Fast Methods for Kernel-based Text Analysis

Taku Kudo and **Yuji Matsumoto**
Graduate School of Information Science,
Nara Institute of Science and Technology
{taku-ku,matsu}@is.aist-nara.ac.jp

Abstract

Kernel-based learning (e.g., Support Vector Machines) has been successfully applied to many hard problems in Natural Language Processing (NLP). In NLP, although feature combinations are crucial to improving performance, they are heuristically selected. Kernel methods change this situation. The merit of the kernel methods is that *effective feature combination* is implicitly expanded without loss of generality and increasing the computational costs. Kernel-based text analysis shows an excellent performance in terms in accuracy; however, these methods are usually too slow to apply to large-scale text analysis. In this paper, we extend a *Basket Mining* algorithm to convert a kernel-based classifier into a simple and fast linear classifier. Experimental results on English BaseNP Chunking, Japanese Word Segmentation and Japanese Dependency Parsing show that our new classifiers are about 30 to 300 times faster than the standard kernel-based classifiers.

1 Introduction

Kernel methods (e.g., Support Vector Machines (Vapnik, 1995)) attract a great deal of attention recently. In the field of Natural Language Processing, many successes have been reported. Examples include Part-of-Speech tagging (Nakagawa et al., 2002) Text Chunking (Kudo and Matsumoto, 2001), Named Entity Recognition (Isozaki and Kazawa, 2002), and Japanese Dependency Parsing (Kudo and Matsumoto, 2000; Kudo and Matsumoto, 2002).

It is known in NLP that combination of features contributes to a significant improvement in accuracy. For instance, in the task of dependency parsing, it would be hard to confirm a correct dependency relation with only a single set of features from either a head or its modifier. Rather, dependency relations should be determined by at least information from both of two phrases. In previous research, feature combination has been selected manually, and the performance significantly depended on these selections. This is not the case with kernel-based methodology. For instance, if we use a polynomial kernel, all feature combinations are implicitly expanded without loss of generality and increasing the computational costs. Although the mapped feature space is quite large, the maximal margin strategy (Vapnik, 1995) of SVMs gives us a good generalization performance compared to the previous manual feature selection. This is the main reason why kernel-based learning has delivered great results to the field of NLP.

Kernel-based text analysis shows an excellent performance in terms in accuracy; however, its inefficiency in actual analysis limits practical application. For example, an SVM-based NE-chunker runs at a rate of only 85 byte/sec, while previous rule-based system can process several kilobytes per second (Isozaki and Kazawa, 2002). Such slow execution time is inadequate for Information Retrieval, Question Answering, or Text Mining, where fast analysis of large quantities of text is indispensable.

This paper presents two novel methods that make the kernel-based text analyzers substantially faster. These methods are applicable not only to the NLP tasks but also to general machine learning tasks where training and test examples are represented in a binary vector.

More specifically, we focus on a *Polynomial Kernel* of degree d, which can attain feature combinations that are crucial to improving the performance of tasks in NLP. Second, we introduce two fast classification algorithms for this kernel. One is PKI (Polynomial Kernel Inverted), which is an extension of *Inverted Index* in Information Retrieval. The other is PKE (Polynomial Kernel Expanded), where all feature combinations are explicitly expanded. By applying PKE, we can convert a kernel-based classifier into a simple and fast liner classifier. In order to build PKE, we extend the *PrefixSpan* (Pei et al., 2001), an efficient *Basket Mining* algorithm, to enumerate effective feature combinations from a set of support examples.

Experiments on English BaseNP Chunking, Japanese Word Segmentation and Japanese Dependency Parsing show that PKI and PKE perform respectively 2 to 13 times and 30 to 300 times faster than standard kernel-based systems, without a discernible change in accuracy.

2 Kernel Method and Support Vector Machines

Suppose we have a set of training data for a binary classification problem:

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_L, y_L) \;\; \mathbf{x}_j \in \Re^N, \;\; y_j \in \{+1, -1\},$$

where $\mathbf{x}_j$ is a feature vector of the j-th training sample, and y_j is the class label associated with this training sample. The decision function of SVMs is defined by

$$y(\mathbf{x}) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j \phi(\mathbf{x}_j) \cdot \phi(\mathbf{x}) + b\Big), \qquad (1)$$

where: (A) ϕ is a non-liner mapping function from $\Re^N$ to $\Re^H$ ($N \ll H$). (B) $\alpha_j, b \in \Re, \;\alpha_j \geq 0$.

The mapping function ϕ should be designed such that all training examples are linearly separable in $\Re^H$ space. Since H is much larger than N, it requires heavy computation to evaluate the dot products $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$ in an explicit form. This problem can be overcome by noticing that both construction of optimal parameter α_i (we will omit the details of this construction here) and the calculation of the decision function only require the evaluation of dot products $\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})$. This is critical, since, in some cases, the dot products can be evaluated by a simple *Kernel Function*: $K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2)$. Substituting kernel function into (1), we have the following decision function.

$$y(\mathbf{x}) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}) + b\Big) \qquad (2)$$

One of the advantages of kernels is that they are not limited to vectorial object $\mathbf{x}$, but that they are applicable to any kind of object representation, just given the dot products.

3 Polynomial Kernel of degree d

For many tasks in NLP, the training and test examples are represented in binary vectors; or *sets*, since examples in NLP are usually represented in so-called *Feature Structures*. Here, we focus on such cases [1].

Suppose a feature set $F = \{1, 2, \ldots, N\}$ and training examples $X_j (j = 1, 2, \ldots, L)$, all of which are subsets of F (i.e., $X_j \subseteq F$). In this case, X_j can be regarded as a binary vector $\mathbf{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jN})$ where $x_{ji} = 1$ if $i \in X_j$, $x_{ji} = 0$ otherwise. The dot product of $\mathbf{x}_1$ and $\mathbf{x}_2$ is given by $\mathbf{x}_1 \cdot \mathbf{x}_2 = |X_1 \cap X_2|$.

Definition 1 *Polynomial Kernel of degree* d
Given sets X *and* Y*, corresponding to binary feature vectors* $\mathbf{x}$ *and* $\mathbf{y}$*, Polynomial Kernel of degree* d $K_d(X, Y)$ *is given by*

$$K_d(\mathbf{x}, \mathbf{y}) = K_d(X, Y) = (1 + |X \cap Y|)^d, \qquad (3)$$

where $d = 1, 2, 3, \ldots$.

In this paper, (3) will be referred to as an *implicit form* of the Polynomial Kernel.

[1] In the Maximum Entropy model widely applied in NLP, we usually suppose binary feature functions $f_i(X_j) \in \{0, 1\}$. This formalization is exactly same as representing an example X_j in a *set* $\{k | f_k(X_j) = 1\}$.

It is known in NLP that a combination of features, a subset of feature set F in general, contributes to overall accuracy. In previous research, feature combination has been selected manually. The use of a polynomial kernel allows such feature expansion without loss of generality or an increase in computational costs, since the Polynomial Kernel of degree d implicitly maps the original feature space F into F^d space. (i.e., $\phi : F \rightarrow F^d$). This property is critical and some reports say that, in NLP, the polynomial kernel outperforms the simple linear kernel (Kudo and Matsumoto, 2000; Isozaki and Kazawa, 2002).

Here, we will give an explicit form of the Polynomial Kernel to show the mapping function $\phi(\cdot)$.

Lemma 1 *Explicit form of Polynomial Kernel.*
The Polynomial Kernel of degree d can be rewritten as

$$K_d(X,Y) = \sum_{r=0}^{d} c_d(r) \cdot |P_r(X \cap Y)|, \qquad (4)$$

where

- $P_r(X)$ *is a set of all subsets of* X *with exactly* r *elements in it,*
- $c_d(r) = \sum_{l=r}^{d} \binom{d}{l} \Big(\sum_{m=0}^{r} (-1)^{r-m} \cdot m^l \binom{r}{m} \Big)$.

Proof See Appendix A.
$c_d(r)$ will be referred as a *subset weight* of the Polynomial Kernel of degree d. This function gives a *prior weight* to the subset s, where $|s| = r$.

Example 1 *Quadratic and Cubic Kernel*
Given sets $X = \{a,b,c,d\}$ *and* $Y = \{a,b,d,e\}$, *the Quadratic Kernel* $K_2(X,Y)$ *and the Cubic Kernel* $K_3(X,Y)$ *can be calculated in an implicit form as:*
$K_2(X,Y) = (1 + |X \cap Y|)^2 = (1+3)^2 = 16$,
$K_3(X,Y) = (1 + |X \cap Y|)^3 = (1+3)^3 = 64$.

Using Lemma 1, the subset weights of the Quadratic Kernel and the Cubic Kernel can be calculated as $c_2(0) = 1,\ c_2(1) = 3,\ c_2(2) = 2$ *and* $c_3(0)=1,\ c_3(1)=7,\ c_3(2)=12,\ c_3(3)=6$.

In addition, subsets $P_r(X \cap Y)$ $(r = 0,1,2,3)$ *are given as follows:* $P_0(X \cap Y) = \{\phi\},\ P_1(X \cap Y) = \{\{a\},\{b\},\{d\}\},\ P_2(X \cap Y) = \{\{a,b\},\{a,d\},\{b,d\}\},\ P_3(X \cap Y) = \{\{a,b,d\}\}$. $K_2(X,Y)$ *and* $K_3(X,Y)$ *can similarly be calculated in an explicit form as:*

$K_2(X,Y) = 1 \cdot 1 + 3 \cdot 3 + 2 \cdot 3 = 16$,
$K_3(X,Y) = 1 \cdot 1 + 7 \cdot 3 + 12 \cdot 3 + 6 \cdot 1 = 64$.

```
function PKI_classify (X)
  r = 0 # an array, initialized as 0
  foreach i ∈ X
    foreach j ∈ h(i)
      r_j = r_j + 1
    end
  end
  result = 0
  foreach j ∈ SV
    result = result + y_j α_j · (1 + r_j)^d
  end
  return sgn(result + b)
end
```

Figure 1: Pseudo code for PKI

4 Fast Classifiers for Polynomial Kernel

In this section, we introduce two fast classification algorithms for the Polynomial Kernel of degree d.

Before describing them, we give the baseline classifier (**PKB**):

$$y(X) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j \cdot (1 + |X_j \cap X|)^d + b \Big). \quad (5)$$

The complexity of PKB is $O(|X| \cdot |SV|)$, since it takes $O(|X|)$ to calculate $(1 + |X_j \cap X|)^d$ and there are a total of $|SV|$ support examples.

4.1 PKI (Inverted Representation)

Given an item $i \in F$, if we know in advance the set of support examples which contain item $i \in F$, we do not need to calculate $|X_j \cap X|$ for all support examples. This is a naive extension of *Inverted Indexing* in Information Retrieval. Figure 1 shows the pseudo code of the algorithm PKI. The function $h(i)$ is a pre-compiled table and returns a set of support examples which contain item i.

The complexity of the PKI is $O(|X| \cdot B + |SV|)$, where B is an average of $|h(i)|$ over all item $i \in F$. The PKI can make the classification speed drastically faster when B is small, in other words, when feature space is relatively sparse (i.e., $B \ll |SV|$). The feature space is often sparse in many tasks in NLP, since lexical entries are used as features.

The algorithm PKI does not change the final accuracy of the classification.

4.2 PKE (Expanded Representation)

4.2.1 Basic Idea of PKE

Using Lemma 1, we can represent the decision function (5) in an explicit form:

$$y(X) = \mathrm{sgn}\Big(\sum_{j \in SV} y_j \alpha_j \big(\sum_{r=0}^{d} c_d(r) \cdot |P_r(X_j \cap X)|\big) + b\Big). \quad (6)$$

If we, in advance, calculate

$$w(s) = \sum_{j \in SV} y_j \alpha_j c_d(|s|) I(s \in P_{|s|}(X_j))$$

(where $I(t)$ is an indicator function [2]) for all subsets $s \in \bigcup_{r=0}^{d} P_r(F)$, (6) can be written as the following simple linear form:

$$y(X) \;=\; \mathrm{sgn}\Big(\sum_{s \in \Gamma_d(X)} w(s) + b\Big). \quad (7)$$

where $\Gamma_d(X) = \bigcup_{r=0}^{d} P_r(X)$.

The classification algorithm given by (7) will be referred to as **PKE**. The complexity of PKE is $O(|\Gamma_d(X)|) = O(|X|^d)$, independent on the number of support examples $|SV|$.

4.2.2 Mining Approach to PKE

To apply the PKE, we first calculate $|\Gamma_d(F)|$ degree of vectors

$$\mathbf{w} = (w(s_1), w(s_2), \ldots, w(s_{|\Gamma_d(F)|})).$$

This calculation is trivial only when we use a Quadratic Kernel, since we just project the original feature space F into $F \times F$ space, which is small enough to be calculated by a naive exhaustive method. However, if we, for instance, use a polynomial kernel of degree 3 or higher, this calculation becomes not trivial, since the size of feature space exponentially increases. Here we take the following strategy:

1. Instead of using the original vector $\mathbf{w}$, we use $\mathbf{w}'$, an approximation of $\mathbf{w}$.

2. We apply the *Subset Mining* algorithm to calculate $\mathbf{w}'$ efficiently.

[2] $I(t)$ returns 1 if t is true,returns 0 otherwise.

Definition 2 $\mathbf{w}'$*: An approximation of* $\mathbf{w}$

An approximation of $\mathbf{w}$ *is given by* $\mathbf{w}' = (w'(s_1), w'(s_2), \ldots, w'(s_{|\Gamma_d(F)|}))$*, where* $w'(s)$ *is set to 0 if* $w(s)$ *is trivially close to 0. (i.e.,* $\sigma_{neg} < w(s) < \sigma_{pos}$ $(\sigma_{neg} < 0,\ \sigma_{pos} > 0)$*, where* σ_{pos} *and* σ_{neg} *are predefined thresholds).*

The algorithm PKE is an approximation of the PKB, and changes the final accuracy according to the selection of thresholds σ_{pos} and σ_{neg}. The calculation of $\mathbf{w}'$ is formulated as the following mining problem:

Definition 3 Feature Combination Mining

Given a set of support examples and subset weight $c_d(r)$*, extract all subsets* s *and their weights* $w(s)$ *if* $w(s)$ *holds* $w(s) \geq \sigma_{pos}$ *or* $w(s) \leq \sigma_{neg}$ *.*

In this paper, we apply a *Sub-Structure Mining* algorithm to the feature combination mining problem. Generally speaking, sub-structures mining algorithms efficiently extract *frequent* sub-structures (e.g., subsets, sub-sequences, sub-trees, or sub-graphs) from a large database (set of transactions). In this context, *frequent* means that there are no less than ξ transactions which contain a sub-structure. The parameter ξ is usually referred to as the *Minimum Support*. Since we must enumerate all subsets of F, we can apply subset mining algorithm, in some times called as *Basket Mining* algorithm, to our task.

There are many subset mining algorithms proposed, however, we will focus on the *PrefixSpan* algorithm, which is an efficient algorithm for sequential pattern mining, originally proposed by (Pei et al., 2001). The PrefixSpan was originally designed to extract frequent sub-sequence (not subset) patterns, however, it is a trivial difference since a set can be seen as a special case of sequences (i.e., by sorting items in a set by lexicographic order, the set becomes a sequence). The basic idea of the PrefixSpan is to divide the database by frequent sub-patterns (prefix) and to grow the prefix-spanning pattern in a depth-first search fashion.

We now modify the PrefixSpan to suit to our feature combination mining.

- size constraint
  We only enumerate up to subsets of size d. when we plan to apply the Polynomial Kernel of degree d.

- Subset weight $c_d(r)$
  In the original PrefixSpan, the frequency of each subset does not change by its size. However, in our mining task, it changes (i.e., the frequency of subset s is weighted by $c_d(|s|)$). Here, we process the mining algorithm by assuming that each transaction (support example X_j) has its frequency $C_d y_j \alpha_j$, where $C_d = \max(c_d(1), c_d(2), \ldots, c_d(d))$. The weight $w(s)$ is calculated by $w(s) = \omega(s) \times c_d(|s|)/C_d$, where $\omega(s)$ is a frequency of s, given by the original PrefixSpan.

- Positive/Negative support examples
  We first divide the support examples into positive ($y_i > 0$) and negative ($y_i < 0$) examples, and process mining independently. The result can be obtained by merging these two results.

- Minimum Supports $\sigma_{pos}, \sigma_{neg}$
  In the original PrefixSpan, minimum support is an integer. In our mining task, we can give a real number to minimum support, since each transaction (support example X_j) has possibly non-integer frequency $C_d y_j \alpha_j$. Minimum supports σ_{pos} and σ_{neg} control the rate of approximation. For the sake of convenience, we just give one parameter σ, and calculate σ_{pos} and σ_{neg} as follows

$$\sigma_{pos} = \sigma \cdot \Big(\frac{\#of\ positive\ examples}{\#of\ support\ examples}\Big),$$
$$\sigma_{neg} = -\sigma \cdot \Big(\frac{\#of\ negative\ examples}{\#of\ support\ examples}\Big).$$

After the process of mining, a set of tuples $\Omega = \{\langle s, w(s)\rangle\}$ is obtained, where s is a frequent subset and $w(s)$ is its weight. We use a TRIE to efficiently store the set Ω. The example of such TRIE compression is shown in Figure 2. Although there are many implementations for TRIE, we use a Double-Array (Aoe, 1989) in our task. The actual classification of PKE can be examined by traversing the TRIE for all subsets $s \in \Gamma_d(X)$.

5 Experiments

To demonstrate performances of PKI and PKE, we examined three NLP tasks: English BaseNP Chunking (EBC), Japanese Word Segmentation (JWS) and

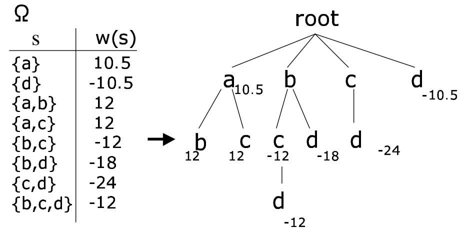

Figure 2: Ω in TRIE representation

Japanese Dependency Parsing (JDP). A more detailed description of each task, training and test data, the system parameters, and feature sets are presented in the following subsections. Table 1 summarizes the detail information of support examples (e.g., size of SVs, size of feature set etc.).

Our preliminary experiments show that a Quadratic Kernel performs the best in EBC, and a Cubic Kernel performs the best in JWS and JDP. The experiments using a Cubic Kernel are suitable to evaluate the effectiveness of the basket mining approach applied in the PKE, since a Cubic Kernel projects the original feature space F into F^3 space, which is too large to be handled only using a naive exhaustive method.

All experiments were conducted under Linux using XEON 2.4 Ghz dual processors and 3.5 Gbyte of main memory. All systems are implemented in C++.

5.1 English BaseNP Chunking (EBC)

Text Chunking is a fundamental task in NLP – dividing sentences into non-overlapping phrases. BaseNP chunking deals with a part of this task and recognizes the chunks that form noun phrases. Here is an example sentence:

```
[He] reckons [the current account deficit]
will narrow to [only $ 1.8 billion] .
```

A BaseNP chunk is represented as sequence of words between square brackets. BaseNP chunking task is usually formulated as a simple tagging task, where we represent chunks with three types of tags: **B**: beginning of a chunk. **I**: non-initial word. **O**: outside of the chunk. In our experiments, we used the same settings as (Kudo and Matsumoto, 2002). We use a standard data set (Ramshaw and Marcus, 1995) consisting of sections 15-19 of the WSJ corpus as training and section 20 as testing.

5.2 Japanese Word Segmentation (JWS)

Since there are no explicit spaces between words in Japanese sentences, we must first identify the word boundaries before analyzing deep structure of a sentence. Japanese word segmentation is formalized as a simple classification task.

Let $s = c_1c_2 \cdots c_m$ be a sequence of Japanese characters, $t = t_1t_2 \cdots t_m$ be a sequence of Japanese character types [3] associated with each character, and $y_i \in \{+1, -1\}$, $(i = (1, 2, \ldots, m-1))$ be a boundary marker. If there is a boundary between c_i and c_{i+1}, $y_i = 1$, otherwise $y_i = -1$. The feature set of example x_i is given by all characters as well as character types in some constant window (e.g., 5): $\{c_{i-2}, c_{i-1}, \cdots, c_{i+2}, c_{i+3}, t_{i-2}, t_{i-1}, \cdots, t_{i+2}, t_{i+3}\}$. Note that we distinguish the relative position of each character and character type. We use the Kyoto University Corpus (Kurohashi and Nagao, 1997), 7,958 sentences in the articles on January 1st to January 7th are used as training data, and 1,246 sentences in the articles on January 9th are used as the test data.

5.3 Japanese Dependency Parsing (JDP)

The task of Japanese dependency parsing is to identify a correct dependency of each *Bunsetsu* (base phrase in Japanese). In previous research, we presented a state-of-the-art SVMs-based Japanese dependency parser (Kudo and Matsumoto, 2002). We combined SVMs into an efficient parsing algorithm, *Cascaded Chunking Model*, which parses a sentence deterministically only by deciding whether the current chunk modifies the chunk on its immediate right hand side. The input for this algorithm consists of a set of the linguistic features related to the head and modifier (e.g., word, part-of-speech, and inflections), and the output from the algorithm is either of the value +1 (dependent) or -1 (independent). We use a standard data set, which is the same corpus described in the Japanese Word Segmentation.

[3] Usually, in Japanese, word boundaries are highly constrained by character types, such as *hiragana* and *katakana* (both are phonetic characters in Japanese), Chinese characters, English alphabets and numbers.

5.4 Results

Tables 2, 3 and 4 show the execution time, accuracy[4], and $|\Omega|$ (size of extracted subsets), by changing σ from 0.01 to 0.0005.

The PKI leads to about 2 to 12 times improvements over the PKB. In JDP, the improvement is significant. This is because B, the average of $h(i)$ over all items $i \in F$, is relatively small in JDP. The improvement significantly depends on the sparsity of the given support examples.

The improvements of the PKE are more significant than the PKI. The running time of the PKE is 30 to 300 times faster than the PKB, when we set an appropriate σ, (e.g., $\sigma = 0.005$ for EBC and JWS, $\sigma = 0.0005$ for JDP). In these settings, we could preserve the final accuracies for test data.

5.5 Frequency-based Pruning

The PKE with a Cubic Kernel tends to make Ω large (e.g., $|\Omega| = 2.32$ million for JWS, $|\Omega| = 8.26$ million for JDP).

To reduce the size of Ω, we examined simple frequency-based pruning experiments. Our extension is to simply give a prior threshold $\xi(= 1, 2, 3, 4 \ldots)$, and erase all subsets which occur in less than ξ support examples. The calculation of frequency can be similarly conducted by the *PrefixSpan* algorithm. Tables 5 and 6 show the results of frequency-based pruning, when we fix σ=0.005 for JWS, and σ=0.0005 for JDP.

In JDP, we can make the size of set Ω about one third of the original size. This reduction gives us not only a slight speed increase but an improvement of accuracy (89.29%→89.34%). Frequency-based pruning allows us to remove subsets that have large weight and small frequency. Such subsets may be generated from errors or special outliers in the training examples, which sometimes cause an overfitting in training.

In JWS, the frequency-based pruning does not work well. Although we can reduce the size of Ω by half, the accuracy is also reduced (97.94%→97.83%). It implies that, in JWS, features even with frequency of one contribute to the final decision hyperplane.

[4] In EBC, accuracy is evaluated using F measure, harmonic mean between precision and recall.

Table 1: Details of Data Set

Data Set	EBC	JWS	JDP
# of examples	135,692	265,413	110,355
$\lvert SV \rvert$ # of SVs	11,690	57,672	34,996
# of positive SVs	5,637	28,440	17,528
# of negative SVs	6,053	29,232	17,468
$\lvert F \rvert$ (size of feature)	17,470	11,643	28,157
Avg. of $\lvert X_j \rvert$	11.90	11.73	17.63
B (Avg. of $\lvert h(i) \rvert$))	7.74	58.13	21.92

(Note: In EBC, to handle K-class problems, we use a *pairwise classification*; building $K \times (K-1)/2$ classifiers considering all pairs of classes, and final class decision was given by majority voting. The values in this column are averages over all pairwise classifiers.)

6 Discussion

There have been several studies for efficient classification of SVMs. Isozaki et al. propose an XQK (eXpand the Quadratic Kernel) which can make their Named-Entity recognizer drastically fast (Isozaki and Kazawa, 2002). XQK can be subsumed into PKE. Both XQK and PKE share the basic idea; all feature combinations are explicitly expanded and we convert the kernel-based classifier into a simple linear classifier.

The explicit difference between XQK and PKE is that XQK is designed only for Quadratic Kernel. It implies that XQK can only deal with feature combination of size up to two. On the other hand, PKE is more general and can also be applied not only to the Quadratic Kernel but also to the general-style of polynomial kernels $(1 + |X \cap Y|)^d$. In PKE, there are no theoretical constrains to limit the size of combinations.

In addition, Isozaki et al. did not mention how to expand the feature combinations. They seem to use a naive exhaustive method to expand them, which is not always scalable and efficient for extracting three or more feature combinations. PKE takes a basket mining approach to enumerating effective feature combinations more efficiently than their exhaustive method.

7 Conclusion and Future Works

We focused on a *Polynomial Kernel* of degree d, which has been widely applied in many tasks in NLP

Table 2: Results of EBC

PKE σ	Time (sec./sent.)	Speedup Ratio	F1	$\lvert\Omega\rvert$ ($\times$ 1000)
0.01	0.0016	105.2	93.79	518
0.005	0.0016	101.3	93.85	668
0.001	0.0017	97.7	93.84	858
0.0005	0.0017	96.8	93.84	889
PKI	0.020	8.3	93.84	
PKB	0.164	1.0	93.84	

Table 3: Results of JWS

PKE σ	Time (sec./sent.)	Speedup Ratio	Acc.(%)	$\lvert\Omega\rvert$ ($\times$ 1000)
0.01	0.0024	358.2	97.93	1,228
0.005	0.0028	300.1	97.95	2,327
0.001	0.0034	242.6	97.94	4,392
0.0005	0.0035	238.8	97.94	4,820
PKI	0.4989	1.7	97.94	
PKB	0.8535	1.0	97.94	

Table 4: Results of JDP

PKE σ	Time (sec./sent.)	Speedup Ratio	Acc.(%)	$\lvert\Omega\rvert$ ($\times$ 1000)
0.01	0.0042	66.8	88.91	73
0.005	0.0060	47.8	89.05	1,924
0.001	0.0086	33.3	89.26	6,686
0.0005	0.0090	31.8	89.29	8,262
PKI	0.0226	12.6	89.29	
PKB	0.2848	1.0	89.29	

Table 5: Frequency-based pruning (JWS)

PKE ξ	time (sec./sent.)	Speedup Ratio	Acc.(%)	$\lvert\Omega\rvert$ ($\times$ 1000)
1	0.0028	300.1	97.95	2,327
2	0.0025	337.3	97.83	954
3	0.0023	367.0	97.83	591
PKB	0.8535	1.0	97.94	

Table 6: Frequency-based pruning (JDP)

PKE ξ	time (sec./sent.)	Speedup Ratio	Acc.(%)	$\lvert\Omega\rvert$ ($\times$ 1000)
1	0.0090	31.8	89.29	8,262
2	0.0072	39.3	89.34	2,450
3	0.0068	41.8	89.31	1,360
PKB	0.2848	1.0	89.29	

and can attain feature combination that is crucial to improving the performance of tasks in NLP. Then, we introduced two fast classification algorithms for this kernel. One is PKI (Polynomial Kernel Inverted), which is an extension of *Inverted Index*. The other is PKE (Polynomial Kernel Expanded), where all feature combinations are explicitly expanded.

The concept in PKE can also be applicable to kernels for discrete data structures, such as String Kernel (Lodhi et al., 2002) and Tree Kernel (Kashima and Koyanagi, 2002; Collins and Duffy, 2001). For instance, Tree Kernel gives a dot product of an ordered-tree, and maps the original ordered-tree onto its all sub-tree space. To apply the PKE, we must efficiently enumerate the effective sub-trees from a set of support examples. We can similarly apply a sub-tree mining algorithm (Zaki, 2002) to this problem.

Appendix A.: Lemma 1 and its proof

$$c_d(r) = \sum_{l=r}^{d} \binom{d}{l} \Big(\sum_{m=0}^{r} (-1)^{r-m} \cdot m^l \binom{r}{m} \Big).$$

Proof.
Let X, Y be subsets of $F = \{1, 2, \ldots, N\}$. In this case, $|X \cap Y|$ is same as the dot product of vector $\mathbf{x}, \mathbf{y}$, where

$$\mathbf{x} = \{x_1, x_2, \ldots, x_N\},\ \mathbf{y} = \{y_1, y_2, \ldots, y_N\} \quad (x_j, y_j \in \{0, 1\})$$

$x_j = 1$ if $j \in X$, $x_j = 0$ otherwise.

$(1 + |X \cap Y|)^d = (1 + \mathbf{x} \cdot \mathbf{y})^d$ can be expanded as follows

$$\begin{aligned} (1 + \mathbf{x} \cdot \mathbf{y})^d &= \sum_{l=0}^{d} \binom{d}{l} \Big(\sum_{j=1}^{N} x_j y_j \Big)^l \\ &= \sum_{l=0}^{d} \binom{d}{l} \cdot \tau(l) \end{aligned}$$

where

$$\tau(l) = \sum_{k_n \geq 0}^{k_1 + \ldots + k_N = l} \frac{l!}{k_1! \ldots k_N!} (x_1 y_1)^{k_1} \ldots (x_N y_N)^{k_N}.$$

Note that $x_j^{k_j}$ is binary ($i.e., x_j^{k_j} \in \{0, 1\}$), the number of r-size subsets can be given by a coefficient of $(x_1 y_1 x_2 y_2 \ldots x_r y_r)$. Thus,

$$\begin{aligned} c_d(r) &= \sum_{l=r}^{d} \binom{d}{l} \Big(\sum_{k_n \geq 1, n=1,2,\ldots,r}^{k_1 + \ldots + k_r = l} \frac{l!}{k_1! \ldots k_r!} \Big) \\ &= \sum_{l=r}^{d} \binom{d}{l} \Big(r^l - \binom{r}{1} (r-1)^l + \binom{r}{2} (r-2)^l - \ldots \Big) \\ &= \sum_{l=r}^{d} \binom{d}{l} \Big(\sum_{m=0}^{r} (-1)^{r-m} \cdot m^l \binom{r}{m} \Big). \quad \square \end{aligned}$$

References

Junichi Aoe. 1989. An efficient digital search algorithm by using a double-array structure. *IEEE Transactions on Software Engineering*, 15(9).

Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Advances in Neural Information Processing Systems 14, Vol.1 (NIPS 2001)*, pages 625–632.

Hideki Isozaki and Hideto Kazawa. 2002. Efficient support vector classifiers for named entity recognition. In *Proceedings of the COLING-2002*, pages 390–396.

Hisashi Kashima and Teruo Koyanagi. 2002. Svm kernels for semi-structured data. In *Proceedings of the ICML-2002*, pages 291–298.

Taku Kudo and Yuji Matsumoto. 2000. Japanese Dependency Structure Analysis based on Support Vector Machines. In *Proceedings of the EMNLP/VLC-2000*, pages 18–25.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of the the NAACL*, pages 192–199.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analyisis using cascaded chunking. In *Proceedings of the CoNLL-2002*, pages 63–69.

Sadao Kurohashi and Makoto Nagao. 1997. Kyoto University text corpus project. In *Proceedings of the ANLP-1997*, pages 115–118.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. 2002. Text classification using string kernels. *Journal of Machine Learning Research*, 2.

Tetsuji Nakagawa, Taku Kudo, and Yuji Matsumoto. 2002. Revision learning and its application to part-of-speech tagging. In *Proceedings of the ACL 2002*, pages 497–504.

Jian Pei, Jiawei Han, and et al. 2001. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proc. of International Conference of Data Engineering*, pages 215–224.

Lance A. Ramshaw and Mitchell P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the VLC*, pages 88–94.

Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer.

Mohammed Zaki. 2002. Efficiently mining frequent trees in a forest. In *Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining KDD*, pages 71–80.