An Approximate Approach for Training Polynomial Kernel SVMs in Linear Time

Yu-Chieh Wu
Dept. of Computer Science and Information Engineering
National Central University
Taoyuan, Taiwan
bcbb@db.csie.ncu.edu.tw

Jie-Chi Yang
Graduate Institute of Network Learning Technology
National Central University
Taoyuan, Taiwan
yang@cl.ncu.edu.tw

Yue-Shi Lee
Dept. of Computer Science and Information Engineering
Ming Chuan University
Taoyuan, Taiwan
lees@mcu.edu.tw

Abstract

Kernel methods such as support vector machines (SVMs) have attracted a great deal of popularity in the machine learning and natural language processing (NLP) communities. Polynomial kernel SVMs showed very competitive accuracy in many NLP problems, like part-of-speech tagging and chunking. However, these methods are usually too inefficient to be applied to large dataset and real time purpose. In this paper, we propose an approximate method to analogy polynomial kernel with efficient data mining approaches. To prevent exponential-scaled testing time complexity, we also present a new method for speeding up SVM classifying which does independent to the polynomial degree d. The experimental results showed that our method is 16.94 and 450 times faster than traditional polynomial kernel in terms of training and testing respectively.

1 Introduction

Kernel methods, for example support vector machines (SVM) (Vapnik, 1995) are successfully applied to many natural language processing (NLP) problems. They yielded very competitive and satisfactory performance in many classification tasks, such as part-of-speech (POS) tagging (Gimenez and Marquez, 2003), shallow parsing (Kudo and Matsumoto, 2001, 2004; Lee and Wu, 2007), named entity recognition (Isozaki and Kazawa, 2002), and parsing (Nivre et al., 2006).

In particular, the use of polynomial kernel SVM implicitly takes the feature combinations into account instead of explicitly combines features. By setting with polynomial kernel degree (i.e., d), different number of feature conjunctions can be implicitly computed. In this way, polynomial kernel SVM is often better than linear kernel which did not use feature conjunctions. However, the training and testing time costs for polynomial kernel SVM is far slow than the linear kernel. For example, it took one day to train the CoNLL-2000 task with polynomial kernel SVM, while the testing speed is merely 20-30 words per second (Kudo and Matsumoto, 2001). Although the author provided the solution for fast classifying with polynomial kernel (Kudo and Matsumoto, 2004), the training time is still inefficient. Nevertheless, the testing time of their method exponentially scales with polynomial kernel degree d, i.e., $O(|X|^d)$ where $|X|$ denotes as the length of example X.

On the contrary, even the linear kernel SVM simply disregards the effect of feature combinations during training and testing, it performs not only more efficient than polynomial kernel, but also can be improved through directly appending features derived from the set of feature combinations. Examples include bigram, trigram, etc. Nevertheless, selecting the feature conjunctions was manually and heuristically encoded and should perform amount of validation trials to discover which is useful or not. In recent years, several studies had reported that the training time of linear kernel SVM can be reduced to linear time (Joachims, 2006; Keerthi and DeCoste, 2005). But they did not and difficult to be extent to polynomial kernels.

In this paper, we propose an approximate approach to extend the linear kernel SVM toward polynomial. By introducing the well-known sequential pattern mining approach (Pei et al., 2004),

Proceedings of the ACL 2007 Demo and Poster Sessions, pages 65–68,
Prague, June 2007.

frequent feature conjunctions, namely patterns could be discovered and also kept as expand feature space. We then adopt the mined patterns to re-represent the training/testing examples. Subsequently, we use the off-the-shelf linear kernel SVM algorithm to perform training and testing. Besides, to exponential-scaled testing time complexity, we propose a new classification method for speeding up the SVM testing. Rather than enumerating all patterns for each example, our method requires $O(F_{avg}*N_{avg})$ which is independent to the polynomial kernel degree. F_{avg} is the average number of frequent features per example, while the N_{avg} is the average number of patterns per feature.

2 SVM and Kernel Methods

Suppose we have the training instance set for binary classification problem:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),\ x_i \in \Re^D,\ y_i \in \{+1, -1\}$$

where x_i is a feature vector in D-dimension space of the i-th example, and y_i is the label of xi either positive or negative. The training of SVMs involves in minimize the following object (primal form, soft-margin) (Vapnik, 1995):

$$\text{minimize}: W(\alpha) = \frac{1}{2}\vec{W} \cdot \vec{W} + C\sum_{i=1}^{n} Loss(\vec{W}x_i, y_i) \quad (1)$$

The loss function indicates the loss of training error. Usually, the hinge-loss is used (Keerthi and DeCoste, 2005). The factor C in (1) is a parameter that allows one to trade off training error and margin. A small value for C will increase the number of training errors.

To determine the class (+1 or -1) of an example x can be judged by computing the following equation.

$$y(x) = \text{sign}((\sum_{x_i \in SVs} \alpha_i y_i K(x, x_i)) + b) \quad (2)$$

α_i is the weight of training example x_i (α_i>0), and b denotes as a threshold. Here the xi should be the support vectors (SVs), and are representative of training examples. The kernel function K is the kernel mapping function, which might map from $\Re^D$ to $\Re^{D'}$ (usually D<<D'). The natural linear kernel simply uses the dot-product as (3).

$$K(x, x_i) = dot(x, x_i) \quad (3)$$

A polynomial kernel of degree d is given by (4).

$$K(x, x_i) = (1 + dot(x, x_i))^d \quad (4)$$

One can design or employ off-the-shelf kernel types for particular applications. In particular to the use of polynomial kernel-based SVM, it was shown to be the most successful kernels for many natural language processing (NLP) problems (Kudo and Matsumoto, 2001; Isozaki and Kazawa, 2002; Nivre et al., 2006).

It is known that the dot-product (linear form) represents the most efficient kernel computing which can produce the output value by linearly combining all support vectors such as

$$y(x) = \text{sign}(dot(x, w) + b) \qquad \text{where } w = \sum_{x_i \in SVs} \alpha_i y_i x_i \quad (5)$$

By combining (2) and (4), the determination of an example of x using the polynomial kernel can be shown as follows.

$$y(x) = \text{sign}((\sum_{x_i \in SVs} \alpha_i y_i (dot(x, x_i) + 1)^d) + b) \quad (6)$$

Usually, degree d is set more than 1. When d is set as 1, the polynomial kernel backs-off to linear kernel. Although the effectiveness of polynomial kernel, it can not be shown to linearly combine all support vectors into one weight vector whereas it requires computing the kernel function (4) for each support vector x_i. The situation is even worse when the number of support vectors become huge (Kudo and Matsumoto, 2004). Therefore, whether in training or testing phrase, the cost of kernel computations is far more expensive than linear kernel.

3 Approximate Polynomial Kernel

In 2004, Kudo and Matsumoto (2004) derived both implicitly (6) and explicitly form of polynomial kernel. They indicated that the use of explicitly enumerate the feature combinations is equivalent to the polynomial kernel (see Lemma 1 and Example 1, Kudo and Matsumoto, 2004) which shared the same view of (Cumby and Roth, 2003).

We follow the similar idea of the above studies that requires explicitly enumerated all feature combinations. To meet with our problem, we employ the well-known sequential pattern mining algorithm, namely PrefixSpan (Pei et al., 2004) to efficient mine the frequent patterns. However, directly adopt the algorithm is not a good idea. To fit with SVM, we modify the original PrefixSpan algorithm according to the following constraints.

Given a set features, the PrefixSpan mines the frequent patterns which occurs more than predefined minimum support in the training set and limited in the length of predefined d, which is equivalent to the polynomial kernel degree d. For example, if the minimum support is 5, and d=2, then a feature combination (f_i, f_j) must appear more than 5 times in set of x.

Definition 1 (*Frequent single-item sequence*):
Given a set of feature vectors x, minimum support, and d, mining the frequent patterns (feature combinations) is to mine the patterns in the single-item sequence database.

Lemma 2 (*Ordered feature vector*):
For each example, the feature vector could be transformed into an ordered item (feature) list, i.e., $f_1<f_2<\ldots<f_{max}$ where f_{max} is the highest dimension of the example.

Proof. It is very easy to sort an unordered feature vector into the ordered list with conventional sorting algorithm.

Definition 3 (*Uniqueness of the features per example*):
Given the set of mined patterns, for any feature f_i, it is impossible to appear more than once in the same pattern.

Different from conventional sequential pattern mining method, in feature combination mining for SVM only contains a set of feature vectors each of which is independently treated. In other words, no compound features in the vector. If it exists, one can simply expand the compound features as another new feature.

By means of the above constraints, mining the frequent patterns can be reduced to mining the limited length of frequent patterns in the single-item database (set of ordered vectors). Furthermore, during each phase, we need only focus on finding the "frequent single features" to expand previous phase. More detail implementation issues can refer (Pei et al., 2004).

3.1 Speed-up Testing

To efficiently expand new features for the original feature vectors, we propose a new method to fast discovery patterns. Essentially, the PrefixSpan algorithm gradually expands one item from previous result which can be viewed as a tree growing. An example can be found in Figure 1.

Each node in Figure 1 is the associate feature of root. The whole patterns expanded by f_j can be represented as the path from root to each node. For example, pattern (f_j, f_k, f_m, f_r) can be found via traversing the tree starting from f_j. In this way, we can re-expand the original feature vector via visiting corresponding trees for each feature.

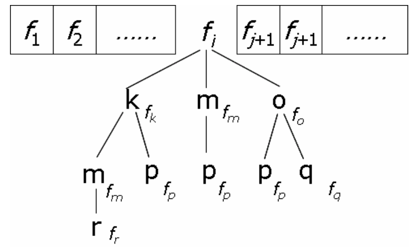

Figure 1: The tree representation of feature f_j

Table 1: Encoding frequent patterns with DFS array representation

Level	0	1	2	3	2	1	2	1	2	2
Label	Root	k	m	r	p	m	p	o	p	q
Item	f_j	f_k	f_m	f_r	f_p	f_m	f_p	f_o	f_p	f_q

However, traversing arrays is much more efficient than visiting trees. Therefore, we adopt the l^2-sequences encoding method based on the DFS (depth-first-search) sequence as (Wang et al., 2004) to represent the trees. An l^2-sequence does not only store the label information but also take the node level into account. Examples can be found in Table 1.

Theorem 4 (*Uniqueness of l^2-sequence*): Given trees T_1, and T_2, their l^2-sequences are identical if and only if T_1 and T_2 are isomorphic, i.e., there exists a one-to-one mapping for set of nodes, node labels, edges, and root nodes.

Proof. see theorem 1 in (Wang et al., 2004).

Definition 5 (*Ascend-descend relation*):
Given a node k of feature f_k in l^2-sequence, all of the descendant of k that rooted by k have the greater feature numbers than f_k.

Definition 6 (*Limited visiting space*):
Given the highest feature $f_{\max}$ of vector X, and f_k rooted l^2-sequence, if $f_{\max}<f_k$, then we can not find any pattern that prefix by f_k.

Both definitions 5 and 6 strictly follow lemma 2 that kept the ordered relations among features. For example, once node k could be found in X, it is unnecessary to visit its children. More specifically, to determine whether a frequent pattern is in X, we need to compare feature vector of X and l^2-sequence database. It is clearly that the time complexity of our method is $O(F_{\text{avg}}*N_{\text{avg}})$ where F_{avg} is the average number of frequent features per example, while the N_{avg} is the average length of l^2-sequence. In other words, our method does not dependent on the polynomial kernel degree.

4 Experiments

To evaluate our method, we examine the well-known shallow parsing task which is the task of CoNLL-2000[1]. We also adopted the released perl-evaluator to measure the recall/precision/f1 rates. The used feature consists of word, POS, orthographic, affix(2-4 prefix/suffix letters), and previous chunk tags in the two words context window size (the same as (Lee and Wu, 2007)). We limited the features should at least appear more than twice in the training set.

For the learning algorithm, we replicate the modified finite Newton SVM as learner which can be trained in linear time (Keerthi and DeCoste, 2005). We also compare our method with the standard linear and polynomial kernels with SVM^{light}[2].

4.1 Results

Table 2 lists the experimental results on the CoNLL-2000 shallow parsing task. Table 3 compares the testing speed of different feature expansion techniques, namely, array visiting (our method) and enumeration.

Table 2: Experimental results for CoNLL-2000 shallow parsing task

CoNLL-2000	F1	Mining Time	Training Time	Testing Time
Linear Kernel	93.15	N/A	0.53hr	2.57s
Polynomial(*d*=2)	94.19	N/A	11.52hr	3189.62s
Polynomial(*d*=3)	93.95	N/A	19.43hr	6539.75s
Our Method (*d*=2,sup=0.01)	93.71	<10s	0.68hr	6.54s
Our Method (*d*=3,sup=0.01)	93.46	<15s	0.79hr	9.95s

Table 3: Classification time performance of enumeration and array visiting techniques

CoNLL-2000	Array visiting		Enumeration	
	d=2	*d*=3	*d*=2	*d*=3
Testing time	6.54s	9.95s	4.79s	11.73s
Chunking speed (words/sec)	7244.19	4761.50	9890.81	4038.95

It is not surprising that the best performance was obtained by the classical polynomial kernel. But the limitation is that the slow in training and testing time costs. The most efficient method is linear kernel SVM but it does not as accurate as polynomial kernel. However, our method stands for both efficiency and accuracy in this experiment. In terms of training time, it slightly slower than the linear kernel, while it is 16.94 and ~450 times faster than polynomial kernel in training and testing. Besides, the pattern mining time is far smaller than SVM training.

As listed in Table 3, we can see that our method provide a more efficient solution to feature expansion when *d* is set more than two. Also it demonstrates that when *d* is small, the enumerate-based method is a better choice (see PKE in (Kudo and Matsumoto, 2004)).

5 Conclusion

This paper presents an approximate method for extending linear kernel SVM to analogy polynomial-like computing. The advantage of this method is that it does not require maintaining the cost of support vectors in training, while achieves satisfactory result. On the other hand, we also propose a new method for speeding up classification which is independent to the polynomial kernel degree. The experimental results showed that our method close to the performance of polynomial kernel SVM and better than the linear kernel. In terms of efficiency, our method did not only improve 16.94 times faster in training and 450 times in testing, but also faster than previous similar studies.

References

Chad Cumby and Dan Roth. 2003. Kernel methods for relational learning. International Conference on Machine Learning, pages 104-114.

Hideki Isozaki and Hideto Kazawa. 2002. *Efficient support vector classifiers for named entity recognition*. International Conference on Computational Linguistics, pages 1-7.

Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal and Mei-Chun Hsu. 2004. Mining Sequential Patterns by Pattern-Growth: The Prefix Span Approach. IEEE Trans. on Knowledge and Data Engineering, 16(11): 1424-1440.

Sathiya Keerthi and Dennis DeCoste. 2005. *A modified finite Newton method for fast solution of large scale linear SVMs*. Journal of Machine Learning Research. 6: 341-361.

Taku Kudo and Yuji Matsumoto. 2001. *Fast methods for kernel-based text analysis*. Annual Meeting of the Association for Computational Linguistics, pages 24-31.

Taku Kudo and Yuji Matsumoto. 2001. *Chunking with support vector machines*. Annual Meetings of the North American Chapter and the Association for the Computational Linguistics.

Yue-Shi Lee and Yu-Chieh Wu. 2007. *A Robust Multilingual Portable Phrase Chunking System*. Expert Systems with Applications, 33(3): 1-26.

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

Chen Wang, Mingsheng Hong, Jian Pei, Haofeng Zhou, Wei Wang and Baile Shi. 2004. *Efficient Pattern-Growth Methods for Frequent Tree Pattern Mining*. Pacific knowledge discovery in database (PAKDD).

[1] http://www.cnts.ua.ac.be/conll2000/chunking/
[2] http://svmlight.joachims.org/