnow algorithm used in our experiment is the algorithm described in (Blum, 1997). Winnow is a neuron-like network where several nodes are connected to a target node. Each node called *specialist* looks at a particular value of an attribute of the target concept, and will vote for a value of the target concept based on its specialty; i.e. based on a value of the attribute it examines. The global algorithm will then decide on weighted-majority votes receiving from those specialists. The pair of (attribute=value) that a specialist examines is a candidate of features we are trying to extract. The global algorithm updates the weight of any specialist based on the vote of that specialist. The weight of any specialist is initialized to 1. In case that the global algorithm predicts incorrectly, the weight of the specialist that predicts incorrectly is halved and the weight of the specialist that predicts correctly is multiplied by 3/2. The weight of a specialist is halved when it makes a mistake even if the global algorithm predicts correctly.

4.2 RIPPER

RIPPER is a propositional rule learning algorithm that constructs a ruleset which classifies the training

data (Cohen, 1995). A rule in the constructed ruleset is represented in the form of a conjunction of conditions:

if T_1 and T_2 and ... T_n then class C_x.

T_1 and T_2 and ... T_n is called the body of the rule. C_x is a target class to be learned; it can be a positive or negative class. A condition T_i tests for a particular value of an attribute, and it takes one of four forms: $A_n = v$, $A_c \geq \theta$, $A_c \leq \theta$ and $v \in A_s$, where A_n is a nominal attribute and v is a legal value for A_n; or A_c is a continuous variable and θ is some value for A_c that occurs in the training data; or A_s is a set-value attribute and v is a value that is an element of A_s. In fact, a condition can include negation. A set-valued attribute is an attribute whose value is a set of strings. The primitive tests on a set-valued attribute A_s are of the form "$v \in A_s$". When constructing a rule, RIPPER finds the test that maximizes *information gain* for a set of examples S efficiently, making only a single pass over S for each attribute. All symbols v, that appear as elements of attribute A for some training examples, are considered by RIPPER.

4.3 Features

To train both algorithms to learn the name, the context around the name is used to form features. The features used are the context words and collocations. Context words are used to test for the presence of a particular word within +10 words and –10 words from the target word. Collocations are patterns of up to 2 contiguous words and part-of-speech tags around the target word. Therefore the total number of features is 10; two features for context words, and eight features for collocations.

5 An overview of the system

Our algorithm for identifying names consists of four steps as follows:

5.1 Word Segmentation

For each input sentence, probabilistic trigram model (Meknavin, et al., 1997) is applied to separate the sentence into a sequence of words and non-words and assign their parts of speech, and N-best segmented sentences are then selected as candidates. The probabilistic trigram model, that generates the N highest probable sentences, can be described formally as following:

Let $C = c_1c_2\ldots c_m$ be an input character string, $W_i = w_1w_2..w_n$ be a possible word segmentation, and $T_i = t_1t_2..t_n$ be a sequence of parts of speech. Find $W_1,W_2...W_N$ which are the N-highest probability of sequence of words. We can compute $P(W_i)$ in the following fashion:

$$P(W_i) = \sum_T P(W_i, T_i) = \sum_T \prod_i P(t_i \mid t_{i-1}, t_{i-2}) * P(w_i \mid t_i) \quad (1)$$

where $P(t_i|t_{i-1}\ t_{i-2})$ and $P(w_i|t_i)$ are computed from the corpus.

For example, let C = "บริษัทนูสกินจัดตั้งตัวแทนจำหน่าย". This sentence should be segmented as "บริษัท นูสกิน จัดตั้ง ตัวแทน จำหน่าย" which means "NuSkin company establishes sale agencies". The results of our word segmentation algorithm of which the format is $[w_1/t_1]$ $[w_2/t_2]$ … $[w_n/t_n]$ are shown as follows:

I. [บริษัท/t_1] [**นูส**/ t_2] [กิน/t_3] [จัดตั้ง/t_4] [ตัวแทน/t_5] [จำหน่าย/t_6]

II. [บริษัท/t_1] [**นูส**/ t_2] [กิน/t_3] [จัด/t_4] [ตั้ง/t_5] [ตัวแทน/t_6] [จำหน่าย/t_7]

The word **นูส** is an unknown string and t_i is an appropriate part-of-speech tag of each word.

5.2 Generating Candidates for Names

From the results of step 5.1, generate all candidates for names by the explicit and hidden name heuristics described in Section 3.

For example, the sentence (I) from step 5.1, we found that the second string, นูส, is a non-word. Therefore we use the method for handling explicit and partially hidden names which is explained in Section 3. Name candidates are นูส, บริษัทนูส, นูสกิน, นูสกินจัดตั้ง, บริษัทนูสกิน and บริษัทนูสกินจัดตั้ง; where k, U, A and B in Figure 2 are 2, นูส, {$\mathcal{E}$, บริษัท} and {$\mathcal{E}$, กิน, กินจัดตั้ง}, respectively. Every sentence will be processed in the same way, and all candidates will be obtained.

5.3 Tagging Part of Speech

New sentences will be formed by combining candidates that are obtained from step 5.2 with the rest of the words in the original sentence. The part of speech of the words in each sentence will be reassigned by the trigram tagger. The name candidates are assumed to be the proper noun. Part of speech trigram can be defined as the following:

Let W be a sequence of words $w_1..w_n$ and T_i be a sequence of part-of-speech tags $t_1..t_n$. Find τ that maximizes $P(T_i \mid W)$:

$$\begin{aligned}\tau &= \arg\max_{T_i} P(T_i \mid W) \\ &= \arg\max_{T_i} P(t_i \mid t_{i-1}, t_{i-2}) * P(w_i \mid t_i)\end{aligned} \quad (2)$$

For example, in sentence (I) from step 5.1, name candidates are created by the procedure in step 5.2. In this step, the new sentences obtained are shown as follows:

III. [บริษัท/t_1] [**นูส**/NPRP] [กิน/t_3] [จัดตั้ง/t_4] [ตัวแทน/t_5] [จำหน่าย/t_6]

IV. [**บริษัทนูส**/NPRP] [กิน/t_2] [จัดตั้ง/t_3] [ตัวแทน/t_4] [จำหน่าย/t_5]

V. [บริษัท/t_1] [**นูสกิน**/NPRP] [จัดตั้ง/t_3] [ตัวแทน/t_4] [จำหน่าย/t_5]

VI. [บริษัท/t_1] [**นูสกินจัดตั้ง**/NPRP] [ตัวแทน/t_3] [จำหน่าย/t_4]

VII. [**บริษัทนูสกิน**/NPRP] [จัดตั้ง/t_2] [ตัวแทน/t_3] [จำหน่าย/t_4]

VIII. [**บริษัทนูสกินจัดตั้ง**/NPRP] [ตัวแทน/t_2] [จำหน่าย/t_3]

where t_i is a part-of-speech tag and *NPRP* is a proper noun tag. The sentence (II) can be processed in the similar manner.

5.4 Predicting by Winnow or RIPPER

Sentences from step 5.3 are sent to Winnow or RIPPER to be ranked, and the sentence with the highest score will be selected as the answer.

Given the sentences in step 5.3, Winnow and RIPPER select the best-score sentence that is the sentence (V) (บริษัท **นูสกิน** จัดตั้ง ตัวแทน จำหน่าย) which gives the correct meaning.

6 Preliminary Experiment

6.1 Data

To test the performance of our approach, we select sentences which contain names from our 25K-words corpus to use in benchmark test. Every sentence is manually separated into words and their parts of speech are manually tagged by linguists. The resulting corpus is divided into two parts; the first part, about 80% of corpus, is utilized for training and the rest is used for testing. To measure the performance of our approach, we classify the test data into two groups according to the method in Section 3. The first group contains explicit and partially hidden names, and the other contains fully hidden names.

6.2 Training Methodology

Using our tagged corpus, we train Winnow and Ripper by the following processes.

- Select sentences that contain names from the corpus.
- Construct positive examples. The names are considered as target words. The context words and collocations of the names are sent to Winnow and RIPPER.
- Construct negative examples. Incorrect candidates of names are considered as target words. The context words and collocations of the incorrect candidate are sent to Winnow and RIPPER. The method for creating incorrect candidates of names are described below:
  - -- Separate the sentences into words using trigram word segmentation algorithm.
  - -- Generate name candidates excluding the right candidates.

6.3 Results

Table 2 and Table 3 summarize the results of our experiment. Table 2 shows the percentage of name detection by our method (Dictionary + Trigram) compared to simple dictionary checking (Dictionary). Table 3 shows the precision and the recall obtained by Winnow and RIPPER.

Table 2: Percentage of name detection

	Explicit & Partially Hidden Names		Fully Hidden Names	
	Training Set	Test Set	Training Set	Test Set
Dictionary	100.00	100.00	0	0
Dictionary + Trigram	100.00	100.00	87.82	83.25

Table 3 : Performance of Winnow & RIPPER

	Training Set		Test Set	
	Precision (%)	Recall (%)	Precision (%)	Recall (%)
Winnow	100.00	94.02	100.00	93.58
RIPPER	100.00	86.68	96.43	85.94

The percentage of name detection indicates the effectiveness of the equations in Figure 2 and Figure 3 for detecting name candidates. As shown in Table 2, explicit and partially hidden names can be

completely detected but fully hidden names can be detected about 87% for the training data and 83% for the test data. The result implies that some fully hidden names cannot be detected by the equation in Figure 3 as their statistical information ($P(w_a|t_a)$ and $P(t_a|t_{a-1},t_{a-2})$) are not lower than the threshold. Note that simple dictionary checking cannot detect this type of names. Based on name candidates generated by the equations, the precision and recall of Winnow and RIPPER are shown in Table 3. The results show that Winnow and RIPPER are effective algorithms for identifying names which achieved high precision and recall. The results also show that Winnow is superior to RIPPER. The precision on our test corpus obtained by Winnow is 100%. This perfect result could be because of the small size of our corpus. One advantage of RIPPER over Winnow is that the RIPPER rules consume less memory than the Winnow network does. In our experiment, the number of RIPPER rules is 27, whereas the number of specialists of Winnow network is 112,966.

7 Conclusion

In this paper, we have proposed an application of Winnow and RIPPER to named-entity identification. Winnow and RIPPER both have been shown to be effective algorithms for identifying names, and Winnow is superior to RIPPER in this task. Moreover, the precision obtained by Winnow perfectly reaches 100%. However, as in Thai words are written consecutively without boundary delimiters, the accuracy of detecting boundary of fully hidden names is still not very high. In future work, we plan to find more effective method to detect and identify fully hidden names.

References

Appelt T., Hobbs J., Bear J., Israel D., Kameyama M., Kehler A., Martin D., Myers K. and Tyson M. 1995. SRI international FASTUS system MUC-6 test results and analysis. *In Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, NIST, Morgan-Kaufmann Publishers.

Brill E. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4), 543-565.

Blum A. 1997. Empirical support for Winnow and weighted-majority algorithm: Results on a calendar scheduling domain, *Machine Learning*, 26:5-23.

Cohen W. 1995 Fast effective rule induction, *In Proceedings of the Twelfth International Conference on Machine Learning*, Lake Taho, California, Morgan Kaufmann.

Gaizauskas R., Wakao T., Humphreys K., Cunningham H. and Wilks Y. 1995. University of Sheffield: Description of the LaSIE system as used for MUC-6. *In Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, NIST, Morgan-Kaufmann Publishers.

Grishman R. and Sundheim B. 1995. Design of the MUC-6 evaluation. *In Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, NIST, Morgan-Kaufmann Publishers.

Iwanska L., Croll M., Yoon T. and Adams M. 1995. Wayne state university: Description of the UNO natural language processing system as used for MUC-6. *In Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, NIST, Morgan-Kaufmann Publishers.

Meknavin S., Charoenpornsawat P. and Kijsirikul B. 1997. Feature-based Thai word segmentation. *In proceeding of the Natural Language Processing Pacific Rim Symposium 1997*, pp.41-46.

Weischedel R. 1995. BBN: description of the PLUM system as used for MUC-6. *In Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, NIST, Morgan-Kaufmann Publishers.

Zelle J.M. and Mooney R.J. 1996. Comparative results on using inductive logic programming for corpus-based parser construction. *In Wermter S., Riloff E. and Scheler G. (Eds.), Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, pp. 355-369. Springer, Berlin.