A Sparse L_2-Regularized Support Vector Machines for Large-Scale Natural Language Learning

Yu-Chieh Wu[1], Yue-Shi Lee[3], Jie-Chi Yang[2], and Show-Jane Yen[3]

[1] Finance Department and School of Communication, Ming Chuan University,
No. 250 Zhong Shan N. Rd., Sec. 5, Taipei 111, Taiwan
bcbb@db.csie.ncu.edu.tw
[2] Graduate Institute of Network Learning Technology, National Central University,
No.300, Jhong-Da Rd., Jhongli City, Taoyuan County 32001, Taiwan, R.O.C.
yang@cl.ncu.edu.tw
[3] Department of Computer Science and Information Engineering, Ming Chuan University,
No.5, De-Ming Rd, Gweishan District, Taoyuan 333, Taiwan, R.O.C.
{leeys,sjyen}@mcu.edu.tw

Abstract. Linear support vector machines (SVMs) have become one of the most prominent classification algorithms for many natural language learning problems such as sequential labeling tasks. Even though the L_2-regularized SVMs yields slightly more superior accuracy than L_1-SVM, it produces too much near but non zero feature weights. In this paper, we present a cutting-weight algorithm to guide the optimization process of L_2-SVM into sparse solution. To verify the proposed method, we conduct the experiments with three well-known sequential labeling tasks and one dependency parsing task. The result shows that our method achieved at least 400% feature parameter reduction rates in comparison to the original L_2-SVM, with almost no change in accuracy and training times. In terms of run time efficiency, our method is faster than the original L_2-regularized SVMs at least 20% in all tasks.

Keywords: L_2-regularization, part-of-speech tagging, support vector machines, machine learning.

1 Introduction

Mining large-scale unlabeled data has received a great attention in recent years. Such methods bring important impacts on not only the system accuracy but also the scalability problems. Though one could train the learners with cloud computing servers, this is not the true way of handling large-scale natural language learning problems.

Kernel machines, such as support vector machines (SVMs) had been wildly used as learners in many natural language learning tasks [8]. Nevertheless, the training time of linear kernel SVM (with either L_1-norm [5] or L_2-norm [4, 7]) is now can be obtained in linear time. It is usually the case that the L_2-regularized SVMs achieves slightly better accuracy than the L_1-regularized SVMs. Unfortunately, L_2-SVM often generates dense models where most feature weights are small but non-zero. The situation is even more salient when training large-scale natural language learning tasks,

P.-J. Cheng et al. (Eds.): AIRS 2010, LNCS 6458, pp. 340–349, 2010.

like Chinese word segmentations. Maintaining such high-dense feature weights is not easy for common processors. In addition, the denser the model, the slower the testing it achieves.

In this paper, we present a cutting-weight algorithm for L_2-SVM for sequential labeling tasks. Our method iteratively guides the L_2-SVM optimization process toward sparse by disregarding a set of weak features. The classic feature selection approaches can also provide downstream input to the cutting-weight algorithm. To validate the effectiveness, we compare our method on three well-known benchmark corpora, namely, CoNLL-2000 syntactic chunking [14], SIGHAN Chinese word segmentation [13, 17, 18], and Chinese word dependency parsing [15]. The experimental result shows that our method is not only faster than the original L_2-SVM but also with no change in accuracy.

2 L_2-Regularized SVMs

Assume a binary classification problem with n labeled examples, $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ where $x_i \in \Re^d$ and $y_i \in \{+1, -1\}$. To obtain the linear classifier W (hypothesis) $y = W \cdot x + b$ the modified finite Newton L_2-SVM solves:

$$\min\{\frac{\lambda}{2} W^T W + \frac{1}{2}\sum_{i=1}^{n} \xi_i^2\} \text{ s.t. } \forall i,\ y_i(W \cdot x_i + b) \geq 1 - \xi_i \tag{1}$$

For simplicity, bias b can be easily modeled by adding an additional of routine constant to each x_i [11]. $\lambda (= 1/C)$ is the regularization parameter that controls the trade-off between margin size and training error.

Following [7], the function of L_2-SVM is a strictly convex, quadratic, and first order differentiable function and has a unique solution. The solution of L_2-SVM objective function is to minimize for:

$$\frac{\partial(2)}{\partial W} = \lambda \times W + X^T(X^T W - Y) \quad \text{and} \quad (\lambda \times I + X^T X)W = X^T Y \tag{2}$$

where X is the training data matrix (i.e. $X \in \Re^{n \times d}$) and Y is the column vector which contains the label of training examples. I is the identity matrix. Keerthi and DeCoste further introduce the modified finite Newton method to solve (2) with Conjugate Gradient (CG) [2] scheme refer to as CGLS.

3 Sparse L_2-Regularized SVMs Optimization

One good property of the L_2-SVM is that its objective function is strictly convex, first order derivable, and can be directly optimized in primal form. By following this line, we propose a cutting-weight algorithm to guide the *dense* weight vector into *sparse*. First, we decompose the weight vector into two different parts, weight vector W_1 (representative) and vector W_2 (non-representative), and $W = W_1 + W_2$. Thus the original objective function of (1) can be re-written as:

$$\min\{\frac{\lambda}{2}(W_1+W_2)^T(W_1+W_2)+\sum_{i=1}^{n}\xi_i^2\} \quad \text{s.t.} \ \forall i,\ y_i(W_1\cdot x_i+W_2\cdot x_i+b)\geq 1-\xi_i \tag{3}$$

The optimization problem is similar to solve:

$$(\lambda\times I+X^TX)(W_1+W_2)=X^TY \quad \text{and} \quad (W_1+W_2)=(\lambda\times I+X^TX)^{-1}X^TY \tag{4}$$

Clearly if W_1 (derived from W) is close enough to abstract W, say $W_1\cong W$, then solving (1) is almost the same as solving (3). In this way, by assuming that W_2 is far *limited* to a zero weight vector and can be further disregarded from the objective function, then the objective function becomes:

$$\min\{\frac{\lambda}{2}W_1^TW_1+\sum_{i=1}^{n}\xi_i^2\} \ \text{s.t.} \ \forall i,\quad y_i(W_1\cdot x_i+b)\geq 1-\xi_i \tag{5}$$

In other words, (5) is the exact solution of (1) iff W_1=W, otherwise it is the approximate solution of (3). This implies that the feature weights in W_2 directly affect the degree of approximation.

One important property of L_2-norm regularization is that it pushes a value less and less as it moves toward zero [3]. To find sparse model for L_2-SVM, we start to find W_2 which can be searched from 0. We present a cutting-weight method to construct W_2 in which W_1 can be easily obtained by W_1 =W-W_2. That is for each feature weight w_i in W_2, it satisfies:

$$\forall w_i\in W_2 \quad -\varepsilon\leq w_i\leq\varepsilon$$

Similarly for each feature weight w_i in W_1, it satisfies:

$$\forall w_i'\in W_1 \quad w_i'>\varepsilon \ \text{and} \ w_i'<-\varepsilon$$

ε is the cutting-weight parameter which controls the model sparsity and approximation of (5). It can also be interpreted as a threshold for distinguishing relevant (representative) or irrelevant (non-representative) feature values.

Obviously the parameter ε determines the trade-off between the model sparsity and how approximate W_1 reaches. By setting up this technique, we modified the original training algorithm. Fig. 1 outlines the presented sparse L_2-SVM optimization algorithm. The general optimization technique is the same as the modified finite Newton method for L_2-SVM [7]. The difference is that our method refines the weight vector by preserving the relevant features before checking the L_2-SVM optimality. Such technique could also be applied to the other gradient descent-based linear SVM optimization methods. For example, line 6-8 in Fig. 1 can be replaced by introducing the dual-coordinate descent algorithms [4]. Line 12 can be replaced by verifying the difference between maximum and minimum projected gradient (Property 3 of Theorem 2 in [4]).

Initialize W^0;
iter := 0;
While (! converged) {
    Set up (3) using
        $SV^k := \{i \mid y_i(W^k \cdot x_i) < 1\}$ **//Get the support vectors**
    Solve (3) with CG methods and obtain $\hat{W}^k$;
    Let $S^k := \hat{W}^k - W^k$, do a line search to find:
        $\alpha^k := \arg\min_{\alpha \geq 0} f(W^k + \alpha \times S^k)$;
    Update weight vector
        $W^{k+1} := W^k + \alpha^k \times S^k$;
        s.t. $\forall w_i \in W^{k+1}$ $w_i > \varepsilon$ and $w_i < -\varepsilon$ otherwise $w_i := 0$;
    if ($\nabla f(W^k) == 0$)
        stop;
    iter := *iter* +1;
}

Fig. 1. Sparse L_2-SVM optimization algorithm

3.1 Speed-Up Local Classifiers

For an m class multiclass problem, it needs to manage at least m weight vectors. Usually the larger the m and dimension are, the slower testing time is obtained. Usually the weight vectors are stored in an $m \times d$ matrix. However, it is not a good idea to represent the sparse model since most of the feature weight is zero. When m and d become large, the matrix will be unmanageable in practice.

To solve it, we further introduce the indexing idea from the Information Retrieval (IR) community to represent the sparse weight vectors. The basic concept is to *retrieve* the testing vector in the index file. Fig. 2 illustrates an example of the index file. For each dimension, we store the set of non-zero feature weight with as postings (linked list structures). The posting directly indicates the corresponding class id and its feature weight. Therefore, for each feature weight f_i in testing vector, by walking at the index file, we can easily retrieve a set of relevant categories that contains the same feature.

We do not discuss how to construct the index file and its complexity here since it can be done with existing IR approaches [12]. Therefore, the computational time complexity of multiclass SVM is O($m \times f_{\text{avg}}$) where f_{avg} is the average length of testing example. On the contrary, the testing time complexity of the sparse multiclass representation depends on the number of *relevant* items, i.e. O($m_{\text{avg}} \times f_{\text{avg}}$) where m_{avg} is the average number of relevant items per feature and $m_{avg} \leq m$.

The use of indexing file to SVM is not new. For example, [9] introduced a similar idea to manage large number of support vectors generated from the polynomial kernel SVM.

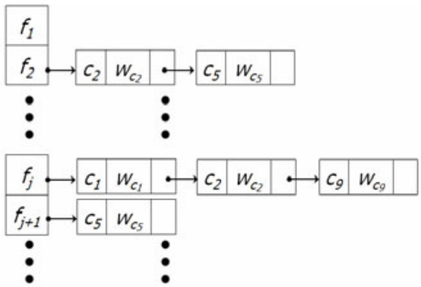

Fig. 2. An example of index file representation

1. F_{base} : the basic feature set;
2. F_{cand} : a set of feature candidates;
3. γ: a threshold parameter that is used to control the goodness of a newly added feature;
4. δ: a threshold parameter that is used to control the goodness of one iteration;
5. *Inner_acc* : the accuracy of the F_{base};
6. While (! converged) {
7.   *Best_Acc* := *Inner_acc*;
8.   for (i :=1~| F_{cand} |) {
9.   Derive one feature F_i from F_{cand};
10. $F := F_{base} + F_i$;
11.   Train SVM model by the feature set F;
12.   Evaluate the accuracy Acc_i with the trained model;
13.   if ($Acc_i > Inner\_acc+\gamma$) {
14. $F := F_{base} + F_i$;
15.   *Inner_acc* := Acc_i;
16.   }
17.   }
18.   if ((*Inner_acc* - *Best_acc*) < δ)
19.   converged := true;
20. }

Fig. 3. An algorithm for bottom-up feature search

3.2 Search the Optimal Feature Set

To further enhance the accuracy, we also design a simple feature selection algorithm which makes use of bottom-up search. Fig. 3 gives the designed feature search algorithm.

Before start search, a set of basic feature set is pre-defined and the set of candidates are stored in F_{cand}. The algorithm runs iteratively add one feature from F_{cand} and check whether the new feature leads to better accuracy or not. Line 13 of Fig. 3, we set a parameter γ which is used to control the quality of a new feature. If the resulted performance is greater than the threshold, the feature is added while the resulted accuracy will be kept. The algorithm will stop when the resulted accuracy no significant change. That means the for-loop (Line 8 to Line 17) should produce better results over the threshold δ.

4 Experiments

To evaluate our method, we ran experiments on three tasks: CoNLL-2000 syntactic chunking, three Chinese word segmentation tasks that derived from SIGHAN-3, and the Chinese Dependency parsing [16]. Table 1 shows the statistics of those datasets.

By following most literatures, we adopted the IOB2 with forward direction chunking scheme for CoNLL-2000 chunking task. As encouraged by [18], incorporating global information (AV feature) and extending the boundary information can improve the accuracy. In this paper, we include the AV features and make use of 6-tags[1] rather than I and B two tags for Chinese word segmentation task. The feature set and the settings of the Chinese dependency parsing were set the same as previous work [16].

To validate the proposed idea, we re-implemented SVM-MFN (L_2-norm with modified finite Newton method) [7] and LibLinear-L_2 [4]. To handle multiclass problems, we adopted the well-known one-versus-all (OVA) method. For the remaining experiments, we simply set the feature cut as 2 for all linear SVMs and $\varepsilon = 10^{-4}$ for our method. For a fair comparison, these methods received the same input training data, the same regularization parameter[2], and the same feature set. Table 2 lists the used feature set. All of the experiments were performed under the E6300 OC 3.2 GHz with 4GB RAM under the Server 2003 32bit environment.

Table 1. Dataset used in our experiments

Statistics		# of examples	# of sentences	# of categories
CoNLL-2000	Training	220K	8,935	11*2+1=23
	Testing	48K	2,011	
SIGHAN-3 UPUC	Training	500K	18,804	6
	Testing	154K	5,117	
SIGHAN-3 MSRA	Training	2.16M	46,364	6
	Testing	63K	4,365	
SIGHAN-3 CityU	Training	1.6M	57,274	6
	Testing	220K	7,511	
Chinese word dependency parsing	Training	740K	16,909	12*2+2=26
	Testing	82K	1,878	

4.1 Results

Table 3 lists overall experimental results of the selected three tasks. The table contains accuracy ($F_{(\beta)}$ rate), training time (sec.) and testing time (sec.), the difference between objective goals, and the number of iterations the learner takes. The model size implies the memory requirement in run-time. The larger number the model is, the larger memory usage requires. As shown in Table 3, our method produced much smaller model size without changing in accuracy. In terms of training, our method truncates near-zero features and makes tight model. Intuitively, the calculation cost of the objective function (eq. (3)) is reduced.

[1] Similar to IOB2 tagging scheme, the 6-tags: S/B/BI/I/IE/E indicate the Single/Begin/SecondBegin/Interior/BeforeEnd/ End of a chunk.

[2] C=0.1/1 for the CoNLL-2000 chunking task and SIGHAN-3 datasets.

Table 2. Feature templates used in our experiments

Feature type	CoNLL-2000	SIGHAN-3 Chinese word segmentation
Unigram	w_{-2}~w_{+2}	w_{-2}~w_{+2}
Bigram	$(w_{-2},w_{-1}),(w_{-1},w_0),(w_0,w_{+1}),$ $(w_{+1},w_{+2}),(w_{+1},w_{-1})$	$(w_{-1},w_0),(w_0,w_{+1}),$ (w_{+1},w_{-1})
POS	p_{-2}~p_{+2}	
POS bigram	$(p_{-2},p_{-1}),(p_{-1},p_0),(p_0,p_{+1}),$ $(p_{+1},p_{+2}),(p_{+1},p_{-1})$	
POS trigram	$(p_{-2},p_{-1},p_0),(p_{-1},p_0,p_{+1}),$ $(p_{-3},p_{-2},p_{-1}),(p_0,p_{+1},p_{+2}),\ (p_{+1},p_{+2},p_{+3})$	
Word+POS bigram	$(w_{-1},p_0),(w_{-2},p_{-1})\ (w_0,p_{+1}),\ (w_{+1},p_{+2})$	
Other features	2~4 suffix letters 2~4 prefix letters Orthographic feature	AV feature [17] of 2~6 grams
History	*Prev.chunk*$_{-1}$, *Prev.chunk*$_{-2}$	

We also run another experiment to compare with the "final cutting" method. This method simply eliminates the weights after the conventional SVM training with the same ε. Table 4 lists the comparison result of the final-cut. In this experiment, we scale down the epsilon to 0.05 since there will be no accuracy difference when epsilon is larger than 0.005. As shown in Table 4, clearly, our method yields more superior results than f-cut in terms of model size and accuracy. In this experiment, we fail to train SVM-MFN with (ε=0.05) in short time. It takes more than 20 hours to train while resulting slightly worse result than f-cut (93.96 v.s. 93.90).

Note that we should carefully select suitable value for parameter ε. The input feature value is either 0 or 1 in many natural language learning tasks. There is no need to use large epsilon. Observing by the above experiments, we found that $\varepsilon=10^{-3}$~10^{-15} usually yield much smaller model size, better training/testing time cost.

Table 3. The overall performance results of the selected tasks

Tasks			# of Iterations	Model Size (non-zero entries)	Sum of the objective goals	Training Time (sec.)	Testing Time (sec.)	Accuracy ($F_{(\beta)}$)
CoNLL-2000	SVM-MFN	ε=0	290	4.20M	-2945.39	576.16	8.42	94.11
		ε=0.0001	305	0.36M	-2945.37	555.98	3.30	94.11
	LibLinear	ε=0	1134	0.40M	-2945.33	76.58	3.54	94.11
		ε=0.0001	1130	0.36M	-2945.33	77.34	3.29	94.11
SIGHAN3-CityU	SVM-MFN	ε=0	103	9.52M	-119927.00	13947.08	4.91	97.41
		ε=0.0001	103	2.21M	-119927.00	10713.27	3.70	97.41
	LibLinear	ε=0	424	2.32M	-119924.00	595.59	3.73	97.41
		ε=0.0001	424	2.22M	-119924.00	585.48	3.69	97.41
Chinese word dependency parsing	SVM-MFN	ε=0	332	51.36M	-6785.00	9455.59	7.95	81.71
		ε=0.0001	333	4.78M	-6810.19	9233.88	6.30	81.73
	LibLinear	ε=0	470	49.51M	-6810.06	591.33	7.23	81.71
		ε=0.0001	491	4.79M	-6810.06	591.50	6.31	81.71

Table 4. Performance comparison between simple "final-cut" and our approach

CoNLL-2000		Model size (non-zero weights)	Acc.
LibLinear	Our method	68,027	94.03
(ε=**0.05**)	F-cut	156,386	93.94
LibLinear	Our method	286,892	94.10
(ε=0.005)	F-cut	313,762	94.10
SVM-MFN	Our method	305,597	**94.14**
(ε=0.005)	F-cut	313,706	94.10

Table 5 lists the detail comparisons among the selected SVM optimization techniques in the CoNLL-2000 and SIGHAN-3 datasets. In this experiment, we still set (ε=0.0001) as default parameter value for our method.

In summary, our cutting-weight algorithm shows better training and testing time performance for most dataset with excepted for Liblinear(Multi). In terms of accuracy, the L_2-regularized groups such as SVM-MFN and Liblinear(L_2) yield more superior $F_{(\beta)}$ rate than the others. In terms of training time cost, the SVM-light performs significantly worse than the others and the Liblinear (multi) won the best training time efficiency. Even Liblinear (multi) shows great training time performance, it does not as accurate as the L_2-regularized groups. Also, its testing time is not fast.

By applying the proposed cutting-weight algorithm to SVM-MFN and Liblinear (L_2), both learners obtains better training and testing time costs than the original while resulting almost no change in accuracy. As compared to SVM-MFN, our cutting-weight optimization technique usually produces better training and much reduced testing time costs while keeping the same accuracy.

In this paper, we do not successfully conduct all experiments for SVM-light since the training time is not human-tolerable. For example, it costs more than one week to train one class with the MSRA and CityU dataset. That means it takes at least 6 weeks to train one Chinese word segmentor.

Table 5. Experimental results of different SVM optimization techniques

Tasks		**This paper** SVM-MFN	**This paper** Liblinear (L_2)	SVM-MFN	Liblinear (L_2)	Liblinear (L_1)	SVM-perf	SVM-light	Liblinear (Mutli)
CoNLL-2000	Tr.Time	556	77	576	77	73	336	2881	40
	Te.Time	3.30	3.29	8.42	3.54	3.30	4.52	3.97	3.42
	$F_{(\beta)}$	94.11	94.11	94.11	94.11	93.96	93.98	93.85	92.26
Chinese word dependency parsing	Tr.Time	6229	245	6859	246	312	4941	97538	312
	Te.Time	6.28	6.33	6.56	6.34	6.75	7.67	6.66	6.66
	$F_{(\beta)}$	81.72	81.71	81.72	81.71	81.66	80.42	79.90	81.83
SIGHAN-3 UPUC	Tr.Time	1247	138	1246	140	150	816	64657	93
	Te.Time	2.36	2.37	3.45	2.72	2.36	2.41	2.42	2.39
	$F_{(\beta)}$	93.95	93.95	93.95	93.95	93.99	93.94	93.91	93.89
SIGHAN-3 MSRA	Tr.Time	7398	481	8440	465	591	4323	Out-of-time-bound*	440
	Te.Time	1.63	1.63	2.66	1.95	2.06	2.05		2.08
	$F_{(\beta)}$	95.93	95.93	95.93	95.93	95.95	95.81		95.43
SIGHAN-3 CityU	Tr.Time	10713	585	13947	596	608	5119		408
	Te.Time	3.70	3.69	4.91	3.73	3.70	3.70		3.69
	$F_{(\beta)}$	97.41	97.41	97.41	97.41	97.30	97.33		97.34

* In our experiments, the overall training time of SVM-light is more than 1 week to train one category.

5 Conclusion

We presented a cutting-weight algorithm for L_2-SVM (online demo can be found here[3]) to save the model space for handling large-scale data. The experimental results show that our method achieves better accuracy on three well-known datasets, namely, CoNLL-2000 syntactic chunking, SIGHAN-3 Chinese word segmentation, and Chinese word dependency parsing. In addition, the results also show that there is no change in accuracy between our method and the original L_2-SVM, while it greatly reduced the model size and also reaches slightly faster training time and at least 20% improvement in runtime efficiency.

References

1. Collins, M.: Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In: Empirical Methods in Natural Language Processing, pp. 1–8 (2002)
2. Frommer, A., Maaß, P.: Fast CG-based methods for Tikhonov-Phillips regularization. Journal of Scientific Computing 20(5), 1831–1850 (1999)
3. Gao, J., Andrew, G., Johnson, M., Toutanova, K.: A comparative study of parameter estimation methods for statistical natural language processing. In: 45th Annual Meeting of the Association of Computational Linguistics, pp. 824–831 (2007)
4. Hsieh, C.J., Chang, K.W., Lin, C.J., Keerthi, S., Sundararajan, S.: A dual coordinate descent method for large-scale linear SVM. In: 15th International Conference on Machine Learning, pp. 408–415 (2008)
5. Joachims, T.: Training linear SVMs in linear time. In: ACM Conference on Knowledge Discovery and Data Mining, pp. 217–226 (2006)
6. Keerthi, S., Sundararajan, S., Chang, K.W., Hsieh, C.J., Lin, C.J.: A sequential dual method for large scale multi-class linear SVMs. In: ACM Conference on Knowledge Discovery and Data Mining, pp. 408–416 (2008)
7. Keerthi, S., DeCoste, D.: A modified finite Newton method for fast solution of large scale linear SVMs. Journal of Machine Learning Research 6, 341–361 (2005)
8. Kudo, T. and Matsumoto, Y.: Chunking with support vector machines. In: North American Chapter of the Association for Computational Linguistics on Language Technologies, pp. 192-199 (2001)
9. Kudo, T., Matsumoto, Y.: Fast methods for kernel-based text analysis. In: The 41st Annual Meeting of the Association of Computational Linguistics, pp. 24–31 (2003)
10. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: 8th International Conference on Machine Learning, pp. 282–289 (2001)
11. Mangasarian, O.L., Musicant, D.: Lagrangian support vector machines. Journal of Machine Learning Research 1, 161–177 (2001)
12. Manning, C., Raghavan, P., Schutze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
13. Ng, H.T., Low, J.K.: Chinese part-of-speech tagging: one-at-a-time or all-at-once? Word-based or character-based? In: Empirical Methods in Natural Language Processing, pp. 277–284 (2004)

[3] http://140.115.112.118/bcbb/CMM-CoNLL/sparseSVM.htm

14. Tjong Kim Sang, E.F., Buchholz, S.: Introduction to the CoNLL 2000 shared task: chunking. In: 4th Conference on Computational Natural Language Learning, pp. 127–132 (2000)
15. Wu, Y.C., Yang, J.C., Lee, Y.S.: An approximate approach for training polynomial kernel SVMs in linear time. In: The 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, pp. 65–68 (2007)
16. Wu, Y.C., Lee, Y.S., Yang, J.C.: Robust and efficient Chinese word dependency analysis with linear kernel support vector machines. In: proceedings of 22nd International Conference on Computational Linguistics Poster, pp. 135–138 (2008)
17. Zhang, Y., Clark, S.: Chinese segmentation with a word-based perceptron algorithm. In: 45th Annual Meeting of the Association of Computational Linguistics, pp. 840–847 (2007)
18. Zhao, H., Kit, C.: Incorporating global information into supervised learning for Chinese word segmentation. In: 10th Conference of the Pacific Association for Computational Linguistics, pp. 66–74 (2007)