

# Efficient Support Vector Classifiers for Named Entity Recognition

Hideki Isozaki and Hideto Kazawa

NTT Communication Science Laboratories

Nippon Telegraph and Telephone Corporation

2-4 Hikari-dai, Seika-cho, Soraku-gun, Kyoto, 619-0237, Japan

{isozaki,kazawa}@cslab.kecl.ntt.co.jp

## Abstract

Named Entity (NE) recognition is a task in which proper nouns and numerical information are extracted from documents and are classified into categories such as person, organization, and date. It is a key technology of Information Extraction and Open-Domain Question Answering. First, we show that an NE recognizer based on Support Vector Machines (SVMs) gives better scores than conventional systems. However, off-the-shelf SVM classifiers are too inefficient for this task. Therefore, we present a method that makes the system substantially faster. This approach can also be applied to other similar tasks such as chunking and part-of-speech tagging. We also present an SVM-based feature selection method and an efficient training method.

## 1 Introduction

Named Entity (NE) recognition is a task in which proper nouns and numerical information in a document are detected and classified into categories such as person, organization, and date. It is a key technology of Information Extraction and Open-Domain Question Answering (Voorhees and Harman, 2000). We are building a trainable Open-Domain Question Answering System called SAIQA-II. In this paper, we show that an NE recognizer based on Support Vector Machines (SVMs) gives better scores than conventional systems. SVMs have given high performance in various classification tasks (Joachims, 1998; Kudo and Matsumoto, 2001).

However, it turned out that off-the-shelf SVM classifiers are too inefficient for NE recognition. The recognizer runs at a rate of only 85 bytes/sec on an Athlon 1.3 GHz Linux PC, while rule-based systems (e.g., Isozaki, (2001)) can process several kilobytes in a second. The major reason is the inefficiency of SVM classifiers. There are other reports on the slowness of SVM classifiers. Another SVM-based NE recognizer (Yamada and Mat-

sumoto, 2001) is 0.8 sentences/sec on a Pentium III 933 MHz PC. An SVM-based part-of-speech (POS) tagger (Nakagawa et al., 2001) is 20 tokens/sec on an Alpha 21164A 500 MHz processor. It is difficult to use such slow systems in practical applications.

In this paper, we present a method that makes the NE system substantially faster. This method can also be applied to other tasks in natural language processing such as chunking and POS tagging. Another problem with SVMs is its incomprehensibility. It is not clear which features are important or how they work. The above method is also useful for finding useless features. We also mention a method to reduce training time.

### 1.1 Support Vector Machines

Suppose we have a set of training data for a two-class problem:  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$ , where  $\mathbf{x}_i (\in \mathbf{R}^D)$  is a feature vector of the  $i$ -th sample in the training data and  $y_i \in \{+1, -1\}$  is the label for the sample. The goal is to find a decision function that accurately predicts  $y$  for unseen  $\mathbf{x}$ . A non-linear SVM classifier gives a decision function  $f(\mathbf{x}) = \text{sign}(g(\mathbf{x}))$  for an input vector  $\mathbf{x}$  where

$$g(\mathbf{x}) = \sum_{i=1}^{\ell} w_i K(\mathbf{x}, \mathbf{z}_i) + b.$$

Here,  $f(\mathbf{x}) = +1$  means  $\mathbf{x}$  is a member of a certain class and  $f(\mathbf{x}) = -1$  means  $\mathbf{x}$  is not a member.  $\mathbf{z}_i$ s are called *support vectors* and are representatives of training examples.  $\ell$  is the number of support vectors. Therefore, computational complexity of  $g(\mathbf{x})$  is proportional to  $\ell$ . Support vectors and other constants are determined by solving a certain quadratic programming problem.  $K(\mathbf{x}, \mathbf{z})$  is a *kernel* that implicitly maps vectors into a higher dimensional space. Typical kernels use dot products:  $K(\mathbf{x}, \mathbf{z}) = k(\mathbf{x} \cdot \mathbf{z})$ . A polynomial kernel of degree  $d$  is given by  $k(x) = (1 + x)^d$ . We can use vari-

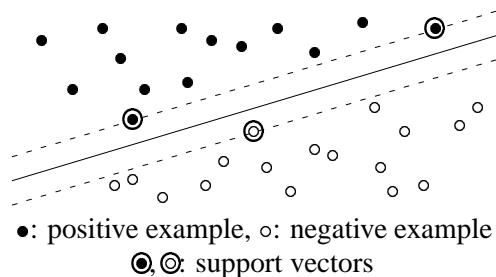


Figure 1: Support Vector Machine

ous kernels, and the design of an appropriate kernel for a particular application is an important research issue.

Figure 1 shows a linearly separable case. The decision hyperplane defined by  $g(\mathbf{x}) = 0$  separates positive and negative examples by the largest margin. The solid line indicates the decision hyperplane and two parallel dotted lines indicate the margin between positive and negative examples. Since such a separating hyperplane may not exist, a positive parameter  $C$  is introduced to allow misclassifications. See Vapnik (1995).

## 1.2 SVM-based NE recognition

As far as we know, the first SVM-based NE system was proposed by Yamada et al. (2001) for Japanese. His system is an extension of Kudo’s chunking system (Kudo and Matsumoto, 2001) that gave the best performance at CoNLL-2000 shared tasks. In their system, every word in a sentence is classified sequentially from the beginning or the end of a sentence. However, since Yamada has not compared it with other methods under the same conditions, it is not clear whether his NE system is better or not. Here, we show that our SVM-based NE system is more accurate than conventional systems. Our system uses the Viterbi search (Allen, 1995) instead of sequential determination.

For training, we use ‘CRL data’, which was prepared for IREX (Information Retrieval and Extraction Exercise<sup>1</sup>, Sekine and Eriguchi (2000)). It has about 19,000 NEs in 1,174 articles. We also use additional data by Isozaki (2001). Both datasets are based on Mainichi Newspaper’s 1994 and 1995 CD-ROMs. We use IREX’s formal test data called GENERAL that has 1,510 named entities in 71 articles from Mainichi Newspaper of 1999. Systems are compared in terms of GENERAL’s F-measure

which is the harmonic mean of ‘recall’ and ‘precision’ and is defined as follows.

Recall =  $M / (\text{the number of correct NEs})$ ,

Precision =  $M / (\text{the number of NEs extracted by a system})$ ,

where  $M$  is the number of NEs correctly extracted and classified by the system.

We developed an SVM-based NE system by following our NE system based on maximum entropy (ME) modeling (Isozaki, 2001). We simply replaced the ME model with SVM classifiers. The above datasets are processed by a morphological analyzer ChaSen 2.2.1<sup>2</sup>. It tokenizes a sentence into words and adds POS tags. ChaSen uses about 90 POS tags such as `common-noun` and `location-name`. Since most unknown words are proper nouns, ChaSen’s parameters for unknown words are modified for better results. Then, a character type tag is added to each word. It uses 17 character types such as `all-kanji` and `small-integer`. See Isozaki (2001) for details.

Now, Japanese NE recognition is solved by the classification of words (Sekine et al., 1998; Borthwick, 1999; Uchimoto et al., 2000). For instance, the words in “President George Herbert Bush said Clinton is . . .” are classified as follows: “President” = OTHER, “George” = PERSON-BEGIN, “Herbert” = PERSON-MIDDLE, “Bush” = PERSON-END, “said” = OTHER, “Clinton” = PERSON-SINGLE, “is” = OTHER. In this way, the first word of a person’s name is labeled as PERSON-BEGIN. The last word is labeled as PERSON-END. Other words in the name are PERSON-MIDDLE. If a person’s name is expressed by a single word, it is labeled as PERSON-SINGLE. If a word does not belong to any named entities, it is labeled as OTHER. Since IREX defines eight NE classes, words are classified into 33 ( $= 8 \times 4 + 1$ ) categories.

Each sample is represented by 15 features because each word has three features (part-of-speech tag, character type, and the word itself), and two preceding words and two succeeding words are also used for context dependence. Although infrequent features are usually removed to prevent overfitting, we use all features because SVMs are robust. Each sample is represented by a long *binary* vector, i.e., a sequence of 0 (false) and 1 (true). For instance, “Bush” in the above example is represented by a

<sup>1</sup><http://cs.nyu.edu/cs/projects/proteus/irex>

<sup>2</sup><http://chasen.aist-nara.ac.jp/>

vector  $\mathbf{x} = (x[1], \dots, x[D])$  described below. Only 15 elements are 1.

```

x[1] = 0 // Current word is not 'Alice'
x[2] = 1 // Current word is 'Bush'
x[3] = 0 // Current word is not 'Charlie'
      :
x[15029] = 1 // Current POS is a proper noun
x[15030] = 0 // Current POS is not a verb
      :
x[39181] = 0 // Previous word is not 'Henry'
x[39182] = 1 // Previous word is 'Herbert'
      :

```

Here, we have to consider the following problems. First, SVMs can solve only a two-class problem. Therefore, we have to reduce the above multi-class problem to a group of two-class problems. Second, we have to consider consistency among word classes in a sentence. For instance, a word classified as PERSON-BEGIN should be followed by PERSON-MIDDLE or PERSON-END. It implies that the system has to determine the best combinations of word classes from numerous possibilities. Here, we solve these problems by combining existing methods.

There are a few approaches to extend SVMs to cover  $n$ -class problems. Here, we employ the “one class versus all others” approach. That is, each classifier  $f_c(\mathbf{x})$  is trained to distinguish members of a class  $c$  from non-members. In this method, two or more classifiers may give +1 to an unseen vector or no classifier may give +1. One common way to avoid such situations is to compare  $g_c(\mathbf{x})$  values and to choose the class index  $c$  of the largest  $g_c(\mathbf{x})$ .

The consistency problem is solved by the Viterbi search. Since SVMs do not output probabilities, we use the SVM+sigmoid method (Platt, 2000). That is, we use a sigmoid function  $s(x) = 1/(1 + \exp(-\beta x))$  to map  $g_c(\mathbf{x})$  to a probability-like value. The output of the Viterbi search is adjusted by a postprocessor for wrong word boundaries. The adjustment rules are also statistically determined (Isozaki, 2001).

### 1.3 Comparison of NE recognizers

We use a fixed value  $\beta = 100$ . F-measures are not very sensitive to  $\beta$  unless  $\beta$  is too small. When we used 1,038,986 training vectors, GENERAL’s F-measure was 89.64% for  $\beta = 0.1$  and 90.03% for  $\beta = 100$ . We employ the quadratic kernel ( $d = 2$ ) because it gives the best results. Polynomial kernels of degree 1, 2, and 3 resulted in 83.03%, 88.31%,

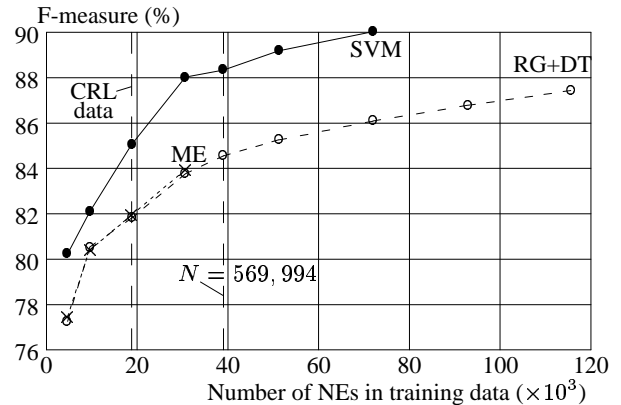


Figure 2: F-measures of NE systems

and 87.04% respectively when we used 569,994 training vectors.

Figure 2 compares NE recognizers in terms of GENERAL’s F-measures. ‘SVM’ in the figure indicates F-measures of our system trained by Kudo’s TinySVM-0.07<sup>3</sup> with  $C = 0.1$ . It attained 85.04% when we used only CRL data. ‘ME’ indicates our ME system and ‘RG+DT’ indicates a rule-based machine learning system (Isozaki, 2001). According to this graph, ‘SVM’ is better than the other systems.

However, SVM classifiers are too slow. Famous SVM-Light 3.50 (Joachims, 1999) took 1.2 days to classify 569,994 vectors derived from 2 MB documents. That is, it runs at only 19 bytes/sec. TinySVM’s classifier seems best optimized among publicly available SVM toolkits, but it still works at only 92 bytes/sec.

## 2 Efficient Classifiers

In this section, we investigate the cause of this inefficiency and propose a solution. All experiments are conducted for training data of 569,994 vectors. The total size of the original news articles was 2 MB and the number of NEs was 39,022. According to the definition of  $g(\mathbf{x})$ , a classifier has to process  $\ell$  support vectors for each  $\mathbf{x}$ . Table 1 shows  $\ell$ s for different word classes. According to this table, classification of one word requires  $\mathbf{x}$ ’s dot products with 228,306 support vectors in 33 classifiers. Therefore, the classifiers are very slow. We have never seen such large  $\ell$ s in SVM literature on pattern recognition. The reason for the large  $\ell$ s is word features. In other domains such as character recognition, dimen-

<sup>3</sup><http://cl.aist-nara.ac.jp/~taku-ku/software/TinySVM>

sion  $D$  is usually fixed. However, in the NE task,  $D$  increases monotonically with respect to the size of the training data. Since SVMs learn combinations of features,  $\ell$  tends to be very large. This tendency will hold for other tasks of natural language processing, too.

Here, we focus on the *quadratic kernel*  $k(x) = (1 + x)^2$  that yielded the best score in the above experiments. Suppose  $\mathbf{x} = (x[1], \dots, x[D])$  has only  $m$  ( $=15$ ) non-zero elements. The dot product of  $\mathbf{x}$  and  $\mathbf{z}_i = (z_i[1], \dots, z_i[D])$  is given by  $\sum_{h=1}^D x[h]z_i[h]$ . Hence,

$$(1 + \mathbf{x} \cdot \mathbf{z}_i)^2 = 1 + 2 \sum_{h=1}^D x[h]z_i[h] + \left( \sum_{h=1}^D x[h]z_i[h] \right)^2.$$

We can rewrite  $g(\mathbf{x})$  as follows.

$$\begin{aligned} g(\mathbf{x}) &= W_0 + \sum_{h=1}^D (W_1[h]x[h] + W_2[h]x[h]^2) \\ &\quad + \sum_{h=1}^{D-1} \sum_{k=h+1}^D W_3[h, k]x[h]x[k], \end{aligned}$$

where

$$\begin{aligned} W_0 &= b + \sum_{i=1}^{\ell} w_i, \\ W_1[h] &= 2 \sum_{i=1}^{\ell} w_i z_i[h], \\ W_2[h] &= \sum_{i=1}^{\ell} w_i z_i[h]^2, \\ W_3[h, k] &= 2 \sum_{i=1}^{\ell} w_i z_i[h]z_i[k]. \end{aligned}$$

For binary vectors, it can be simplified as

$$g(\mathbf{x}) = W_0 + \sum_{h: x[h]=1} (W'_1[h] + \sum_{k: k>h, x[k]=1} W_3[h, k]),$$

where

$$\begin{aligned} W'_1[h] &= W_1[h] + W_2[h] = 3 \sum_{i: z_i[h]=1} w_i, \\ W_3[h, k] &= 2 \sum_{i: z_i[h]=z_i[k]=1} w_i. \end{aligned}$$

Now,  $g(\mathbf{x})$  can be given by summing up  $W'_1[h]$  for every non-zero element  $x[h]$  and  $W_3[h, k]$  for every non-zero pair  $x[h]x[k]$ . Accordingly, we only need to add  $1 + m + m(m-1)/2$  ( $=121$ ) constants to get  $g(\mathbf{x})$ . Therefore, we can expect this method to be much faster than a naïve implementation that computes tens of thousands of dot products at run time. We call this method ‘XQK’ (eXpand the Quadratic Kernel).

Table 1 compares TinySVM and XQK in terms of CPU time taken to apply 33 classifiers to process the training data. Classes are sorted by  $\ell$ . Small numbers in parentheses indicate the initialization time for reading support vectors  $\{\mathbf{z}_i\}$  and allocating memory. XQK requires a longer initialization time in order to prepare  $W'_1$  and  $W_3$ . For instance, TinySVM took 11,490.26 seconds (3.2 hours) in total for applying OTHER’s classifier to all vectors in the training data. Its initialization phase took 2.13 seconds and all vectors in the training data were classified in 11,488.13 ( $= 11,490.26 - 2.13$ ) seconds. On the other hand, XQK took 225.28 seconds in total and its initialization phase took 174.17 seconds. Therefore, 569,994 vectors were classified in 51.11 seconds. The initialization time can be disregarded because we can reuse the above coefficients. Consequently, XQK is 224.8 ( $=11,488.13/51.11$ ) times faster than TinySVM for OTHER. TinySVM took 6 hours to process all the word classes, whereas XQK took only 17 minutes. XQK is 102 times faster than SVM-Light 3.50 which took 1.2 days.

### 3 Removal of useless features

XQK makes the classifiers faster, but memory requirement increases from  $O(\sum_{i=1}^{\ell} m_i)$  to  $O(\sum_{i=1}^{\ell} m_i(m_i+1)/2)$  where  $m_i$  ( $=15$ ) is the number of non-zero elements in  $\mathbf{z}_i$ . Therefore, removal of useless features would be beneficial. Conventional SVMs do not tell us how an individual feature works because weights are given not to features but to  $K(\mathbf{x}, \mathbf{z}_i)$ . However, the above weights ( $W'_1$  and  $W_3$ ) clarify how a feature or a feature pair works. We can use this fact for feature selection *after* the training.

We simplify  $f(\mathbf{x})$  by removing all features  $h$  that satisfy  $\max(|W'_1[h]|, \max_k |W_3[h, k]|, \max_k |W_3[k, h]|) < \theta$ . The largest  $\theta$  that does not change the number of misclassifications for the training data is found by using the binary search for each word class. We call this method ‘XQK-FS’ (XQK with Feature Selection). This approximation slightly degraded GENERAL’s F-measure from 88.31% to 88.03%.

Table 2 shows the reduction of features that appear in support vectors. Classes are sorted by the numbers of original features. For instance, OTHER has 56,220 features in its support vectors. According to the binary search, its performance did not change even when the number of features was reduced to 21,852 at  $\theta = 0.02676$ .

Table 1: Reduction of CPU time (in seconds) by XQK

word class	$\ell$	TinySVM (init)	XQK (init)	speed up	SVM-Light
OTHER	64,970	11,488.13 (2.13)	51.11 (174.17)	224.8	29,986.52
ARTIFACT-MIDDLE	14,171	1,372.85 (0.51)	41.32 (14.98)	33.2	6,666.26
LOCATION-SINGLE	13,019	1,209.29 (0.47)	38.24 (11.41)	31.6	6,100.54
ORGANIZ...-MIDDLE	12,050	987.39 (0.44)	37.93 (11.70)	26.0	5,570.82
:	:	:	:	:	:
TOTAL	228,306	21,754.23 (9.83)	1,019.20 (281.28)	21.3	104,466.31

Table 2: Reduction of features by XQK-FS

word class	number of features	number of non-zero weights	seconds
OTHER	56,220 $\rightarrow$ 21,852 (38.9%)	1,512,827 $\rightarrow$ 892,228 (59.0%)	42.31
ARTIFACT-MIDDLE	22,090 $\rightarrow$ 4,410 (20.0%)	473,923 $\rightarrow$ 164,632 (34.7%)	30.47
LOCATION-SINGLE	17,169 $\rightarrow$ 3,382 (19.7%)	366,961 $\rightarrow$ 123,808 (33.7%)	27.72
ORGANIZ...-MIDDLE	17,123 $\rightarrow$ 9,959 (58.2%)	372,784 $\rightarrow$ 263,695 (70.7%)	31.02
ORGANIZ...-END	15,214 $\rightarrow$ 3,073 (20.2%)	324,514 $\rightarrow$ 112,307 (34.6%)	26.87
:	:	:	:
TOTAL	307,721 $\rightarrow$ 75,455 (24.5%)	6,669,664 $\rightarrow$ 2,650,681 (39.7%)	763.10

The total number of features was reduced by 75% and that of weights was reduced by 60%. The table also shows CPU time for classification by the selected features. XQK-FS is 28.5 ( $=21754.23/763.10$ ) times faster than TinySVM. Although the reduction of features is significant, the reduction of CPU time is moderate, because most of the reduced features are infrequent ones. However, simple reduction of infrequent features without considering weights damages the system’s performance. For instance, when we removed 5,066 features that appeared four times or less in the training data, the modified classifier for ORGANIZATION-END misclassified 103 training examples, whereas the original classifier misclassified only 19 examples. On the other hand, XQK-FS removed 12,141 features without an increase in misclassifications for the training data.

XQK can be easily extended to a more general quadratic kernel  $k(x) = (c_0 + c_1x)^2$  and to non-binary sparse vectors. XQK-FS can be used to select useful features *before* training by other kernels. As mentioned above, we conducted an experiment for the cubic kernel ( $d = 3$ ) by using all features. When we trained the cubic kernel classifiers by using only features selected by XQK-FS, TinySVM’s classification time was reduced by 40% because  $\ell$

was reduced by 38%. GENERAL’s F-measure was slightly improved from 87.04% to 87.10%. On the other hand, when we trained the cubic kernel classifiers by using only features that appeared three times or more (without considering weights), TinySVM’s classification time was reduced by only 14% and the F-measure was slightly degraded to 86.85%. Therefore, we expect XQK-FS to be useful as a feature selection method for other kernels when such kernels give much better results than the quadratic kernel.

#### 4 Reduction of training time

Since training of 33 classifiers also takes a long time, it is difficult to try various combinations of parameters and features. Here, we present a solution for this problem. In the training time, calculation of  $k(\mathbf{x}_a \cdot \mathbf{x}_1), k(\mathbf{x}_a \cdot \mathbf{x}_2), \dots, k(\mathbf{x}_a \cdot \mathbf{x}_N)$  for various  $\mathbf{x}_a$ s is dominant. Conventional systems save time by caching the results. By analyzing TinySVM’s classifier, we found that they can be calculated more efficiently.

For sparse vectors, most SVM classifiers (e.g., SVM-Light) use a sparse dot product algorithm (Platt, 1999) that compares non-zero elements of  $\mathbf{x}$  and those of  $\mathbf{z}_i$  to get  $k(\mathbf{x} \cdot \mathbf{z}_i)$  in  $g(\mathbf{x})$ . However,  $\mathbf{x}$  is common to all dot products in  $k(\mathbf{x} \cdot \mathbf{z}_1), \dots, k(\mathbf{x} \cdot$

$\mathbf{z}_\ell$ ). Therefore, we can implement a faster classifier that calculates them concurrently. TinySVM’s classifier prepares a list `fi2si[h]` that contains all  $\mathbf{z}_i$ s whose  $h$ -th coordinates are not zero. In addition, counters for  $\mathbf{x} \cdot \mathbf{z}_1, \dots, \mathbf{x} \cdot \mathbf{z}_\ell$  are prepared because dot products of binary vectors are integers. Then, for each non-zero  $x[h]$ , the counters are incremented for all  $\mathbf{z}_i \in \text{fi2si}[h]$ . By checking only members of `fi2si[h]` for non-zero  $x[h]$ , the classifier is not bothered by fruitless cases:  $x[h] = 0, z_i[h] \neq 0$  or  $x[h] \neq 0, z_j[h] = 0$ . Therefore, TinySVM’s classifier is faster than other classifiers. This method is applicable to any kernels based on dot products.

For the training phase, we can build `fi2si'[h]` that contains all  $\mathbf{x}_i$ s whose  $h$ -th coordinates are not zero. Then,  $k(\mathbf{x}_a \cdot \mathbf{x}_1), \dots, k(\mathbf{x}_a \cdot \mathbf{x}_N)$  can be efficiently calculated because  $\mathbf{x}_a$  is common. This improvement is effective especially when the cache is small and/or the training data is large. When we used a 200 MB cache, the improved system took only 13 hours for training by the CRL data, while TinySVM and SVM-Light took 30 hours and 46 hours respectively for the same cache size. Although we have examined other SVM toolkits, we could not find any system that uses this approach in the training phase.

## 5 Discussion

The above methods can also be applied to other tasks in natural language processing such as chunking and POS tagging because the quadratic kernels give good results.

Utsuro et al. (2001) report that a combination of two NE recognizers attained  $F = 84.07\%$ , but wrong word boundary cases are excluded. Our system attained  $85.04\%$  and word boundaries are automatically adjusted. Yamada (Yamada et al., 2001) also reports that  $d = 2$  is best. Although his system attained  $F = 83.7\%$  for 5-fold cross-validation of the CRL data (Yamada and Matsumoto, 2001), our system attained  $86.8\%$ . Since we followed Isozaki’s implementation (Isozaki, 2001), our system is different from Yamada’s system in the following points: 1) adjustment of word boundaries, 2) ChaSen’s parameters for unknown words, 3) character types, 4) use of the Viterbi search.

For efficient classification, Burges and Schölkopf (1997) propose an approximation method that uses “reduced set vectors” instead of support vectors. Since the size of the reduced set vectors is smaller than  $\ell$ , classifiers become more efficient, but the

computational cost to determine the vectors is very large. Osuna and Girosi (1999) propose two methods. The first method approximates  $g(\mathbf{x})$  by support vector regression, but this method is applicable only when  $C$  is large enough. The second method reformulates the training phase. Our approach is simpler than these methods. Downs et al. (Downs et al., 2001) try to reduce the number of support vectors by using linear dependence.

We can also reduce the run-time complexity of a multi-class problem by cascading SVMs in the form of a binary tree (Schwenker, 2001) or a direct acyclic graph (Platt et al., 2000). Yamada and Matsumoto (2001) applied such a method to their NE system and reduced its CPU time by 39%. This approach can be combined with our SVM classifiers.

NE recognition can be regarded as a variable-length multi-class problem. For this kind of problem, probability-based kernels are studied for more theoretically well-founded methods (Jaakkola and Haussler, 1998; Tsuda et al., 2001; Shimodaira et al., 2001).

## 6 Conclusions

Our SVM-based NE recognizer attained  $F = 90.03\%$ . This is the best score, as far as we know. Since it was too slow, we made SVMs faster. The improved classifier is 21 times faster than TinySVM and 102 times faster than SVM-Light. The improved training program is 2.3 times faster than TinySVM and 3.5 times faster than SVM-Light. We also presented an SVM-based feature selection method that removed 75% of features. These methods can also be applied to other tasks such as chunking and POS tagging.

## Acknowledgment

We would like to thank Yutaka Sasaki for the training data. We thank members of Knowledge Processing Research Group for valuable comments and discussion. We also thank Shigeru Katagiri and Ken-ichiro Ishii for their support.

## References

- James Allen. 1995. *Natural Language Understanding 2nd. Ed.* Benjamin Cummings.
- Andrew Borthwick. 1999. *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. thesis, New York University.
- Chris J. C. Burges and Bernhard Schölkopf. 1997. Improving speed and accuracy of support vector

- learning machines. In *Advances in Neural Information Processing Systems 9*, pages 375–381.
- Tom Downs, Kevin E. Gates, and Annette Masters. 2001. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297.
- Hideki Isozaki. 2001. Japanese named entity recognition based on a simple rule generator and decision tree learning. In *Proceedings of Association for Computational Linguistics*, pages 306–313.
- Tommi S. Jaakkola and David Haussler. 1998. Exploiting generative models in discriminative classifiers. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*. MIT Press.
- Thorsten Joachims. 1998. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*.
- Thorsten Joachims. 1999. Making large-scale support vector machine learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, chapter 16, pages 170–184. MIT Press.
- Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL*, pages 192–199.
- Tetsuji Nakagawa, Taku Kudoh, and Yuji Matsumoto. 2001. Unknown word guessing and part-of-speech tagging using support vector machines. In *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium*, pages 325–331.
- Edgar E. Osuna and Federico Girosi. 1999. Reducing the run-time complexity in support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, chapter 16, pages 271–283. MIT Press.
- John C. Platt, Nello Cristianini, and John Shawe-Taylor. 2000. Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems 12*, pages 547–553. MIT Press.
- John C. Platt. 1999. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*, chapter 12, pages 185–208. MIT Press.
- John C. Platt. 2000. Probabilities for SV machines. In A. J. Smola, P. L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, chapter 5, pages 61–71. MIT Press.
- Friedhelm Schwenker. 2001. Solving multi-class pattern recognition problems with tree-structured support vector machines. In B. Radig and S. Florczyk, editors, *Pattern Recognition, Proceedings of the 23rd Symposium*, number 2191 in LNCS, pages 283–290. Springer.
- Satoshi Sekine and Yoshio Eriguchi. 2000. Japanese named entity extraction evaluation — analysis of results —. In *Proceedings of 18th International Conference on Computational Linguistics*, pages 1106–1110.
- Satoshi Sekine, Ralph Grishman, and Hiroyuki Shinnou. 1998. A decision tree method for finding and classifying names in Japanese texts. In *Proceedings of the Sixth Workshop on Very Large Corpora*.
- Hiroshi Shimodaira, Ken-ichi Noma, Mitsuru Naka, and Shigeki Sagayama. 2001. Support vector machine with dynamic time-alignment kernel for speech recognition. In *Proceedings of Eurospeech*, pages 1841–1844.
- Koji Tsuda, M. Kawanabe, G. Rätsch, S. Sonnenburg, and K. Müller. 2001. A new discriminative kernel from probabilistic models. In *Advances in Neural Information Processing Systems 14*.
- Kiyotaka Uchimoto, Qing Ma, Masaki Murata, Hiromi Ozaku, Masao Utiyama, and Hitoshi Isahara. 2000. Named entity extraction based on a maximum entropy model and transformation rules (in Japanese). *Journal of Natural Language Processing*, 7(2):63–90.
- Takehito Utsuro, Manabu Sassano, and Kiyotaka Uchimoto. 2001. Learning to combine outputs of multiple Japanese named entity extractors (in Japanese). In *IPSJ SIG notes NL-144-5*.
- Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.
- E. M. Voorhees and D. K. Harman, editors. 2000. *Proceedings of the 9th Text Retrieval Conference*.
- Hiroyasu Yamada and Yuji Matsumoto. 2001. Applying support vector machine to multi-class classification problems (in Japanese). In *IPSJ SIG Notes NL-146-6*.
- Hiroyasu Yamada, Taku Kudoh, and Yuji Matsumoto. 2001. Japanese named entity extraction using support vector machines (in Japanese). In *IPSJ SIG Notes NL-142-17*.