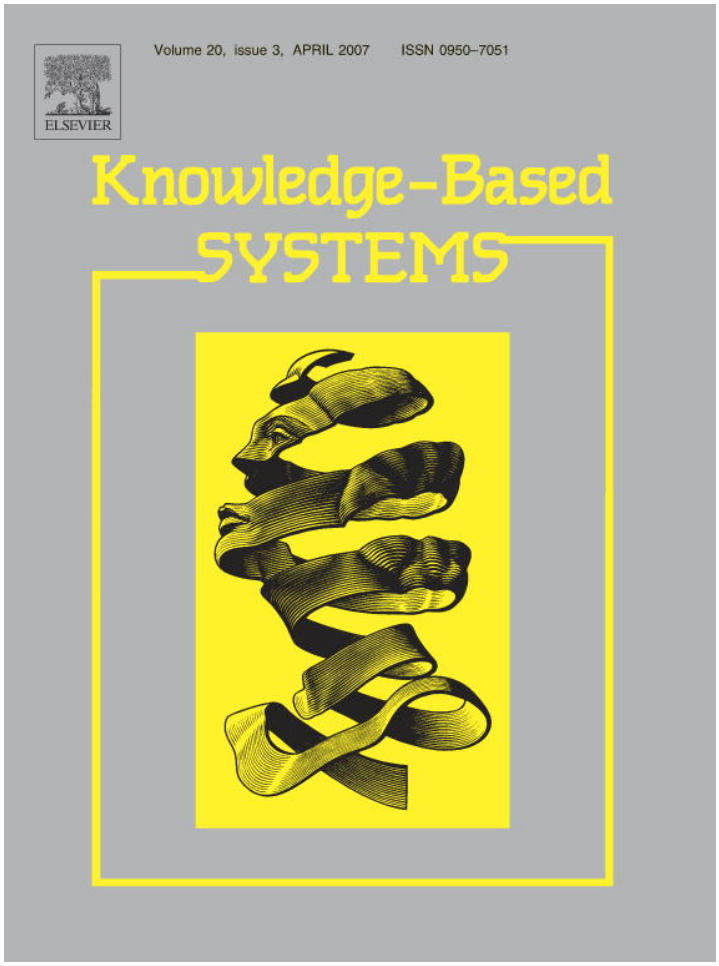

Available online at www.sciencedirect.com
ScienceDirect
Knowledge-Based Systems 20 (2007) 209–219
Knowledge-Based SYSTEMS
www.elsevier.com/locate/knosys

Efficient text chunking using linear kernel with masked method

Yu-Chieh Wu *, Chia-Hui Chang

Department of Computer Science, National Central University, 320, Jhongli City, Taoyuan, Taiwan

Received 6 July 2005; accepted 10 April 2006
Available online 30 August 2006

Abstract

In this paper, we proposed an efficient and accurate text chunking system using linear SVM kernel and a new technique called masked method. Previous researches indicated that systems combination or external parsers can enhance the chunking performance. However, the cost of constructing multi-classifiers is even higher than developing a single processor. Moreover, the use of external resources will complicate the original tagging process. To remedy these problems, we employ richer features and propose a masked-based method to solve unknown word problem to enhance system performance. In this way, no external resources or complex heuristics are required for the chunking system. The experiments show that when training with the CoNLL-2000 chunking dataset, our system achieves 94.12 in $F_{(\beta)}$ rate with linear. Furthermore, our chunker is quite efficient since it adopts a linear kernel SVM. The turn-around tagging time on CoNLL-2000 testing data is less than 50 s which is about 115 times than polynomial kernel SVM.

Keywords: Text chunking; Support vector machines; Shallow parsing; Masked method

1. Introduction

Automatic text chunking aims to recognize non-overlap phrases in a given sentence. Non-overlap phrases are phrases that are not included in other chunks [1]. A chunk is usually meaningful and contains more than one word. For example, a phrase ''Natural Language Processing'' is much less ambiguous than its individual component words ''Natural'', or ''Language'', or ''Processing''. Noun Phrase chunk (NP-chunk) recognition is similar to the text chunking problem, although it focused solely on extracting noun phrases in texts rather than all other chunk types.

Chunk information is important to many Natural Language Processing (NLP) studies, like shallow parsing [18,20], full parsing [27,26], clause identification [3,26], Semantic Role Labeling (SRL) [5], text categorization [22], Named Entity Recognition (NER) [10], and question and answering [31]. In particular for parsing and SRL tasks, the performance of the final result is greatly influenced by the effect of the chunker. In Q/A tasks, chunkers also faced the challenges of huge datasets. Thus, accurate and efficient text chunkers are needed.

Text chunking problem can be viewed as a sequence of word tagging [23], which assigns the optimal tag sequence for an input sentence. Traditionally, a set of human-made rules is used to extract all of the phrases in a text. Such rule-based systems can usually perform well on a specific domain, but are not applicable to other domains. Therefore, Machine Learning (ML) approaches are proposed for learning such rules in order to predict new data without human intervention. In recent, many machine learning-based chunking systems are presented, e.g., Transformation-Based Learning (TBL) [23], Hidden Markov Model (HMM) [19,34], Winnow [32,33], memory-based learning [13,26,28], voted-perceptrons [4], and voted-SVMs [16,17].

Zhang et al. [33] proposed a text chunking system involving Winnow algorithm which employed an external parser that was called English Slot Grammar (ESG). The learning speed of Winnow is quite fast (about 12 min). It achieves the best performance on the Computational Natural Language Learning (CoNLL) chunking dataset (94.17 in $F_{(\beta)}$ rate). However, using parsers is not the real purpose

* Corresponding author.
E-mail addresses: bcbb@db.csie.ncu.edu.tw (Y.-C. Wu), chia@csie.ncu.edu.tw (C.-H. Chang).

0950-7051/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2006.04.016

of chunking tasks. The chunker generates a phrase-level granule to the parser, which constructs dependency and tree structures of the sentence. Without ESG, their generalized-Winnow chunker obtained 93.57 $F_{(\beta)}$ rate. Kudoh and Matsumoto [16] developed an SVM chunker with the polynomial kernel type. Their system attained the highest performance for CoNLL-2000 shared task (93.48 $F_{(\beta)}$ rate). However, the training time is long and tagging speed is slow since the kernel is polynomial. It costs 24 h to train the system with CoNLL-2000 chunking data and identify 20 tokens per second. They further combined eight different single chunkers to improve system performance (from $F_{(\beta)}$ rate 93.48 to 93.91) in their later study [17]. Moreover, both [26] using Majority voting with five memory-based learners, and [12], employing Weighted Probability Distribution Voting models, presented different frameworks to combine several memory-based classifiers. The $F_{(\beta)}$ rates are 92.50 and 93.32, respectively. Similar to Kudoh's later results, the combination of several classifiers performs better than an individual system. Although the combinations of multiple classifiers and integration of external resources can relatively enhance the chunking performance, the use of multi-classifiers can largely increase the tagging speed as well as complicate the system design. In addition, external resources are not always available in many domains and languages.

In general, the efficient chunking systems, like HMM, and TBL, are not high-performanced, although they could handle large data set in few seconds. On the contrary, the high-performanced chunkers, for example, SVM, and voted-perceptrons, are not efficient since they employed non-linear kernels. Fewer of the current text chunkers are both accurate and efficient. To remedy this impact, we present a linear SVM-based chunker in terms of efficiency and performance. Both training and testing time are largely reduced when compared with previous kernel-method approaches since we employ a simple linear kernel SVM instead of more complex kernels. Theoretically speaking, the linear kernel type is the simplest and most efficient kernel to perform similarity calculation; while other kernel types, like polynomial, require more computation time for training and testing. Thus, the linear kernel type is a better choice for developing efficient learners.

Another observation is that many tagging errors occurred when the words were unknown to the training set. When unknown word occurs during testing, the lexical-related features, like unigram, are missed. The chunk class of the unknown word is determined by other non-lexical-related features, like Part-of-Speech (POS) tags. In general, the lexical feature is more distinct than non-lexical features. Even though unknown words had been identified and labeled with POS tags via POS taggers, these features are too general for guessing unknown word in chunk task. To overcome this problem, we propose a simple masked method for collecting unknown word examples. These examples are derived from existing training instances by masking some known words. In this way, we provide new training examples that do not contain complete lexical information. Therefore, when the learning algorithm is trained with these additional examples, the system performance can be improved. In our experiments, when the chunker is trained with only known examples, the $F_{(\beta)}$ rate is 93.53; and when trained with unknown word instances, it achieves 94.12 $F_{(\beta)}$. With the help of the unknown word examples and a simple linear kernel to SVM, our chunker achieves the state-of-the-art performance within the satisfactory time and at a reasonable cost.

The remainder of this paper is organized as follows. In Section 2, we discuss text chunking task. Section 3 explains our SVM chunking model and the feature selection techniques. The proposed masked method will be discussed in Section 4. Experimental results on benchmarking corpus (CoNLL-2000 shared task) and another large corpus are shown in Section 5. Concluding remarks and future work are given in Section 6.

2. Text chunk and base NP-chunk

Chunking is generally considered as a shallow parsing [1]. In 1995, Ramshaw and Marcus [23] used transformation-based error driven learning to derive transformation rules for tagging the Noun Phrase (NP) chunk. They regarded finding out chunks in a text as a sequence of word tag classification. Ramshaw and Marcus defined the representation of chunking task that is called B–I–O tags. The I tag denotes the word insides a chunk. The O tag denotes a word that is outside a chunk. Finally, the B tag denotes the first word of the second noun phrase indicating the boundary between two immediate chunks.

In addition to the B–I–O representation, Tjong Kim Sang and Veenstra [30] proposed three other chunk tag representations: IOB2, IOE1, IOE2, and renamed the Ramshaw and Marcus's B–I–O tag representation as IOB1. IOB2 is a variant type of IOB1. In IOB2 tagging method, the B tag is always the beginning of a noun phrase, and I tag is used to indicate the words inside a chunk. IOE1 can be viewed as another variant type of IOB1 where E tag is used to indicate the ending word of a phrase immediately followed by the next phrase, while the use of E tag in IOE2 is similar to that of B tag in IOB2, where the E tag is used to indicate the end of any noun phrase regardless whether it is followed by another chunk or not. Table 1 shows the four representation styles of the above example.

In addition to NP chunks, there are other chunk types as shown in Table 2 [29]. The goal of text chunking is to identify the optimal chunk tag sequence for a given sentence, i.e., assigning a suitable class for each word. Tjong Kim Sang and Buchholz [29] used the IOB2 representation style to indicate the boundaries between chunks. According to the POS tag set of the Wall Street Journal, they derived 11 chunk types for CoNLL-2000 text chunking task as shown in Table 2. Details of each chunk have been described by Tjong Kim Sang and Buchholz [29]. For example, the B-PP means the beginning of a prepositional phrase, and the I-VP is the

Table 1
The four representation styles of base NP chunking

Word	POS tag	IOB1	IOB2	IOE1	IOE2
The	DT	O	O	O	O
Security	NN	I	B	I	I
Authority	NN	I	I	I	I
Robert	NNP	I	I	I	I
L.	NNP	I	I	I	I
Duston	NNP	I	I	I	E
Favors	VBZ	O	O	O	O
Disciplining	VBG	O	O	O	O
All	DT	I	B	I	I
Employees	NNS	I	I	E	E
Who	WP	B	B	I	E
Cheat	VBP	O	O	O	O
.	.	O	O	O	O

Table 2
Various chunk types

Notation	Meaning
ADJP	Adjective phrase
ADVP	Adverb phrase
CONJP	Conjunction phrase
INTJP	Interjection
LST	List marker
NP	Noun phrase
PP	Prepositional phrase
PRT	Particles
SBAR	Subordinated clause
UCP	Unlike coordinated phrase
VP	Verb phrase

interior word of a verb phrase. Totally, there are 11(chunk types) × 2(B/I tags) + 1(O tag) = 23 chunk tags used. Table 3 shows the IOB2 tags of a text chunking example. The POS tag of each word is shown as well.

2.1. Related works

In this section, we compare the performance of various approaches to the CoNLL-2000 chunking shared tasks in terms of system performance and the time efficiency for training and testing. The surveyed approaches include Generalized/Regularized-Winnow, voted-SVMs, SVMs, voted-perceptions, WPDV, and memory-based chunkers. The performances of these systems are summarized in Table 4.

Table 3
The IOB2 chunk representation

Word	POS tag	Chunk tag (IOB2)
Fear	CC	B-NP
Of	IN	B-PP
The	DT	B-NP
Price	NN	I-NP
Police	NN	I-NP
Could	MD	B-VP
Help	VB	I-VP
Cool	JJ	I-VP
Things	NNS	B-NP
Off	IN	B-PRT
In	IN	B-PP
The	DT	B-NP
1990s	CD	I-NP
.	.	O

2.1.1. G/R-Winnow systems

Zhang et al. [33] derived two versions (generalized and regularized) of the Winnow algorithm. They used a similar way for deriving a "soft-margin" version of the algorithms for non-separable problems. They employed the standard representation style (IOB2) and used the first and second order feature sets. The total dimension of their chunking system is about 460,000 where the number of non-zero features per datum is 48. Surprisingly, the training time is only 12 min which is shown to be more efficient than other chunkers, like SVM or voted-perceptrons. They achieved the state-of-the-art chunking performance ($F_{(\beta)} = 94.17$ for generalized Winnow and $F_{(\beta)} = 94.13$ for regularized Winnow) among other systems. However, the best results were contributed by integrating an external parser, English Slot Grammar (the improvement is about 0.6). As described above, the cost of constructing such a chunker is not cheap. When the ESG parser is excluded, the performance is decreased to 93.57 in $F_{(\beta)}$ for generalized Winnow.

2.1.2. SVM systems

Kudoh and Matsumoto [16] proposed an SVM-based chunking system. In their research, they used the IOB2 representation style and trained 231 ($C_2^{23} = 231$) SVMs in so-called "One-Again-One" type with polynomial kernels (degree = 2). Their chunking systems were developed from the polynomial kernel since they use fewer feature types. However, the training time for their system was about 24 h. In their latter study [17], they reported another high-performance chunker (voted SVMs) by combining eight individual chunking systems (four representation styles multiplied by forward/backward processing directions) and achieved 93.91 in $F_{(\beta)}$ value. In other words, there are 8×231 SVMs in their chunking system. It is worth to note that the $F_{(\beta)}$ rate of an individual chunker of theirs with IOE2 style and backward processing direction is 93.85. Because the training time taken by SVMs scales exponentially with the number of input examples [14,15], it may not be an efficient choice in practice for developing learners.

2.1.3. Voted-perceptrons systems

The voted-perceptrons chunking system was proposed by Carreras et al. [4]. In their chunking systems, two-pass processing is used where pass one identifies the boundaries (both start and end) for each chunk class and pass two disambiguates the phrase type. In their studies, each of the perceptron employed the polynomial kernel [6] instead of the original linear kernel type. The performance of the two-pass voted-perceptrons reached 93.74 in $F_{(\beta)}$ rate. They did not report the training and testing time. Analytically,

Table 4
A selection of previous results for CoNLL-2000 chunking task

System name		Recall	Precision	$F_{(\beta)}$
Generalized Winnow [33]	Without ESG	93.60	93.54	93.57
	With ESG	94.07	94.28	94.17
Regularized Winnow [32]	Without ESG	93.49	93.53	93.51
	With ESG	94.01	94.24	94.13
Voted-SVMs [17]		93.89	93.92	93.91
SVMs [16]		93.51	93.45	93.48
Voted-perceptrons [4]		93.38	94.20	93.79
WPDV [12]		93.51	93.13	93.32
Memory-based model [26]		91.00	94.04	92.50

the dual form of perceptron learning is similar to the SVM but the perceptron classifier is simpler than the SVM. Nevertheless, both training and time cost are not cheap, where the testing phase relies on comparing with all of the support vectors of each category. As reported by Kudo and Matsumoto, they spent at least one day for training a single SVM with the polynomial kernel type. Thus, to train two-pass voted-perceptrons for chunking, the time cost is similar to that for training SVMs.

2.1.4. WPDV and memory-based systems

The combined approaches of memory-based chunking systems are very similar to voted-SVMs chunkers. There were two multiple classifiers architecture: WPDV [12] and majority-voted method [26]. The former used five memory-based learning models [8] with weighted probability distribution voting method to determine the final output from the five sources. The latter approach used the same learner but trained with four different B–I–O chunk representation styles and another O + C type. In comparison, the WPDV seemed to outperform the majority-voted method in text chunking (93.32 $F_{(\beta)}$ rate compared with 92.50 $F_{(\beta)}$ rate). The two memory-based chunking systems are not high-performance even they combined multiple classifiers. The multiple-based chunking systems involve training and testing cost several times higher and more complicate than a single-based system. However, as reported by [17] each of the individual SVM-based chunker achieved at least 93.48. Thus, we work toward single-learner chunking systems.

3. SVM-based chunking model

This section describes the proposed chunking system developed from SVM using a linear kernel. SVM learns to predict new items by training with these vector sets. The system is divided into two parts: the known part and unknown word instances (see Fig. 1. Training processing for the proposed chunker). We describe the known part in this section and leave the unknown part to Section 4.

3.1. Example representation

One of the advantages of the SVM with a linear kernel is that it can handle high dimensional data effectively [14,15] since it compares the "active" features rather than the complete dimensions. We can therefore impose richer feature types upon each training example to enhance system performance. As reported [11,3], the richer feature set had shown to be more effective than the simple feature set. In general, contextual information is often used as the seed feature type [2,9,17]), the other features can then be derived from the surrounding words. For example, let W_i be the current word, W_{i-1} be the word before W_i, and W_{i+1} be the word after W_i. To classify W_i, we use the words and features surrounding W_i. For example, when we tag the word "police", the context words are:

the (W_{i-2}) price (W_{i-1})**police**(W_i) could (W_{i+1}) help (W_{i+2})

If we include two words surrounding the current word, both the preceding two words and following two words should be considered. Therefore, a vector comprises five sections, which are equal to the locations of the contextual words. In this paper, we set the context window size to be 2, which is consistent with previous researches [16,32,33]. Using a good feature selection technique not only eases the searching for important or discriminative examples, but also increases the accuracy for classification tasks [25]. The SVM chunker proposed by [16] encoded only unigram, POS tag, and chunk information but require a polynomial kernel to perform well. As mentioned by Giménez and Márquez [11], high-quality POS tagging performance can be achieved by employing a richer feature set, such as OrthoFlags, Affixes, *N*-grams, to represent the meaning and relations of these words. Therefore, we adopt more features while using a linear kernel to reduce training time.

- Lexical information (unigram and bigram). Lexical feature is the most commonly used feature in NLP for a word itself carries certain meanings. For example, the words "Mr" or "Mrs" are good lexicons indicating the start of a noun phrase, while the words "was" and "were" usually precede a verb phrase. Bigram lexical information is generally considered more informative than unigram lexicons. In the vector representation, each unigram/bigram dimension indicates the presence or absence of a unigram/bigram. The unigram/bigram lexical dictionary is the set of frequent unigrams/bigrams in the training data.

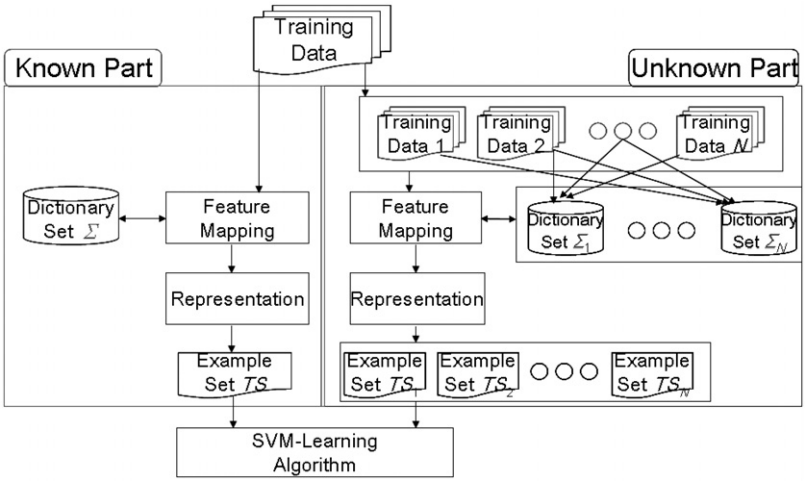

Fig. 1. Training processing for the proposed chunker.

- POS tag information (Uni-POS, Bi-POs, and Tri-POS). POS tags are other informative features used in NLP, since they show more abstracted and syntactic relations. The size of the tag set is 48, i.e., there are 48 different POS tags. We adopt Uni-POS, Bi-POS and Tri-POS tags. For the above example, the Bi-POS tags of the word "police" include "NN(price) NN(police)", and "NN(police) MD(could)". Similarly, the Tri-POS tags are "DT(the) NN(price) NN(police)", "NN(price) NN(police) MD(could)", and "NN(police) MD(could) VB(help)".
- Affix (2–4 suffix and prefix letters). The affix feature is widely used for guessing an unseen word from the training examples [9,11]. For this feature type, we extract different length of prefix/suffix letters in the head/tail of a word. We select length 2-, 3-, 4-letters from the beginning and ending of a given word. For example, the affix features of "police" are:
  prefix: "po", "pol", and "poli"
  suffix: "ce", "ice", and "lice".
- Token feature. Token feature type is a language-dependent feature type that is commonly used in named entity recognition. It recognizes by specific symbols and punctuations, such as "$", or "%". For example, the word "1990s" matches the token category "Year decade" (see Table 5). Each term will be assigned to a token class type via the pre-defined word category mapping.
- Previous chunk information (uni/bi-chunks). This feature type is generated by including previous word chunk tags. Most chunking systems use this feature type to help find the context relations between previous chunks and the current word. In this paper, we include both uni-chunk and bi-chunk in our feature set.
- Possible chunk classes. For the current word to be tagged, we recall its possible chunk tags in the training data and use its possible chunk class as a feature.

As an example, Fig. 2 shows the vector representation for word W_t. In general, each feature type has its own dictionary that collects homogeneous items. For example, the unigram dictionary contains "frequent" unigrams in the training set and the dictionary of bigram collects "frequent" bigrams from the training data. Here, the term "frequent" means that the terms in the dictionary appear at least M (=2 in this paper) times in the training set. For those that are not frequent, we simply discard them since they are rare. The feature mapping process is to determine the vector representation for each word. The total number of dimensions for the CoNLL-2000 chunking task is about

Table 5
Token feature categories

Feature description	Example text	Explanation
1 Digit number	9	Digital number
2 Digit number	20	Two-digit year
4 Digit number	1997	Four-digit year
Decade	2010s	Year decade
Only digit	432	Misc. numbers
Number + one slash	55/4	Amount or date
Number + two slash	2005/7/7	Date
Number + money	$222	Money
Number + percent	90%	Percent
Number + hyphen	3–4	Number period
Number + comma	1999	Monetary amount
Number + period	3.441	Percentage or monetary amount
Number + colon	1:00	Time
Number +α + slash	1/10th	Fraction or score
All Capital word	BLP	Organization name
Capital + one period	F.	Name abbreviation
Capital + periods	M.S.	Organization abbreviation
α + money	NT$	Money
α + periods	Dr.	Titles
Capital word	Taipei	Proper noun
Number + α	F-91	Some code
Initial capitalization	Jhonli	Person/Location/ Organization names
Inner capitalization	WordNet	Proper noun
All lower case	be	General words
Others	5\/4	Special symbols

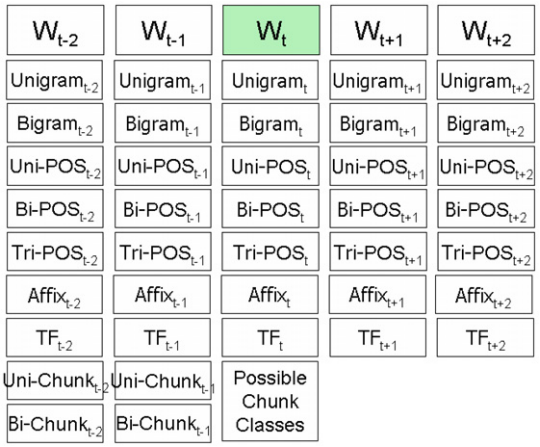

Fig. 2. The employed contextual feature representation style in this paper.

600,000, as for the large-scale training set WSJ sections 00–19, the number of dimensions is 2,300,000. The number of non-zero features per datum is 51.19 and 54.05 for the two datasets, respectively.

3.2. SVM-learning algorithms

We use the SVM as the classification algorithm, which has been shown to perform well on text classification problems [14,15]. All of the training and testing data are converted into vectors as described above. In this paper, we use SVM^{light} [14] as the learning algorithm, which is quite effective and efficient in dealing with high-dimensional data. We do not tune the parameter settings to that of SVM, i.e., we simply use the default parameters of SVM^{light}.

Since the SVM algorithm is a binary classifier, we have to convert it into several binary problems. Here, we use the "One-Against-All" type to solve the problem. Thus, for 23 chunk types, 23 SVM models are trained. The default kernel type of SVM^{light} is a linear kernel-based model which is the simplest and the most efficient model among other kernel types, such as "polynomial", "radial basis function (RBF)", and "sigmoid". When using these advanced kernel types in SVM, it will increase both training and testing time since these kernels types map original data items into higher dimensional vectors. For example, the training time of Kudoh's SVM chunker (polynomial kernel with degree two) is 24 h and the tagging speed is about 20 tokens per second. On the contrary, our SVM chunking system spent 37 min on training but required about 3 h when training with unknown word model. The complete tagging speed of our chunking model is about 810–988 tokens per second (including the full turn-around time). Both the training and testing time are largely improved.

4. The masked method

In real world, training data is usually not sufficient where the out-of-vocabulary problem often occurs. During testing, if a term is an unknown word (or one of its context words is unknown), then its unigram, bigram, and possible chunk class features will be set to zero because we do not have any information of the term from the training data. In this case, the chunk class of this word is mainly determined by non-lexical features. Thus, there is no lexical information when the term is unknown.

The most common way for solving unknown word problem is to use different feature sets for unknown words and divide the training data into several parts to collect unknown word examples [2,7,21,11]. However, the selection of these feature sets for known word and unknown word were often arranged heuristically and it is difficult to select when the feature sets are different. Moreover, they just extracted the unknown word examples and miss the instances that contain unknown contextual words.

To solve this, we change the view of unknown word chunking problem. We present the masked method which aims to derive more training examples from the original training set. Fig. 3 lists the proposed masked algorithm. Suppose we have derived lexicon-related features from the training dataset, including unigram dictionary (*UD*), bigram dictionary (*BD*), and possible-chunk-class dictionary (*PD*) from all training parts except for part *i*. We then generate new training examples by mapping the new dictionary set Σ_i (Σ_i is created from *F* by replacing *UD*, *BD*, and *PD* with UD_i, BD_i, and PD_i, respectively). Technically, we derive a new feature set Σ' of length $|\Sigma|$ where a bit is set for a lexicon in Σ_i and clear if the lexicon is not in Σ_i. We then generate new vectors for all examples by representing the original examples with the new derived feature set Σ'. Thus, items which appear only in part *i* are regarded as unknown words. The process is repeated for *k* times and a total of $(k+1) \times n$ example vectors are generated (*n* is the original number of training examples).

Let us starts with a simple example, suppose we have the feature set Σ that includes eight dimensions (A–H). By making one part, the derived feature set Σ_i includes features: B, C, D. The masked feature set Σ' should be produced by "AND" the Σ and Σ_i. If an example that contains features A, B, H, it will be re-represented B since only B occurs in Σ'. In other words, both A and H were masked as regarded as unknown words.

The masked method can transform the known lexical features into unknown lexical features and add *k* times training materials from the original training set. Thus, it is not necessary to prepare additional training materials to derive unknown word examples and the original data can be reused effectively. As outlined in Fig. 3, new examples are produced through mapping into the derived new feature set Σ_i. This is quite different from previous approaches, which employed variant feature set for unknown words. The proposed method aims to emulate examples that do not contain lexical information, since errors often occur due to the lack of lexical features in training phase. Traditionally, the learners are given sufficient and complete lexical information; therefore, the

Algorithm: Masked_Method(Σ, TS)

Input: the set of labeled training example set TS
Output: masked labeled training example set MTS.
Algorithm:
Step 1. Let Σ be the feature dictionary constructed in Section 3.
Let lexical-related dictionary be:
UD: the unigram feature dictionary
BD: the bigram feature dictionary
PD: the possible-chunk-class dictionary
($UD \subset \Sigma$, $BD \subset \Sigma$, and $PD \subset \Sigma$)
$MTS = \phi$
Step 2. Divide the original training data TS into k parts.

$$TS = \bigcup_{i=1}^{k} TS_i$$

Step 3. For each part i, mask TS_i by operating $TS' = TS - TS_i$
31 Generate lexical-related dictionaries from TS'
UD_i: the unigram dictionary of TS'
BD_i: the bigram dictionary of TS'
PD_i: the possible-chunk-class dictionary of TS'
Σ_i is created from Σ by replacing $UD/BD/PD$ with $UD_i/BD_i/PD_i$
32 Create a new feature set Σ' and
$\Sigma' = \Sigma_i \cap \Sigma$
33 For each training example v_j represented by feature Σ',
3.4 $MTS = MTS \cup v_j$
Step 4. Go to Step 3 until all parts have been processed

Fig. 3. The masked method algorithm.

trained models cannot generalize examples that contain incomplete lexical information. By including incomplete examples, the learners can re-adjust the feature weights ti improve testing performance for unknown words.

5. Evaluations and experimental results

In this section, we report the experimental results on the performance of the proposed chunker, the effect of the masked method, and the scalability when porting to large datasets. We use two standard chunking datasets: one from the CoNLL-2000 shared task and one from WSJ sections 00–24 for the large-scale experiment.

For the CoNLL-2000 chunking task, the training data are derived from the WSJ sections 15–18 while the testing data are taken from section 20. Table 6 lists the detailed records of the two datasets. For the training part, there are 220,663 tokens (terms and punctuations) in the training set and 49,389 tokens in the testing part. The ratio of the training and testing sets is about 4.5:1. For the large-scale chunking task, the training data is derived from the WSJ sections 00–19 and the testing data are taken from sections 20–24. Both the size of training and testing parts are four times larger than those of CoNLL-2000. The distribution of the chunk types in the training data of the large-scale WSJ was shown in Table 7. The largest chunk type is NP (Noun Phrase) and the three large chunk types (NP, VP, and PP) cover more than 90% of the whole dataset.

There are three parameters in our chunking system: the context window size, the frequent threshold for each feature dictionary, and the number of division parts for the unknown word examples (k). We set the first two parameters to 2 as previous chunking systems [16,17]. Since the training time taken by SVMs scales exponentially with the number of input examples [14,15], we set k as two for all of the following experiments.

Table 6
Description of the CoNLL-2000 and large-scale datasets

	Tokens	Phrases	Sentences
CoNLL-2000			
Training	220663	106978	8936
Testing	49389	23852	2012
Large-scale dataset			
Training (00–19)	999277	481529	42581
Testing (20–24)	226832	108750	9762

Table 7
Percentage of each chunk type in the training set of the large-scale task

Chunk type	Tokens	Phrases	Percentage (%)
ADJP	12338	9240	1.92
ADVP	19742	17846	3.71
CONJP	716	302	0.06
INTJ	162	125	0.03
LST	47	45	0.001
NP	536884	249196	51.75
PP	96754	95427	19.82
PRT	2594	2590	0.54
SBAR	10636	10325	2.14
UCP	33	9	0.00
VP	150881	96424	20.02

The performance of the chunking task is usually measured with three rates, namely recall, precision, and $F_{(\beta=1)}$ [29]. First, the recall rate is to estimate the ratio of chunks found by the system. Second, the precision rate measures the percentage of the predicted chunks that are correct. Finally, the $F_{(\beta=1)}$ rate combines both recall and precision rates into one single measure by the following,

$$F_{(\beta=1)} = \frac{2 \times \text{recall} \times \text{precision}}{\text{precision} + \text{recall}}$$

We use the perl-script evaluator (see http://lcg-www.uia.ac.be/conll2000/chunking/conlleval.txt) released by CoNLL to evaluate the three measures for the following experimental results.

5.1. Standard test vs non-standard test

The first test (standard) is performed under a strict constraint, i.e., all of the settings should coincide with those of the CoNLL-2000 shared task. Thus, the use of external resources or other components is not required. We report two results, one is trained with complete lexical information and the other is trained with both complete and incomplete examples (i.e., the masked method). The performance comparison is shown in Table 8, where the latter model achieves better system performance. The training time is less than 30 min and 2.8 h, respectively. The total tagging time for the testing data is about 50 s; in other words, the tagging speed is nearly 1000 tokens per second, respectively. Compared with the voted-SVMs and voted-perceptrons, our system is better in both accuracy and efficiency. Although the Winnow algorithm is more efficient in training (12 min), our chunker has better performance (94.12 vs 93.57 in $F_{(\beta=1)}$ rate). The overall chunking results are listed in Table 9 (left part) where the masked method is used.

In the second test (non-standard), chunking systems could make use of external resources to enhance system performances. Therefore, we employ an advanced POS tagger proposed by Giménez and Márquez [11] instead of the original Brill-tagger. In our chunking system, rich POS *N*-gram features are used to extract more syntactic information. Thus, the more accurate the POS tagger is, the better the chunker performs. In our experiments, the SVM POS tagger achieves 97.20 token accuracy on the CoNLL-2000 test data compared with 97.00 token accuracy achieved by the original Brill-tagger. It is worth to note that the SVM POS tagger was trained with WSJ sections 00–18 which did not overlap with testing part of chunking tasks. But the testing data is a part of validation data for tuning the parameter settings of the SVM POS tagger.

Table 10 lists the chunking performances of related researches that adopt the same constraint. Again, our chunker achieves the best performance ($F_{(\beta=1)}$ rate = 94.20 obtained by employing the masked method). The second and third best systems (Generalized/Regularized Winnows) were mainly contributed by the combinations of external parsers (ESG grammar). In practice, the Treebank data are not always available in many languages and domains. Thus, the use of parsers is not an economical choice of external resources when porting to other languages and

Table 8
Comparison of chunking performance for standard test

Chunking system	Recall	Precision	$F_{(\beta)}$
This paper (with masked method)	94.11	94.13	94.12
Voted-SVMs [17]	93.89	93.92	93.91
Voted-perceptrons [3]	94.20	93.38	93.79
Generalized Winnow [33]	93.60	93.54	93.57
This paper (without masked method)	93.60	93.53	93.56
Regularized Winnow [32]	93.49	93.53	93.51
SVMs [16]	93.51	93.45	93.48

Table 10
Chunking performance of some related (non-standard) studies

Chunking system	Recall	Precision	$F_{(\beta)}$
SVM+ incomplete examples (This paper)	94.32	94.07	94.20
Generalized Winnow [33]	94.07	94.28	94.17
Regularized Winnow [32]	94.01	94.24	94.13
Voted-perceptrons [5]	93.65	94.28	93.96
Voted-SVMs [17]	93.89	93.92	93.91
SVM (This paper)	93.88	93.62	93.75

Table 9
Chunking performance for each chunk type for standard (left) and non-standard (right) tests

Chunk type	Recall	Precision	$F_{(\beta)}$	Recall	Precision	$F_{(\beta)}$
ADJP	71.46	81.09	75.97	73.52	81.73	77.40
ADVP	81.41	83.63	82.50	82.43	82.33	82.38
CONJP	55.56	41.67	47.62	45.45	55.56	50.00
INTJP	50.00	100.00	66.67	50.00	100.00	66.67
LST	0.00	0.00	0.00	0.00	0.00	0.00
NP	94.49	94.62	94.55	94.57	94.54	94.55
PP	98.30	96.91	97.60	98.44	96.97	97.70
PRT	79.25	76.36	77.78	83.02	75.86	79.28
SBAR	86.92	87.74	87.32	88.60	87.29	87.94
VP	94.61	94.19	94.40	94.72	94.33	94.53
All	94.11	94.13	94.12	94.32	94.07	94.20

domains. On the contrary, to develop POS-tagging and chunking corpus is much easier. Note that the same POS tagger is used by the fourth best chunking system, voted-perceptrons [5]. The improvement of voted-perceptrons by SVM POS tagger is from 93.79 to 93.96 while our chunking system enhances $F_{(\beta)}$ rates from 94.12 to 94.20. It is clear that when employing more advanced POS taggers, the chunking performance can be improved.

The learning curve of our chunker is shown in Fig. 4. When using 0.1 million examples, we achieve 91.32 in $F_{(\beta=1)}$ rate. On the contrary, the HMM-based chunker [19] made use of one million training examples to achieve 93.25 in $F_{(\beta=1)}$ rate. Although HMM is much efficient than our SVM chunker, our chunker outperforms the HMM-based systems and the need of training data is almost 1/10 times. In addition, when training with 0.14 and 0.16 million examples, the performance is equivalent (93.63 and 93.68).

We also show the masked method combined with the polynomial kernel SVM. Table 11 lists the results on the 11 chunk types for the CoNLL-2000 shared task. In this experiment, the total training time of the polynomial kernel is about 4 days with masked method (in the right hand side of Table 11), and 14 h without masked method (in the left hand side of Table 11). For the tagging speed, the turn-around times of the two chunkers are about 105 min. The improvement of the polynomial kernel is marginally (from 94.12 to 94.21), but the time cost is not tolerable.

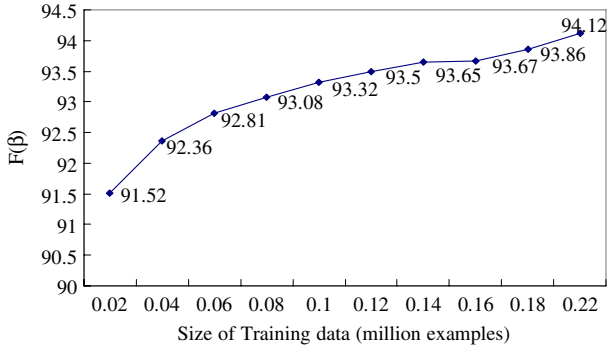

Fig. 4. System performance using CoNLL-2000 training set of different sizes.

Table 11
Chunking performance for each chunk type for polynomial kernel without (left) and with (right) the masked method

Chunk type	Recall	Precision	$F_{(\beta)}$	Recall	Precision	$F_{(\beta)}$
ADJP	72.60	81.69	77.00	71.92	78.75	75.18
ADVP	81.99	83.33	82.65	81.29	82.15	81.72
CONJP	55.56	55.56	55.56	55.56	45.45	50.00
INTJP	0.00	0.00	0.00	100.00	100.00	100.00
LST	0.00	0.00	0.00	0.00	0.00	0.00
NP	94.52	94.26	94.39	94.74	94.67	94.71
PP	98.34	96.53	97.43	98.32	97.01	97.66
PRT	76.42	77.88	77.14	78.30	78.30	78.30
SBAR	85.61	89.80	87.66	87.66	89.33	88.49
VP	94.48	94.22	94.35	94.59	94.39	94.49
All	94.10	93.95	94.03	94.26	94.16	94.21

5.2. The improvement of masked method

Tables 8 and 10 showed that the inclusion of incomplete examples as training instances has improved the chunking performance from 93.56 to 94.12 (from 93.75 to 94.20) for standard (non-standard) test. As listed in Table 12, the percentage of unknown phrases in the testing set is about 13.53% and the improvement for unknown phrases is from 89.68 to 93.43 (89.31–90.19). The masked method improves not only the unknown phrase chunking, but also known phrase chunking for both standard and non-standard tests.

In order to compare with previous studies, we also implement several unknown word recognition methods similar to [21] and [11]. The first method used a simple feature set, unigram, uni-POS, uni-Chunk tags for representing the known word examples, while the affix features are used to represent unknown word examples. The second method employed more features (see Section 3.1) to represent the known word examples, while the affix features are used for representing the unknown word examples. In this evaluation, we use the same settings to the SVM under the standard test. Table 13 shows the experimental results of these methods.

5.3. Impact on POS taggers

As discussed above, chunking system performance can be improved by employing a better POS tagger. We extend our work to compare the influence on various POS taggers. Three POS taggers:Brill [2], SVM [11] and Tree [24] are compared here. The performances of the three POS taggers on the CoNLL-2000 chunking and large-scale datasets are listed in Table 14. Note that these taggers are trained from various sources. SVM POS tagger achieves the best result on CoNLL-2000 and large-scale dataset.

Table 12
The improvement by the masked method

CoNLL-2000	Percentage (%)	Standard	Non-standard
Unknown	13.53	89.84 → 90.74	89.46 → 90.18
Known	86.46	94.34 → 94.79	94.63 → 95.03
Total	100.00	93.56 → 94.12	93.75 → 94.21

Table 13
Comparison to other unknown example collecting methods

	Recall	Precision	$F_{(\beta)}$
Simple feature set	92.03	91.46	91.74
Complete feature set	93.55	93.50	93.53
Nakagawa's method [21]	92.12	91.53	91.82
Nakagawa's method + our feature set	93.42	93.34	93.38
Masked method	94.11	94.13	94.12

Table 14
Results of using POS taggers on the CoNLL-2000 and large-scale test sets

	CoNLL-2000	Large-scale dataset
Brill-tagger	97.00	96.92
Tree-tagger	96.80	97.03
SVM POS tagger	97.23	97.13

Table 15
Comparison of the chunking performance using various POS taggers

	Recall	Precision	$F_{(\beta)}$
Chunking + Brill-tagger	94.11	94.13	94.12
Chunking + Tree-tagger	94.32	94.40	94.36
Chunking + SVM POS tagger	94.26	94.16	94.21
Chunking + Gold-tagger	94.72	94.58	94.65

Table 15 shows the chunking performances attained by using various POS taggers. In this experiment, an additional POS taggers is used, i.e., Gold (hand-annotated) POS tagger. As seen in the table, the first and last results give the lower and upper bounds of the chunking performances, respectively. Interestingly, when our chunker is combined with the Tree-tagger, it achieves better results than combined with the SVM POS tagger, although Tree-tagger does not perform better than SVM POS tagger in POS tagging. However, since the training data of the Tree-tagger cover various parts in the WSJ and some of the testing data of CoNLL-2000 may be included in the Tree-tagger. The SVM POS tagger made use of WSJ sections 19–21 for validation. We therefore use the Brill-tagger in the large-scale experiments.

5.4. Large-scale experiments

For large-scale experiments, we use the Brill-tagger to generate POS tags. Table 16 gives the chunking results of each phrase type. The total training times of large-scale experiments are 27.8 and 4.8 h without using the masked method. Both training times increase more than nine folds compared with previous results. However, the total tagging time of the large-scale data is only 4 min, meaning that the tagging speed is near 1000 tokens per second.

Table 16
Chunking performance on the large-scale chunking task

Chunk Type	Recall	Precision	$F_{(\beta)}$
ADJP	78.10	84.28	81.07
ADVP	85.18	83.57	84.37
CONJP	70.77	71.88	71.32
INTJP	65.52	73.08	69.09
LST	30.00	66.67	41.38
NP	95.59	95.73	95.66
PP	98.37	97.27	97.82
PRT	79.13	80.51	79.81
SBAR	89.55	90.74	90.14
UCP	0.00	0.00	0.00
VP	95.00	95.13	95.06
All	95.01	95.01	95.01

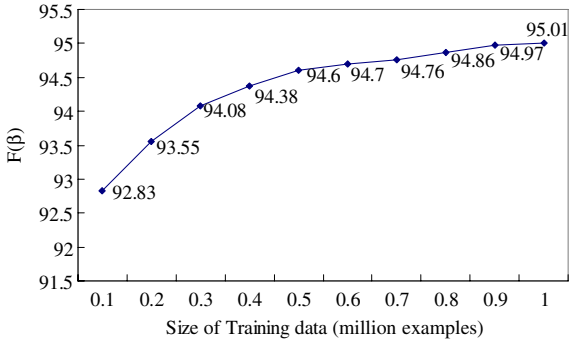

Fig. 5. System performance using large-scale training set of different sizes.

Fig. 5 shows the system performances for different training sizes. The lower bound of our chunker achieves 92.83 $F_{(\beta)}$ rate using 0.07 million training examples, while the upper bound 95.01 is reached using all (~1 million) training examples.

6. Conclusion and future remark

Efficient and high-performance text chunker is important for many real world NLP applications. Unknown words in testing examples are one of the reasons accounting for chunking errors. This paper proposes a "masked-based" method to derive incomplete training examples and therefore improve chunking performance. Compared with pervious unknown word example collecting techniques, the masked method is better. The proposed masked method improves the performance not only of known words but also unknown words. From the experiments, our chunking system performs better than other systems in both standard and non-standard tests. The best system performances of our chunker are 94.21 (with polynomial kernel) and 94.12 (with linear kernel) in $F_{(\beta)}$ rates, which outperforms the other chunking systems that employed external resources or combined multiple learners.

Moreover, the masked method can be combined with other learners, like Winnow and voted-perceptrons. In terms of efficiency, by using a linear SVM kernel with richer feature set and masked method, the learning time is less than 3 h for CoNLL-2000 training data and the turn-around time of the testing data is 50 sec. In the large-scale test, our system also performs better than HMM-based chunking systems. The online demonstration of our chunker can be found at the following web site: http://dblab87.csie.ncu.edu.tw/bcbb/chunking.htm.

Acknowledgements

We would like to thank Prof. Jie-Chi Yang, and Yue-Shi Lee for valuable comments and revisions. This work is sponsored by National Science Council, Taiwan under

Grant NSC94-2524-S-008-002 and NSC94-2622-E-130-001-CC3.

References

[1] S. Abney, Parsing by chunks, in: Principle-Based Parsing: Computation and Psycholinguistics, 1991, pp. 257–278.
[2] E. Brill, Transformation-based error-driven learning and natural language processing: a case study in part of speech tagging, Comput. Linguist. 21 (4) (1995) 543–565.
[3] X. Carreras, L. Marquez, Phrase recognition by filtering and ranking with perceptrons, in: Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP), 2003, pp. 205–216.
[4] X. Carreras, L. Marquez, J. Castro, Filtering-ranking perceptron learning for partial parsing, Mach. Learn. J. 60 (2005) 41–71.
[5] X. Carreras, L. Marquez, Introduction to the CoNLL-2004 shared task: semantic role labeling, in: Proceedings of Conference on Natural Language Learning (CoNLL), 2004, pp. 89–97.
[6] N. Cristianini, J. Shawe-Taylor, An introduction to support vector machines and other kernel-based learning methods, Cambridge University Press, Cambridge, 2000.
[7] W. Daelemans, Memory-based lexical acquisition and processing, in: Proceedings of the 3rd International EAMT Workshop on Machine Translation and the Lexicon, 1995, pp. 85–98.
[8] W. Daelemans, J. Zavrel, K. van der Sloot, A. van den Bosch, TiMBL Tilburg memory based learner, version 3.0, Reference Guide.ILK Technical Report 00-01, 2000.
[9] W. Daelemans, J. Zavrel, P. Berck, S. Gillis, TiMBLL MBT: a memory-mased part of speech tagger-generator, in: Proceedings of Workshop on Very Large Corpora (WVLC), 1996, pp. 14–27.
[10] R. Florian, A. Ittycheriah, H. Jing, T. Zhang, Named Entity Recognition through Classifier Combination, in: Proceedings of Conference on Natural Language Learning (CoNLL), 2003, pp. 168–171.
[11] J. Giménez, L. Márquez, Fast and accurate Part-of-Speech tagging: the SVM approach revisited, in: Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP), 2003, pp. 158–165.
[12] H. van Halteren, Chunking with WPDV Models, in: Proceedings of Conference on Natural Language Learning (CoNLL), 2000, pp. 154–156.
[13] H.v. Halteren, J. Zavrel, W. Daelemans, Improving accuracy in word class tagging through the combination of machine learning systems, Comput. Linguist. 27 (2) (2001) 199–229.
[14] T. Joachims, Text categorization with support vector machines: learning with many relevant features, in: Proceedings of the European Conference on Machine Learning, 1998, pp. 137–142.
[15] T. Joachims, A statistical learning model of text classification with support vector machines, in: Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2001, pp. 128–136.
[16] T. Kudoh, Y. Matsumoto, Use of support vector learning for chunk identification, in: Proceedings of Conference on Natural Language Learning (CoNLL), 2000, pp. 142–144.
[17] T. Kudoh, Y. Matsumoto, Chunking with support vector machines, in: Proceedings of the 2nd Meeting of North American Chapter of Association for Computational Linguistics (NAACL), 2001, pp. 192–199.
[18] B. Megesi, Shallow parsing with PoS taggers and linguistic knowledge, J. Mach. Learn. Res. (2002) 639–668.
[19] A. Molina, F. Pla, Shallow parsing using Specialized HMMs, J. Mach. Learn. Res. (2002) 595–613.
[20] M. Muñoz, V. Punyakanok, D. Roth, D. Zimak, A learning approach to shallow parsing, in: Proceedings of 1999 Joint SIGDAT Conference on Empirical methods in NLP and Very Large Corpora, 1999, pp. 168–178.
[21] T. Nakagawa, T. Kudoh, Y. Matsumoto, Unknown word guessing and Part-of-Speech tagging using support vector machines, in: Proceedings of the 6th Natural Language Processing Pacific Rim Symposium, 2001, pp. 325–331.
[22] S. Bae Park, B. Tak Zhang, Co-trained support vector machines for large scale unstructured document classification using unlabeled data and syntactic information, J. Inf. Proc. Manage 40 (2004) 421–439.
[23] Lance A. Ramshaw, Mitchell P. Marcus, Text chunking using transformation-based learning, in: Proceedings of the ACL 3rd Workshop on Very Large Corpora, 1995, pp. 82–94.
[24] H. Schmid, Probabilistic Part-of-Speech tagging using decision trees, in: International Conference on New Methods in Language Processing, 1994, pp. 44–49.
[25] F. Sebastiani, Machine learning in automated text categorization, J. ACM Comput. Surv. 34 (1) (2002) 1–47.
[26] Eric F. Tjong Kim Sang, Memory-based shallow parsing, J. Mach. Learn. Res. (2002) 559–594.
[27] Eric F. Tjong Kim Sang, Transforming a chunker to a parser, in: Computational Linguistics in the Netherlands, 2000a, pp. 177–188.
[28] Eric F. Tjong Kim Sang, Text chunking by system combination, in: Proceedings of Conference on Natural Language Learning (CoNLL), 2000b, pp. 151–153.
[29] Eric F. Tjong Kim Sang, Sabine Buchholz, Introduction to the CoNLL-2000 shared task: chunking, in: Proceedings of Conference on Natural Language Learning (CoNLL), 2000, pp. 127–132.
[30] Erik F. Tjong Kim Sang, Jorn Veenstra, Representing Text Chunks, in: Proceedings of 9th Conference of the European Chapter of the Association for Computational Linguistics, 1999, pp. 173–179.
[31] Yang, Hui, Tat-Seng Chua, Shuguang Wang, Chun-Keat Koh, Structural use of external knowledge for event-based open domain question answering, in: Proceedings of the 26th ACM SIGIR conference on research and development in information retrieval, 2003, pp. 33–40.
[32] T. Zhang, F. Damerau, D. Johnson, Text chunking using regularized Winnow, in: Proceedings of 39th Annual Meetings of the Association for Computational Linguistics, 2001, pp. 539–546.
[33] T. Zhang, F. Damerau, D. Johnson, Text Chunking based on a Generalization Winnow, J. Mach. Learn. Res. 2 (2002) 615–637.
[34] G. Dong Zhou, J. Su, Error-driven HMM-based chunk tagger with context-dependent Lexicon, in: Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, 2000, pp. 71–79.