Discriminative Classifiers for Deterministic Dependency Parsing

Johan Hall
Växjö University
jni@msi.vxu.se

Joakim Nivre
Växjö University and
Uppsala University
nivre@msi.vxu.se

Jens Nilsson
Växjö University
jha@msi.vxu.se

Abstract

Deterministic parsing guided by treebank-induced classifiers has emerged as a simple and efficient alternative to more complex models for data-driven parsing. We present a systematic comparison of memory-based learning (MBL) and support vector machines (SVM) for inducing classifiers for deterministic dependency parsing, using data from Chinese, English and Swedish, together with a variety of different feature models. The comparison shows that SVM gives higher accuracy for richly articulated feature models across all languages, albeit with considerably longer training times. The results also confirm that classifier-based deterministic parsing can achieve parsing accuracy very close to the best results reported for more complex parsing models.

1 Introduction

Mainstream approaches in statistical parsing are based on nondeterministic parsing techniques, usually employing some kind of dynamic programming, in combination with generative probabilistic models that provide an n-best ranking of the set of candidate analyses derived by the parser (Collins, 1997; Collins, 1999; Charniak, 2000). These parsers can be enhanced by using a discriminative model, which reranks the analyses output by the parser (Johnson et al., 1999; Collins and Duffy, 2005; Charniak and Johnson, 2005). Alternatively, discriminative models can be used to search the complete space of possible parses (Taskar et al., 2004; McDonald et al., 2005).

A radically different approach is to perform disambiguation deterministically, using a greedy parsing algorithm that approximates a globally optimal solution by making a sequence of locally optimal choices, guided by a classifier trained on gold standard derivations from a treebank. This methodology has emerged as an alternative to more complex models, especially in dependency-based parsing. It was first used for unlabeled dependency parsing by Kudo and Matsumoto (2002) (for Japanese) and Yamada and Matsumoto (2003) (for English). It was extended to labeled dependency parsing by Nivre et al. (2004) (for Swedish) and Nivre and Scholz (2004) (for English). More recently, it has been applied with good results to lexicalized phrase structure parsing by Sagae and Lavie (2005).

The machine learning methods used to induce classifiers for deterministic parsing are dominated by two approaches. Support vector machines (SVM), which combine the maximum margin strategy introduced by Vapnik (1995) with the use of kernel functions to map the original feature space to a higher-dimensional space, have been used by Kudo and Matsumoto (2002), Yamada and Matsumoto (2003), and Sagae and Lavie (2005), among others. Memory-based learning (MBL), which is based on the idea that learning is the simple storage of experiences in memory and that solving a new problem is achieved by reusing solutions from similar previously solved problems (Daelemans and Van den Bosch, 2005), has been used primarily by Nivre et al. (2004), Nivre and Scholz (2004), and Sagae and Lavie (2005).

Comparative studies of learning algorithms are relatively rare. Cheng et al. (2005b) report that SVM outperforms MaxEnt models in Chinese dependency parsing, using the algorithms of Yamada and Matsumoto (2003) and Nivre (2003), while Sagae and Lavie (2005) find that SVM gives better performance than MBL in a constituency-based shift-reduce parser for English.

In this paper, we present a detailed comparison of SVM and MBL for dependency parsing using the deterministic algorithm of Nivre (2003). The comparison is based on data from three different languages – Chinese, English, and Swedish – and on five different feature models of varying complexity, with a separate optimization of learning algorithm parameters for each combination of language and feature model. The central importance of feature selection and parameter optimization in machine learning research has been shown very clearly in recent research (Daelemans and Hoste, 2002; Daelemans et al., 2003).

The rest of the paper is structured as follows. Section 2 presents the parsing framework, including the deterministic parsing algorithm and the history-based feature models. Section 3 discusses the two learning algorithms used in the experiments, and section 4 describes the experimental setup, including data sets, feature models, learning algorithm parameters, and evaluation metrics. Experimental results are presented and discussed in section 5, and conclusions in section 6.

2 Inductive Dependency Parsing

The system we use for the experiments uses no grammar but relies completely on inductive learning from treebank data. The methodology is based on three essential components:

1. Deterministic parsing algorithms for building dependency graphs (Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre, 2003)
2. History-based models for predicting the next parser action (Black et al., 1992; Magerman, 1995; Ratnaparkhi, 1997; Collins, 1999)
3. Discriminative learning to map histories to parser actions (Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre et al., 2004)

In this section we will define dependency graphs, describe the parsing algorithm used in the experiments and finally explain the extraction of features for the history-based models.

2.1 Dependency Graphs

A dependency graph is a labeled directed graph, the nodes of which are indices corresponding to the tokens of a sentence. Formally:

Definition 1 Given a set R of dependency types (arc labels), a *dependency graph* for a sentence $x = (w_1, \ldots, w_n)$ is a labeled directed graph $G = (V, E, L)$, where:

1. $V = \mathbf{Z}_{n+1}$
2. $E \subseteq V \times V$
3. $L : E \rightarrow R$

The set V of *nodes* (or *vertices*) is the set $\mathbf{Z}_{n+1} = \{0, 1, 2, \ldots, n\}$ $(n \in \mathbf{Z}^+)$, i.e., the set of non-negative integers up to and including n. This means that every token index i of the sentence is a node ($1 \leq i \leq n$) and that there is a special node 0, which does not correspond to any token of the sentence and which will always be a root of the dependency graph (normally the only root). We use V^+ to denote the set of nodes corresponding to tokens (i.e., $V^+ = V - \{0\}$), and we use the term *token node* for members of V^+.

The set E of *arcs* (or *edges*) is a set of ordered pairs (i, j), where i and j are nodes. Since arcs are used to represent dependency relations, we will say that i is the *head* and j is the *dependent* of the arc (i, j). As usual, we will use the notation $i \rightarrow j$ to mean that there is an arc connecting i and j (i.e., $(i, j) \in E$) and we will use the notation $i \rightarrow^* j$ for the reflexive and transitive closure of the arc relation E (i.e., $i \rightarrow^* j$ if and only if $i = j$ or there is a path of arcs connecting i to j).

The function L assigns a dependency type (arc label) $r \in R$ to every arc $e \in E$.

Definition 2 A dependency graph G is *well-formed* if and only if:

1. The node 0 is a root.
2. Every node has in-degree at most 1.
3. G is connected.[1]
4. G is acyclic.
5. G is projective.[2]

Conditions 1–4, which are more or less standard in dependency parsing, together entail that the graph is a rooted tree. The condition of projectivity, by contrast, is somewhat controversial, since the analysis of certain linguistic constructions appears to

[1]To be more exact, we require G to be *weakly connected*, which entails that the corresponding undirected graph is connected, whereas a *strongly connected* graph has a *directed* path between any pair of nodes.

[2]An arc (i, j) is projective iff there is a path from i to every node k such that $i < j < k$ or $i > j > k$. A graph G is projective if all its arcs are projective.

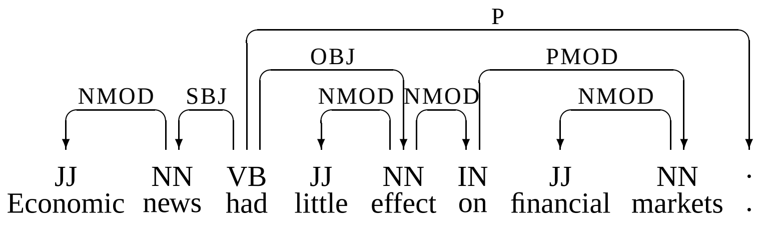

Figure 1: Dependency graph for an English sentence from the WSJ section of the Penn Treebank

require non-projective dependency arcs. For the purpose of this paper, however, this assumption is unproblematic, given that all the treebanks used in the experiments are restricted to projective dependency graphs.

Figure 1 shows a well-formed dependency graph for an English sentence, where each word of the sentence is tagged with its part-of-speech and each arc labeled with a dependency type.

2.2 Parsing Algorithm

We begin by defining parser configurations and the abstract data structures needed for the definition of history-based feature models.

Definition 3 Given a set $R = \{r_0, r_1, \ldots r_m\}$ of dependency types and a sentence $x = (w_1, \ldots, w_n)$, a *parser configuration* for x is a quadruple $c = (\sigma, \tau, h, d)$, where:

1. σ is a stack of tokens nodes.
2. τ is a sequence of token nodes.
3. $h : V_x^+ \rightarrow V$ is a function from token nodes to nodes.
4. $d : V_x^+ \rightarrow R$ is a function from token nodes to dependency types.
5. For every token node $i \in V_x^+$, $h(i) = 0$ if and only if $d(i) = r_0$.

The idea is that the sequence τ represents the remaining input tokens in a left-to-right pass over the input sentence x; the stack σ contains partially processed nodes that are still candidates for dependency arcs, either as heads or dependents; and the functions h and d represent a (dynamically defined) dependency graph for the input sentence x. We refer to the token node on top of the stack as the *top token* and the first token node of the input sequence as the *next token*.

When parsing a sentence $x = (w_1, \ldots, w_n)$, the parser is initialized to a configuration $c_0 = (\epsilon, (1, \ldots, n), h_0, d_0)$ with an empty stack, with all the token nodes in the input sequence, and with all token nodes attached to the special root node 0 with a special dependency type r_0. The parser terminates in any configuration $c_m = (\sigma, \epsilon, h, d)$ where the input sequence is empty, which happens after one left-to-right pass over the input.

There are four possible parser transitions, two of which are parameterized for a dependency type $r \in R$.

1. LEFT-ARC(r) makes the top token i a (left) dependent of the next token j with dependency type r, i.e., $j \xrightarrow{r} i$, and immediately pops the stack.
2. RIGHT-ARC(r) makes the next token j a (right) dependent of the top token i with dependency type r, i.e., $i \xrightarrow{r} j$, and immediately pushes j onto the stack.
3. REDUCE pops the stack.
4. SHIFT pushes the next token i onto the stack.

The choice between different transitions is nondeterministic in the general case and is resolved by a classifier induced from a treebank, using features extracted from the parser configuration.

2.3 Feature Models

The task of the classifier is to predict the next transition given the current parser configuration, where the configuration is represented by a feature vector $\Phi_{(1,p)} = (\phi_1, \ldots, \phi_p)$. Each feature ϕ_i is a function of the current configuration, defined in terms of an *address function* a_{ϕ_i}, which identifies a specific token in the current parser configuration, and an *attribute function* f_{ϕ_i}, which picks out a specific attribute of the token.

Definition 4 Let $c = (\sigma, \tau, h, d)$ be the current parser configuration.

1. For every i ($i \geq 0$), σ_i and τ_i are address functions identifying the ith token of σ and τ, respectively (with indexing starting at 0).

2. If α is an address function, then $h(\alpha)$, $l(\alpha)$, and $r(\alpha)$ are address functions, identifying the head (h), the leftmost child (l), and the rightmost child (r), of the token identified by α (according to the function h).

3. If α is an address function, then $p(\alpha)$, $w(\alpha)$ and $d(\alpha)$ are feature functions, identifying the part-of-speech (p), word form (w) and dependency type (d) of the token identified by α. We call p, w and d attribute functions.

A feature model is defined by specifying a vector of feature functions. In section 4.2 we will define the feature models used in the experiments.

3 Learning Algorithms

The learning problem for inductive dependency parsing, defined in the preceding section, is a pure classification problem, where the input instances are parser configurations, represented by feature vectors, and the output classes are parser transitions. In this section, we introduce the two machine learning methods used to solve this problem in the experiments.

3.1 MBL

MBL is a lazy learning method, based on the idea that learning is the simple storage of experiences in memory and that solving a new problem is achieved by reusing solutions from similar previously solved problems (Daelemans and Van den Bosch, 2005). In essence, this is a k nearest neighbor approach to classification, although a variety of sophisticated techniques, including different distance metrics and feature weighting schemes can be used to improve classification accuracy.

For the experiments reported in this paper we use the TiMBL software package for memory-based learning and classification (Daelemans and Van den Bosch, 2005), which directly handles multi-valued symbolic features. Based on results from previous optimization experiments (Nivre et al., 2004), we use the modified value difference metric (MVDM) to determine distances between instances, and distance-weighted class voting for determining the class of a new instance. The parameters varied during experiments are the number k of nearest neighbors and the frequency threshold l below which MVDM is replaced by the simple Overlap metric.

3.2 SVM

SVM in its simplest form is a binary classifier that tries to separate positive and negative cases in training data by a hyperplane using a linear kernel function. The goal is to find the hyperplane that separates the training data into two classes with the largest margin. By using other kernel functions, such as polynomial or radial basis function (RBF), feature vectors are mapped into a higher dimensional space (Vapnik, 1998; Kudo and Matsumoto, 2001). Multi-class classification with n classes can be handled by the one-versus-all method, with n classifiers that each separate one class from the rest, or the one-versus-one method, with $n(n-1)/2$ classifiers, one for each pair of classes (Vural and Dy, 2004). SVM requires all features to be numerical, which means that symbolic features have to be converted, normally by introducing one binary feature for each value of the symbolic feature.

For the experiments reported in this paper we use the LIBSVM library (Wu et al., 2004; Chang and Lin, 2005) with the polynomial kernel $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$, where d, γ and r are kernel parameters. Other parameters that are varied in experiments are the penalty parameter C, which defines the tradeoff between training error and the magnitude of the margin, and the termination criterion ϵ, which determines the tolerance of training errors.

We adopt the standard method for converting symbolic features to numerical features by binarization, and we use the one-versus-one strategy for multi-class classification. However, to reduce training times, we divide the training data into smaller sets, according to the part-of-speech of the next token in the current parser configuration, and train one set of classifiers for each smaller set. Similar techniques have previously been used by Yamada and Matsumoto (2003), among others, without significant loss of accuracy. In order to avoid too small training sets, we pool together all parts-of-speech that have a frequency below a certain threshold t (set to 1000 in all the experiments).

4 Experimental Setup

In this section, we describe the experimental setup, including data sets, feature models, parameter optimization, and evaluation metrics. Experimental results are presented in section 5.

4.1 Data Sets

The data set used for Swedish comes from Talbanken (Einarsson, 1976), which contains both written and spoken Swedish. In the experiments, the professional prose section is used, consisting of about 100k words taken from newspapers, textbooks and information brochures. The data has been manually annotated with a combination of constituent structure, dependency structure, and topological fields (Teleman, 1974). This annotation has been converted to dependency graphs and the original fine-grained classification of grammatical functions has been reduced to 17 dependency types. We use a pseudo-randomized data split, dividing the data into 10 sections by allocating sentence i to section $i \bmod 10$. Sections 1–9 are used for 9-fold cross-validation during development and section 0 for final evaluation.

The English data are from the Wall Street Journal section of the Penn Treebank II (Marcus et al., 1994). We use sections 2–21 for training, section 0 for development, and section 23 for the final evaluation. The head percolation table of Yamada and Matsumoto (2003) has been used to convert constituent structures to dependency graphs, and a variation of the scheme employed by Collins (1999) has been used to construct arc labels that can be mapped to a set of 12 dependency types.

The Chinese data are taken from the Penn Chinese Treebank (CTB) version 5.1 (Xue et al., 2005), consisting of about 500k words mostly from Xinhua newswire, Sinorama news magazine and Hong Kong News. CTB is annotated with a combination of constituent structure and grammatical functions in the Penn Treebank style, and has been converted to dependency graphs using essentially the same method as for the English data, although with a different head percolation table and mapping scheme. We use the same kind of pseudo-randomized data split as for Swedish, but we use section 9 as the development test set (training on section 1–8) and section 0 as the final test set (training on section 1–9).

A standard HMM part-of-speech tagger with suffix smoothing has been used to tag the test data with an accuracy of 96.5% for English and 95.1% for Swedish. For the Chinese experiments we have used the original (gold standard) tags from the treebank, to facilitate comparison with results previously reported in the literature.

Feature	Φ_1	Φ_2	Φ_3	Φ_4	Φ_5
$p(\sigma_0)$	+	+	+	+	+
$p(\tau_0)$	+	+	+	+	+
$p(\tau_1)$	+	+	+	+	+
$p(\tau_2)$				+	+
$p(\tau_3)$				+	+
$p(\sigma_1)$					+
$d(\sigma_0)$		+	+	+	+
$d(l(\sigma_0))$		+	+	+	+
$d(r(\sigma_0))$		+	+	+	+
$d(l(\tau_0))$		+	+	+	+
$w(\sigma_0)$			+	+	+
$w(\tau_0)$			+	+	+
$w(\tau_1)$					+
$w(h(\sigma_0))$					+

Table 1: Feature models

4.2 Feature Models

Table 1 describes the five feature models Φ_1–Φ_5 used in the experiments, with features specified in column 1 using the functional notation defined in section 2.3. Thus, $p(\sigma_0)$ refers to the part-of-speech of the top token, while $d(l(\tau_0))$ picks out the dependency type of the leftmost child of the next token. It is worth noting that models Φ_1–Φ_2 are unlexicalized, since they do not contain any features of the form $w(\alpha)$, while models Φ_3–Φ_5 are all lexicalized to different degrees.

4.3 Optimization

As already noted, optimization of learning algorithm parameters is a prerequisite for meaningful comparison of different algorithms, although an exhaustive search of the parameter space is usually impossible in practice.

For MBL we have used the modified value difference metric (MVDM) and class voting weighted by inverse distance (ID) in all experiments, and performed a grid search for the optimal values of the number k of nearest neighbors and the frequency threshold l for switching from MVDM to the simple Overlap metric (cf. section 3.1). The best values are different for different combinations of data sets and models but are generally found in the range 3–10 for k and in the range 1–8 for l.

The polynomial kernel of degree 2 has been used for all the SVM experiments, but the kernel parameters γ and r have been optimized together with the penalty parameter C and the termination

FM	LM	Swedish				English				Chinese			
		AS		EM		AS		EM		AS		EM	
		U	L	U	L	U	L	U	L	U	L	U	L
Φ_1	MBL	75.3	68.7	16.0	11.4	*76.5	73.7	9.8	7.7	66.4	63.6	14.3	12.1
	SVM	75.4	68.9	16.3	12.1	76.4	73.6	9.8	7.7	66.4	63.6	14.2	12.1
Φ_2	MBL	81.9	74.4	31.4	19.8	81.2	78.2	19.8	14.9	73.0	70.7	22.6	18.8
	SVM	*83.1	*76.3	*34.3	*24.0	81.3	78.3	19.4	14.9	*73.2	*71.0	22.1	18.6
Φ_3	MBL	85.9	81.4	37.9	28.9	85.5	83.7	26.5	23.7	77.9	76.3	26.3	23.4
	SVM	86.2	*82.6	38.7	*32.5	*86.4	*84.8	*28.5	*25.9	*79.7	*78.3	*30.1	*25.9
Φ_4	MBL	86.1	82.1	37.6	30.1	87.0	85.2	29.8	26.0	79.4	77.7	28.0	24.7
	SVM	86.0	82.2	37.9	31.2	*88.4	*86.8	*33.2	*30.3	*81.7	*80.1	*31.0	*27.0
Φ_5	MBL	86.6	82.3	39.9	29.9	88.0	86.2	32.8	28.4	81.1	79.2	30.2	25.9
	SVM	86.9	*83.2	40.7	*33.7	*89.4	*87.9	*36.4	*33.1	*84.3	*82.7	*34.5	*30.5

Table 2: Parsing accuracy; FM: feature model; LM: learning method; AS: attachment score, EM: exact match; U: unlabeled, L: labeled

criterion e. The intervals for the parameters are: γ: 0.16–0.40; r: 0–0.6; C: 0.5–1.0; e: 0.1–1.0.

4.4 Evaluation Metrics

The evaluation metrics used for parsing accuracy are the *unlabeled attachment score* AS_U, which is the proportion of tokens that are assigned the correct head (regardless of dependency type), and the *labeled attachment score* AS_L, which is the proportion of tokens that are assigned the correct head and the correct dependency type. We also consider the *unlabeled exact match* EM_U, which is the proportion of sentences that are assigned a completely correct dependency graph without considering dependency type labels, and the *labeled exact match* EM_L, which also takes dependency type labels into account. Attachment scores are presented as mean scores per token, and punctuation tokens are excluded from all counts. For all experiments we have performed a McNemar test of significance at $\alpha = 0.01$ for differences between the two learning methods. We also compare learning and parsing times, as measured on an AMD 64-bit processor running Linux.

5 Results and Discussion

Table 2 shows the parsing accuracy for the combination of three languages (Swedish, English and Chinese), two learning methods (MBL and SVM) and five feature models (Φ_1–Φ_5), with algorithm parameters optimized as described in section 4.3. For each combination, we measure the *attachment score* (AS) and the *exact match* (EM). A significant improvement for one learning method over the other is marked by an asterisk (*).

Independently of language and learning method, the most complex feature model Φ_5 gives the highest accuracy across all metrics. Not surprisingly, the lowest accuracy is obtained with the simplest feature model Φ_1. By and large, more complex feature models give higher accuracy, with one exception for Swedish and the feature models Φ_3 and Φ_4. It is significant in this context that the Swedish data set is the smallest of the three (about 20% of the Chinese data set and about 10% of the English one).

If we compare MBL and SVM, we see that SVM outperforms MBL for the three most complex models Φ_3, Φ_4 and Φ_5, both for English and Chinese. The results for Swedish are less clear, although the labeled accuracy for Φ_3 and Φ_5 are significantly better. For the Φ_1 model there is no significant improvement using SVM. In fact, the small differences found in the AS_U scores are to the advantage of MBL. By contrast, there is a large gap between MBL and SVM for the model Φ_5 and the languages Chinese and English. For Swedish, the differences are much smaller (except for the EM_L score), which may be due to the smaller size of the Swedish data set in combination with the technique of dividing the training data for SVM (cf. section 3.2).

Another important factor when comparing two learning methods is the efficiency in terms of time. Table 3 reports learning and parsing time for the three languages and the five feature models. The learning time correlates very well with the complexity of the feature model and MBL, being a lazy learning method, is much faster than SVM. For the unlexicalized feature models Φ_1 and Φ_2, the parsing time is also considerably lower for MBL, especially for the large data sets (English and Chinese). But as model complexity grows, especially with the addition of lexical features, SVM gradually gains an advantage over MBL with respect to parsing time. This is especially striking for Swedish,

Method	Model	Swedish		English		Chinese	
		LT	PT	LT	PT	LT	PT
Φ_1	MBL	1 s	2 s	16 s	26 s	7 s	8 s
	SVM	40 s	14 s	1.5 h	14 min	1.5 h	17 min
Φ_2	MBL	3 s	5 s	35 s	32 s	13 s	14 s
	SVM	40 s	13 s	1 h	11 min	1.5 h	15 min
Φ_3	MBL	6 s	1 min	1.5 min	9.5 min	46 s	10 min
	SVM	1 min	15 s	1 h	9 min	2 h	16 min
Φ_4	MBL	8 s	2 min	1.5 min	9 min	45 s	12 min
	SVM	2 min	18 s	2 h	12 min	2.5 h	14 min
Φ_5	MBL	10 s	7 min	3 min	41 min	1.5 min	46 min
	SVM	2 min	25 s	1.5 h	10 min	6 h	24 min

Table 3: Time efficiency; LT: learning time, PT: parsing time

where the training data set is considerably smaller than for the other languages.

Compared to the state of the art in dependency parsing, the unlabeled attachment scores obtained for Swedish with model Φ_5, for both MBL and SVM, are about 1 percentage point higher than the results reported for MBL by Nivre et al. (2004). For the English data, the result for SVM with model Φ_5 is about 3 percentage points below the results obtained with the parser of Charniak (2000) and reported by Yamada and Matsumoto (2003). For Chinese, finally, the accuracy for SVM with model Φ_5 is about one percentage point lower than the best reported results, achieved with a deterministic classifier-based approach using SVM and preprocessing to detect root nodes (Cheng et al., 2005a), although these results are not based on exactly the same dependency conversion and data split as ours.

6 Conclusion

We have performed an empirical comparison of MBL (TiMBL) and SVM (LIBSVM) as learning methods for classifier-based deterministic dependency parsing, using data from three languages and feature models of varying complexity. The evaluation shows that SVM gives higher parsing accuracy and comparable or better parsing efficiency for complex, lexicalized feature models across all languages, whereas MBL is superior with respect to training efficiency, even if training data is divided into smaller sets for SVM. The best accuracy obtained for SVM is close to the state of the art for all languages involved.

Acknowledgements

The work presented in this paper was partially supported by the Swedish Research Council. We are grateful to Hiroyasu Yamada and Yuan Ding for sharing their head percolation tables for English and Chinese, respectively, and to three anonymous reviewers for helpful comments and suggestions.

References

Ezra Black, Frederick Jelinek, John D. Lafferty, David M. Magerman, Robert L. Mercer, and Salim Roukos. 1992. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, pages 31–37.

Chih-Chung Chang and Chih-Jen Lin. 2005. LIBSVM: A library for support vector machines.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 173–180.

Eugene Charniak. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139.

Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2005a. Chinese deterministic dependency analyzer: Examining effects of global features and root node finder. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, pages 17–24.

Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. 2005b. Machine learning-based dependency analyzer for Chinese. In *Proceedings of the International Conference on Chinese Computing (ICCC)*.

Michael Collins and Nigel Duffy. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31:25–70.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 16–23.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Walter Daelemans and Veronique Hoste. 2002. Evaluation of machine learning methods for natural language processing tasks. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC)*, pages 755–760.

Walter Daelemans and Antal Van den Bosch. 2005. *Memory-Based Language Processing*. Cambridge University Press.

Walter Daelemans, Veronique Hoste, Fien De Meulder, and Bart Naudts. 2003. Combined optimization of feature selection and algorithm parameter interaction in machine learning of language. In *Proceedings of the 14th European Conference on Machine Learning (ECML)*, pages 84–95.

Jan Einarsson. 1976. *Talbankens skriftspråkskonkordans*. Lund University, Department of Scandinavian Languages.

Mark Johnson, Stuart Geman, Steven Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 535–541.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Taku Kudo and Yuji Matsumoto. 2002. Japanese dependency analysis using cascaded chunking. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL)*, pages 63–69.

David M. Magerman. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 276–283.

Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate-argument structure. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 114–119.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 91–98.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 64–70.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–10.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pages 125–132.

Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. 2004. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.

Ulf Teleman. 1974. *Manual för grammatisk beskrivning av talad och skriven svenska*. Studentlitteratur.

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

Vladimir Vapnik. 1998. *Statistical Learning Theory*. John Wiley and Sons, New York.

Volkan Vural and Jennifer G. Dy. 2004. A hierarchical method for multi-class support vector machines. *ACM International Conference Proceeding Series*, 69:105–113.

Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng. 2004. Probability estimates for multi-class classification by pairwise coupling. *Journal of Machine Learning Research*, 5:975–1005.

Nianwen Xue, Fei Xia, Fu-Dong Chiou, and Martha Palmer. 2005. The Penn Chinese Treebank: Phrase structure annotation of a large corpus. *Natural Language Engineering*, 11(2):207–238.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.