School of Mathematics and Systems Engineering

**Reports from MSI** - Rapporter från MSI

# CoNLL-X Shared Task: Multi-lingual Dependency Parsing

Johan Hall and Jens Nilsson
{jha, jni}@msi.vxu.se

Jun
2006

MSI
Växjö University
**SE-351 95 VÄXJÖ**

Report 06060
ISSN 1650-2647
ISRN VXU/MSI/DA/E/--06060/--SE

## 1 Introduction

The goal of this report is to summarize our experiments and present the final result of our participation in the CoNLL-X Shared Task 2006.[1] The topic of this year's shared task was multi-lingual dependency parsing, which is the sixth shared task in the CoNLL history.

The organizers have prepared 13 existing dependency treebanks so that they all comply to the same markup format. The training and test data for the languages differ in size, granularity and quality, but they have tried to even out differences in the markup format. No additional information is allowed to be used besides the provided training data, forcing the parser to be fully automatic and data-driven. Ideally, the same parser should be trainable for all languages, possibly by adjusting parameters.

The main goal is to assign labeled dependency structure for all languages on held out test data, approximately 5 000 tokens for each language. The main metric for comparison of the different parsers of the participants is therefore labeled attachment score, i.e., the proportion of tokens that are assigned both the correct head and the correct dependency relation.

Our approach have been to use the existing trainable dependency parser MaltParser (Nivre et al., 2006b), adapt it to the definition of the CoNLL markup format, and develop it further in order to improve parsing accuracy. MaltParser is a configurable system that implements more than one machine learner and parsing algorithm. It is also possible to define different feature models. This raises the question: exactly what is meant by "the same parser should be trainable for all languages". How much flexibility is allowed while maintaining the concept "the same parser". In addition, to what extent are we allowed to do different kinds of preprocessing and postprocessing of the data, which is not part of the actual parser system? In particular, how do we deal with non-projectivity using the projective parsing algorithm of MaltParser (section 2.3)? The somewhat fuzzy rule leaves room for different interpretations, which is something that we had to consider during the development phase.

[1] The web page http://nextens.uvt.nl/∼conll/ has detailed information the CoNLL-X Shared Task 2006.

In section 2, we will briefly present the framework of inductive dependency parsing (Nivre, 2006), which is realized in MaltParser. More specifically, we will define the dependency graphs, discuss the parsing algorithms and the machine learning algorithms that it uses. The markup format is discussed in section 3, together with a brief overview of the 13 treebanks. Our course of action and our results during development phase are the topics of section 4. Before concluding in section 6, section 5 summarizes the outcome the final evaluation.

Since the authors are not the only people involved in our shared task group, we want to acknowledge especially three people. Thanks to Gülşen Eryiğit and Svetoslav Marinov for being the main responsible persons for the Turkish and Bulgarian experiments, respectively, and to Joakim Nivre for being the coordinator and for doing several experiments when our workload did not permit us to do it ourselves. Together with these people we have reported our participation in a paper in the CoNLL 2006 proceedings (Nivre et al., 2006a), with a short error analysis of Swedish and Turkish.

## 2 Inductive Dependency Parsing

Our approach to dependency parsing, realized in the MaltParser system, is based on the framework of inductive dependency parsing. It was characterized by Nivre (2006), which is based on three essential elements:

1. Deterministic parsing algorithms for building dependency graphs (Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre, 2003)

2. History-based feature models for predicting the next transition from one parser configuration to another (Black et al., 1992; Magerman, 1995; Ratnaparkhi, 1997; Collins, 1999)

3. Discriminative learning methods to map histories to transitions (Veenstra and Daelemans, 2000; Kudo and Matsumoto, 2002; Yamada and Matsumoto, 2003; Nivre et al., 2004)

Given that we have a treebank for a specific language, our approach is to induce a parser model at learning time and use this parser model to parse

sentences. However, since it is problematic to use the dependency graph directly to construct such a model, we instead use a deterministic parsing algorithm to map a dependency graph to a transition sequence such that this transition sequence uniquely determines the dependency graph. The transition system in itself is normally nondeterministic and we therefore need a mechanism that resolves this nondeterminism. We use a discriminative learning method to construct a classifier. Moreover, we use history-based feature models to extract vectors of feature-value pairs from the current parser state as training material for the classifier.

### 2.1 Dependency Graphs

Given a sentence $x_i$ in a text $T = (x_1, \ldots, x_n)$, the goal of dependency parsing is to create a dependency graph consisting of lexical nodes linked by binary relations called *dependencies*. We define dependency graphs as follows (Nivre, 2006):

**Definition 1** *Given a sentence $x = (w_1, \ldots, w_n)$, where $w_i$ is a token, and a set $R = \{r_0, r_1, \ldots r_m\}$ of dependency types, a* dependency graph *for a sentence $x$ is a labeled directed graph $G = (V, E, L)$, where:*

1. $V = \mathbf{Z}_{n+1} = \{0, 1, 2, \ldots, n\}$
2. $E \subseteq V \times V$
3. $L : E \rightarrow R$

A dependency graph consists of a set $V$ of nodes, where a node is a non-negative integer (including $n$). Every positive node has a corresponding token in the sentence $x$ and we will use the term *token node* for these nodes (i.e., the token $w_i$ corresponds to the token node $i$). In addition, there is a special root node 0, which is the root of the dependency graph and has no corresponding token in the sentence $x$.

An arc $(i, j) \in E$ connects two nodes $i$ and $j$ in the graph and represents a dependency relation where $i$ is the head and $j$ is the dependent. Finally, the function $L$ labels every arc $(i, j)$ with a dependency type $r \in R$.

Figure 1 shows an example of a dependency graph, which uses binary relations between words in a Swedish sentence and each relation is labeled with a dependency type.

### 2.2 Parsing Algorithms

We use Nivre's parsing algorithm to build a labeled dependency graph in one left-to-right pass over the input, using a stack to store partially processed tokens (Nivre, 2006). The algorithm uses a parser configuration consisting of a stack $\sigma$ of partially processed token nodes and a list $\tau$ of remaining input token nodes. The algorithm comes in two versions: *arc-eager* and *arc-standard*. The first version uses a transition system with four transitions that defines the transition from one parser configuration to another (where top is the token on top of the stack $\sigma$ and next is the next token of the list remaining input token $\tau$):

- SHIFT: Push next onto the stack.
- REDUCE: Pop the stack.
- RIGHT-ARC(r): Add an arc labeled r from top to next; push next onto the stack.
- LEFT-ARC(r): Add an arc labeled r from next to top; pop the stack.

The transition SHIFT shifts (pushes) the next input token onto the stack. This is done when the head of the next word is positioned to the right of the next word. The transition REDUCE reduces (pops) the token on top of the stack. It is important to ensure that the parser does not pop the top token if it has not been assigned a head, since it will otherwise be left unattached.

The RIGHT-ARC transition adds an arc from the token on top of the stack to the next input token and pushes the next input token onto the stack. Finally, the transition LEFT-ARC adds an arc from the next input token to the token on top of the stack. In order to prohibit words from more than one head, it presupposes that the top token has no head.

The arc-standard version uses three transition SHIFT, RIGHT-ARC and LEFT-ARC:[2]

- SHIFT: Push next onto the stack.
- RIGHT-ARC(r): Add an arc labeled r from top to next; move back the topmost token on the stack to the list of remaining input tokens so

[2]The transitions SHIFT and LEFT-ARC are applied in exactly the same way as for the arc-eager version.

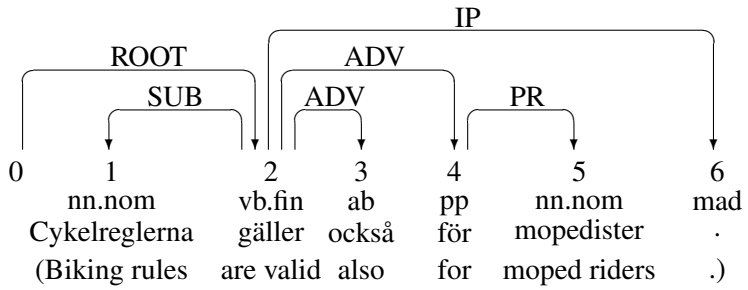


Figure 1: Dependency graph for Swedish sentence, converted from Talbanken

that this token becomes the new next input token.

- LEFT-ARC(r): Add an arc labeled r from next to top; pop the stack.

Both versions of Nivre's parsing algorithm were not able to assign dependency labels to roots before the CoNLL-X shared task. To overcome this problem, the implementation of the algorithm was adapted to handle labeled roots. This variant of the algorithm starts by pushing an *artificial root* token onto the stack. Tokens having the root as its head is attached to the artificial root in a RIGHT-ARC(r) action, which means that they can be assigned any label.

### 2.3 Projective Parsing

The parsing algorithm described in section 2.2 is projective, that is, it can only produce projective dependency structures. This is an obvious problem since most of the treebanks contain non-projectivity (see section 3). Neglecting these relations when adopting projective parsing will result in a decreased accuracy of at least the same magnitude as the amount of non-projective relations. Dutch, which contains 5.4% non-projective relations (see section 3.1), will then have at least 5.4%-points lower accuracy. A sentence containing at least one non-projective relation will not even in theory be assigned the correct dependency structure, implying that a completely correct analysis is not even a possible asymptotic goal. However, parsing algorithms that can produce non-projectivity tend to be less efficient as well as less robust and accurate.

The parser conforms to the following definition of projectivity: an arc $(i, k)$ is *projective* iff, for every node $j$ occurring between the nodes $i$ and $k$ (i.e., $i < j < k$ or $i > j > k$), there is a path from $i$ to $j$. A graph is projective iff all its arcs are projective.

We have in this study applied pseudo-projective parsing in order to recover non-projective constructions, which is inspired by Kahane et al. (1998). This is a form of graph transformation technique both on the non-projective training data and on the projective output of the parser. The procedure is divided into three steps:

1. The training data is preprocessed (projectivized) so that it conforms to the definition of projectivity. The dependency relations of some of the arcs involved in this graph transformation are augmented with additional information to facilitate the inverse transformation in step 3.

2. The parser is trained using the projectivized training data, and the test data is parsed.

3. An inverse transformation (deprojectivization) is applied on the parser output using the additional information added in step 1 to recover the non-projective relations.

This technique has previously been shown to work well for Czech (Nivre and Nilsson, 2005), using three different encoding schemes to encode the graph transformations: HEAD, PATH and HEAD+PATH. All of these have been tested and evaluated here.

### 2.4 History-Based Models

The parsing algorithm is deterministic in the sense that it always uses one transition sequence $S = (t_1, \ldots, t_m)$ to derive the dependency graph, but the transition systems for both arc-eager and arc-standard are nondeterministic. Hence, several transitions are applicable for the same parser configuration. At learning time the parsing algorithm uses an *oracle* to get the correct transition (Kay, 2000), which derives the transition from syntactically annotated sentences in a treebank. A transition $t_i$ is dependent on all previously made transitions $(t_1, \ldots, t_{i-1})$ and all available information about these transitions, called the *history*. The history $H_i = (t_1, \ldots, t_{i-1})$ corresponds to some partially built structure and we also include static properties that are kept constant during the parsing of a sentence, such as, word form and part-of-speech of a token.

The basic idea is thus to train a classifier that approximates an oracle given that a treebank is available. We will call the approximated oracle a *guide* (Boullier, 2003), because the guide does not guarantee that the transition is correct. The history $H_i = (t_1, \ldots, t_{i-1})$ contains complete information about all previous decisions. All this information is intractable for training a classifier. Instead we can use history-based feature models for predicting the next transition (Black et al., 1992). To make it tractable, the history $H_i$ is replaced by a feature vector defined by a feature model $\Phi = (\phi_1, \ldots, \phi_p)$, where each feature $\phi_i$ is a function that identifies some significant property of the history $H_i$ and/or the input string $x$. To simplify notation we will write $\Phi(H_i, x)$ to denote the application of the feature vector $(\phi_1, \ldots, \phi_p)$ to $H_i$ and $x$, i.e., $\Phi(H_i, x) = (\phi_1(H_i, x), \ldots, \phi_p(H_i, x))$.

The feature extraction uses the feature model $\Phi = (\phi_1, \ldots, \phi_p)$, where each feature $\phi_i$ is a function, defined in terms of two simpler functions: an *address function* $a_{\phi_i}$, which identifies a specific token in a given parser configuration, an *attribute function* $f_{\phi_i}$, which picks out a specific attribute of the token.

1. For every $i$, $i \geq 0$, $\sigma[i]$, $\tau[i]$ are address functions identifying the $i+1$th token from the top of the stack $\sigma$ and the start of the input list $\tau$, respectively. (Hence, $\sigma[0]$ is the top of the stack and $\tau[0]$ is the next input token)
2. If $\alpha$ is an address function, then $l(\alpha)$ and $r(\alpha)$ are address function, identifying the left and right string neighbors, respectively, of the token identified by $\alpha$.
3. If $\alpha$ is an address function, then $h(\alpha)$, $lc(\alpha)$, $rc(\alpha)$, $ls(\alpha)$ and $rs(\alpha)$ are address functions, identifying the head ($h$), the leftmost child ($lc$), the rightmost child ($rc$), the next left sibling ($ls$) and the next right sibling ($rs$), respectively, of the token identified by $\alpha$ (according to the partially built dependency graph $G$).
4. If $\alpha$ is an address function, then $f(\alpha)$ are feature functions, identifying a particular attribute of the token identified by $\alpha$. The part-of-speech ($p$) of fine-grained tagset, part-of-speech ($c$) of the coarse-grained tagset, word form ($w$), lemma ($lem$), morphological features ($fea$) and dependency type ($d$) are example of attributes which can be identified (where the dependency type, if any, is given by the partially built dependency graph $G$). We call $p$, $c$, $w$, $lem$, $fea$ and $d$ attribute functions.

At learning time the parser derives the correct transition by using an oracle function applied to gold standard treebank. For each transition it provides the learner with a training instance $\Phi((H_i, x), t_i)$, where $\Phi(H_i, x)$ is a current vector of feature values and $t_i$ is the correct transition. A set of training instances $I$ is then used by the learner to induce a parser model, by using a supervised learning method.

At parsing time the parser uses the parser model, as a guide, to predict the next transition and now the vector of feature values $\Phi(H_i, x)$ is the input and the transition $t_i$ is the output of the guide.

### 2.5 Learning Methods

The learning problem is to induce a classifier from a set of training instances $I$ relative to a specific feature model $\Phi$ by using a learning algorithm. MaltParser comes with two different learning algorithms: memory-based learning (MBL) and support vector machines (SVM).

MBL is based on two fundamental principles: learning is storing experiences in memory, and solving a new problem is achieved by reusing solutions

from previously solved problems that are similar to the new problem. The idea during training for MBL is to collect the values of different features from the training data together with the correct class (Daelemans and Van den Bosch, 2005). MBL generalizes by applying a similarity metric without abstracting or eliminating low-frequency events.

MaltParser implements an interface to a software package called TiMBL (Tilburg Memory-Based Learner), which is used for memory-based learning and classification (Daelemans and Van den Bosch, 2005). TiMBL can directly handle multi-valued symbolic features and has a wide variety of parameters, which can be tuned for a specific learning task. We use the same settings for the TiMBL learner that Nivre (2006) used in the final evaluation for Swedish, where the number of nearest neighbors is set to $k = 5$ and the Modified Value Difference Metric (MVDM) is used to compute distances between feature values. Inverse distance-weighted class voting (ID) is used to determine the majority class. MVDM is used down to $l = 3$; below that threshold the simple Overlap metric is used.

SVM is based on the idea that two linearly separable classes, the positive and negative samples in the training data, can be separated by a hyperplane with the largest margin (Kudo and Matsumoto, 2001; Vapnik, 1998). SVM can be extended to solve problems that are not linearly separable. One solution is to allow some misclassifications by introducing a penalty parameter $C$, which defines the trade off between the training error and the magnitude of the margin. Another solution is to map the feature vector to a higher dimensional space by the function, which makes it possible to carry out non-linear classification. There exist several functions for doing this mapping. We use the polynomial kernel function $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, \gamma > 0$, where $\gamma$, $r$ and $d$ denote different kernel parameters (Hsu et al., 2004). It is also possible to determine the tolerance of training errors by tuning the termination criterion $\epsilon$.

SVM is in its basic form a binary classifier, but our learning problem has to deal with more than two classes. To make SVM handle multi-classification we can use the method one-against-all. Given that we have $n$ classes, the method trains $n$ classifiers to separate each class from the rest.

MaltParser implements an interface to a software library called LIBSVM (Chang and Lin, 2001) to handle multi-class SVM classification. The interface has a mechanism to divide the training data into smaller sets, according to a feature $\phi_s$ in the feature model, for example $p(\tau[0])$, and train one classifier for each smaller set (Hall, 2006). Similar techniques have previously been used by Yamada and Matsumoto (2003), among others, without significant loss of accuracy. There is a parameter $T$ specifying the frequency threshold $t$ that determines if a certain feature value should be pooled together with other values that occur less than $t$ times in the training data.

## 3 The Data

All 13 treebanks (Hajič et al., 2004; Simov et al., 2005; Simov and Osenova, 2003; Chen et al., 2003; Böhmová et al., 2003; Kromann, 2003; van der Beek et al., 2002; Brants et al., 2002; Kawata and Bartels, 2000; Afonso et al., 2002; Džeroski et al., 2006; Civit Torruella and Martí Antonín, 2002; Nilsson et al., 2005; Oflazer et al., 2003; Atalay et al., 2003) are dependency treebanks and comply to the data format specified by the organizers of the shared task, where some were originally encoded as dependency trees whereas others were converted to dependency structure. All treebanks are tokenized and segmented into sentences (or utterances), where each token is a list of features, some obligatory and others optional (see section 3.2). In addition to this, there are several other properties that are worth keeping in mind when evaluating the experiments. Some of them are listed below:

- The sizes of the treebanks and the average sentence lengths.
- The proportion of non-scoring tokens.
- The proportion of non-projectivity in the treebanks.
- All gold-standard features specified in the data format is not available for all treebanks. An optional missing feature in the data file for a treebank is marked with a special symbol.
- The number of distinct values for the gold-standard features.

| | #T | #S | #T/#S | %NST | %NPR | %NPS | IR |
|---|---|---|---|---|---|---|---|
| Arabic | 54 | 1.5 | 37.2 | 8.8 | 0.4 | 11.2 | Yes |
| Bulgarian | 190 | 12.8 | 14.8 | 14.4 | 0.4 | 5.4 | No |
| Chinese | 337 | 57 | 5.9 | 0.8 | 0.0 | 0.0 | No |
| Czech | 1249 | 72.7 | 17.2 | 14.9 | 1.9 | 23.2 | Yes |
| Danish | 94 | 5.2 | 18.2 | 13.9 | 1.0 | 15.6 | No |
| Dutch | 195 | 13.3 | 14.6 | 11.3 | 5.4 | 36.4 | No |
| German | 700 | 39.2 | 17.8 | 11.5 | 2.3 | 27.8 | No |
| Japanese | 151 | 17 | 8.9 | 11.6 | 1.1 | 5.3 | No |
| Portuguese | 207 | 9.1 | 22.8 | 14.2 | 1.3 | 18.9 | Yes |
| Slovene | 29 | 1.5 | 18.7 | 17.3 | 1.9 | 22.2 | Yes |
| Spanish | 89 | 3.3 | 27 | 12.6 | 0.1 | 1.7 | No |
| Swedish | 191 | 11 | 17.3 | 11.0 | 1.0 | 9.8 | No |
| Turkish | 58 | 5 | 11.5 | 33.1 | 1.5 | 11.6 | No |

Table 1: Treebank information; #T = number of tokens * 1000, #S = number of sentences * 1000, #T/#S = tokens per sentence, %NST = % of non-scoring tokens, %NPR = % of non-projective relations, %NPS = % of non-projective sentences, IR = has informative root labels

### 3.1 Treebank Overview

An overview of first three points above is shown in table 1. The first and second column display that the amount of data varies greatly, where the largest (Czech) contains more than 40 times as many tokens as the smallest (Slovene). To facilitate evaluation, one way to group them is according to size: Czech and German are large (above 500k words), Arabic, Danish, Slovene, Spanish and Turkish are small (below 100k), and the rest are medium.

The treebanks can also be grouped according to sentence length (#T/#S), since this tends to correlate with accuracy (long sentences are harder to parse). Arabic[3], Portuguese, and Spanish have long sentences ($>$20), Chinese and Japanese[4] have short sentences ($<$10), and the rest have sentences with medium length.

The shared task organizers have taken the decision to exclude certain tokens, mostly punctuation. A non-scoring token is a token where all its characters have the Unicode category property "Punctuation". Chinese, for example, has very few non-scoring token (%NST), because most of them were excluded in the conversion process. In Turkish, the non-last inflection groups are treated as individual tokens but categorized as "Punctuation", making as many as one third of the tokens non-scoring.

Another thing that varies a lot is the proportion of non-projectivity. The Dutch treebank contains the most non-projectivity, both with respect to the proportion of non-projective relations (%NPR) and the proportion of sentences containing non-projective relations (%NPS). The Bulgarian, Chinese and Spanish treebanks have no or very little non-projectivity, which partly is the result of the lack of discontinuity in the original treebanks, and partly in the conversion to the dependency based data format of the CoNLL shared task.

Four of the treebanks have so called informative root labels, i.e. tokens that are not dependent of another token (HEAD=0) have an "ordinary" dependency label. In addition to Portuguese, the treebanks Arabic, Czech and Slovene that are based on functional generative description (Sgall et al., 1986) have this property. This poses a problem for MaltParser, which in its basic configuration is unable to assign dependency labels to tokens without a head (other than a predefined and fix root label).

### 3.2 Feature Overview

As mentioned above some token features are obligatory. These are ID, FORM, CPOSTAG, POSTAG,

[3]In many cases the unit in the Arabic treebank is not a sentence but a paragraph

[4]The Japanese treebank consists of transcribed dialogs, in which some sentences are very short, e.g. just "Yes"

HEAD and DEPREL:

- ID: Token counter, starting at 1 for each new sentence.
- FORM: Word form or punctuation symbol.
- CPOSTAG: Coarse-grained part-of-speech tag, where the tagset is language-specific. It is mapped from POSTAG.
- POSTAG: Fine-grained part-of-speech tag, where the tagset depends on the language, or identical to the coarse-grained part-of-speech tag if not available.
- HEAD: Head of the current token, which is either a value of ID or zero ('0'). Depending on the original treebank annotation, there may be multiple tokens with ID$= 0$.
- DEPREL: Dependency relation to the HEAD. The set of dependency types depends on the particular language.

In addition to these, a treebank has zero or more of the these optional features:

- LEMMA: Lemma or stem (depending on the particular data set) of the word form.
- FEATS: Unordered set of syntactic and/or morphological features (e.g. for some treebanks temporal and case information).
- PHEAD: Projective head of current token.
- PDEPREL: Dependency relation to the PHEAD.

We did not use PHEAD and PDEPREL at all, since we deal with non-projectivity using pseudo-projective parsing instead.

Table 2 shows an overview of the information available in the treebanks. It clearly reveals that there are differences between the encoded information in the treebanks, which has various reasons. For instance, Chinese and Dutch have a high number of distinct POSTAG values. The former because POSTAG also encodes sub-categorization information for verbs and some semantic information for conjunctions and nouns, and the latter because the part-of-speech for multi-word units is the concatenation of the part-of-speech of its parts.

| | L | #C | #P | #F | #D |
|---|---|---|---|---|---|
| Arabic | Yes | 14 | 19 | 19 | 27 |
| Bulgarian | No | 11 | 53 | 50 | 19 |
| Chinese | No | 22 | 303 | - | 134 |
| Czech | Yes | 12 | 63 | 61 | 84 |
| Danish | No | 10 | 24 | 47 | 53 |
| Dutch | Yes | 13 | 302 | 81 | 26 |
| German | No | 52 | 52 | - | 46 |
| Japanese | No | 20 | 77 | 4 | 8 |
| Portuguese | Yes | 15 | 21 | 146 | 55 |
| Slovene | Yes | 11 | 28 | 51 | 26 |
| Spanish | Yes | 15 | 38 | 33 | 21 |
| Swedish | No | 37 | 37 | - | 63 |
| Turkish | Yes | 14 | 30 | 82 | 26 |

Table 2: Available info; L = has LEMMA, #C = number of different CPOSTAG values, #P = number of different POSTAG values, #F = number of parts (separated by '|') in FEATS, #D = number of different DEPREL values

Also the number of dependency types differs a lot, ranging from 8 (for Japanese) to 134 (for Chinese). This is of importance, especially for the evaluation since the main measurement is labeled attachment score, but also for the parser itself since the number of possible transitions depends on the number of dependency types. Furthermore, although it does not affect the parsing result, it is worth noting that FORM and LEMMA for Arabic contains both the original word and its English transcription.

## 4 Experiments

We had less than two months to prepare models for all 13 languages. We organized the work in different steps. Every week we had a project meeting where we together decided which tracks were the best or if we had to reconsider the plan. This section contains a summarization of all the experiments, but several details are left out.[5]

First we did a preliminary study to decide which machine learning methods and parsing algorithms that should be used. Also we needed to decide some settings for the parsing algorithm. We ran several experiments with feature models and with learner

[5]The web page http://www.vxu.se/msi/users/jha/conllx/ contains a more detailed summary of all experiments.

parameters that have shown to give good results before. For this step it was important to find the best settings over all the languages, because the contest in principle only allowed one learning method and parsing algorithm. These preliminary experiments are discussed in sections 4.1 and 4.2.

Secondly, we tried out several strategies to transform non-projective structures into projective structures and some of the strategies involved the inverse transformation. We adopted the same principle as above, that is, we wanted to use one strategy for all languages. These projectivization experiments are presented in section 4.3.

When we had decided which machine learning method, parsing algorithm and its settings, and projectivization strategy to use, we continued with the time consuming process of feature and learner parameter optimization. There exist infinite number of combinations of feature models and learner parameters. Therefore, given the limited amount of time, we used different optimization strategies for the different languages. These optimization strategies are discussed in sections 4.4 and 4.5.

10-fold cross validation was used for most of the small treebanks and for the other languages we used 80% of the data as training data and 20% of the data as development test set.

Finally, after thousands of experiments we had derived a reasonably optimized feature model and learner parameters for each language. Probably, if we have had more time, we could have derived even better feature models and learner parameters.

We did what we called a dry run for all languages to ensure that everything had been properly done. For this dry run we used 90% for training and 10% for testing. Section 4.6 discusses the outcome of the dry run.

### 4.1 Machine Learning Method

The outcome of the comparison between the two machine learning methods that are incorporated into the parser, MBL and SVM, is presented in this section. The method that yields the highest labeled attachment score on average will be used for all languages in the contest. Previous studies have indicated that SVM outperforms MBL, both in similar experiments conducted with MaltParser for other treebanks (Hall et al., 2006), as well as for constituency-based parsing (Sagae and Lavie, 2005).

As stated by Daelemans and Hoste (2002), unless an optimization of the feature model and the machine learner parameters is performed simultaneous, a completely fair comparison between the machine learners is hard to achieve. They also say that simply applying the default machine learning parameters in the comparison may be misleading. Doing the former has not been possible due to the limited time constraints. We have used a feature model and machine learner parameters that have worked well for other treebanks in previous studies. This is likely to be a better approach than using the default parameters and very simple set of features, even though SVM tends to have more features in its optimal model, and more fickle parameters, than MBL.

The features in the feature model applied for all languages are marked with * in table 7, consisting of 14 features, with 4 lexical, 6 part-of-speech and 4 dependency type features. Only Nivre's arc-eager algorithm has been used, but the oracle parameter varies between the languages (see section 4.2).

Despite the fact that no exhaustive evaluation was performed, the figures in table 3 clearly confirm the results from previous studies. SVM results, with no exceptions, in a higher labeled attachment score, although the differences between MBL and SVM for the different languages vary. The last column shows the error reduction. SVM decreases the proportion of errors the most for Portuguese and the least for Arabic, compared to MBL. The general tendency seems to be that SVM results in a lower error reduction for small treebanks (such as Arabic, Slovene and Turkish) than MBL, compared to the larger ones.

On average, SVM decreases the amount of errors with just above 10%, which made the choice between MBL and SVM quite simple. We used SVM throughout the rest of the experiments and in the final parsers for the contest.

### 4.2 Parsing Algorithm

In the experiments above, the arc-eager version of Nivre's algorithm was used, since this has previously resulted in a higher accuracy for several languages than arc-standard version. Exceptions exist, such as the Chinese Treebank, converted

| | MBL | SVM | %ER |
|---|---|---|---|
| Arabic | 61.7 | 63.0 | 3.4 |
| Bulgarian | 82.7 | 84.6 | 11.0 |
| Chinese | 82.9 | 85.4 | 14.6 |
| Czech | 69.0 | 72.5 | 11.3 |
| Danish | 80.2 | 82.6 | 12.1 |
| Dutch | 73.5 | 76.8 | 12.5 |
| German | 82.0 | 84.1 | 11.7 |
| Japanese | 89.6 | 90.6 | 9.6 |
| Portuguese | 77.5 | 84.3 | 30.2 |
| Slovene | 62.2 | 64.4 | 5.8 |
| Spanish | 74.3 | 77.1 | 10.9 |
| Swedish | 80.6 | 82.8 | 11.3 |
| Turkish | 63.6 | 66.2 | 7.1 |
| Average | 75.4 | 78.0 | 10.6 |

Table 3: Comparison of machine learners; %ER = % error reduction; * = split training data

| | no AR | AR |
|---|---|---|
| Arabic | 59.04 | 62.97 |
| Bulgarian | 84.55 | 84.48 |
| Chinese | 82.98 | 82.76 |
| Czech | 68.06 | 72.37 |
| Danish | 82.26 | 82.26 |
| Dutch | 76.04 | 75.86 |
| German | 84.39 | 84.44 |
| Japanese | 90.61 | 90.61 |
| Portuguese | 80.44 | 84.34 |
| Slovene | 62.12 | 64.45 |
| Spanish | 76.84 | 77.09 |
| Swedish | 82.78 | 82.47 |
| Turkish | 75.8 | 75.8 |

Table 4: Comparison of the use of artificial root or not; AR = Artificial root; labeled AS

from phrase structure to dependency structure (Hall, 2006).

Given the time constraints, an exhaustive study was not possible, but several experiments were conducted in order to compare arc-eager and arc-standard. We used the same feature model as in section 4.1 for arc-eager, and a slightly modified feature model for arc-standard compared to the one for arc-eager. The outcome seems to confirm previous studies. Arc-standard results in approximately 0.9%-points higher labeled AS accuracy for Chinese, but for all other languages the result is the opposite.[6] However, on average arc-eager outperforms arc-standard.

Given the rules of the contest, i.e. one parser for all languages, the use of more than one algorithm for the different languages is likely not allowed. Therefore, we took the decision to use arc-eager in the coming experiments.

Moreover, as mentioned in section 3, four treebanks contains "informative" root labels. Since the parser in its basic configuration cannot not assign informative labels to tokens without a head token, the use of an artificial root token solves this. We did two experiments for each language, one without artificial root (no AR) and another with artificial root (AR), and the results are shown in table 4.

The use of an artificial root improves accuracy substantially for all languages having informative root, i.e. Arabic, Czech, Portuguese and Slovene. The increase of especially high for Portuguese with an error reduction of approximately 25%. A look at the figures for the other languages shows small variations, although the average labeled accuracy for these languages drops slightly.

In the light of these observations, and given our interpretation the "one parser"-rule, we made the choice to use artificial root for the languages with informative roots, and use the original parsing algorithm without informative root for all others.

### 4.3 Pseudo-Projective Transformations

Since the version of the parsing algorithm implemented in MaltParser only can output projective dependency trees, the non-projectivity in the treebanks needs special treatment. Two approaches have been evaluated, filtering and pseudo-projective transformations. This section will present the experimental outcome of applying pseudo-projective transformations in order to deal with non-projective constructions.

Except for Turkish which uses MBL, all results are based on SVM although the machine learning parameters are not the same for all languages. The experiments follow the procedure of Nivre and

[6] Possibly also for Turkish, having 0.2%-points higher labeled AS for SVM using arc-standard

Nilsson (2005), comparing the encoding schemes BASELINE, HEAD, HEAD+PATH and PATH with the use of non-projective training data. BASELINE means that the training data is transformed, but no pseudo-projective information is added to the dependency labels. Consequently, no inverse transformation on the parser output is performed. Here we have also added FILTER, which simply removes all non-projective sentences from the training data prior training. The proportion of removed sentences is therefore the same as %NPS in table 1.

The results are summarized in table 5. For Chinese, which contains no non-projective constructions at all, the accuracy will not change since no sentences will be transformed. It has thus been omitted from this experiment.

The first thing to note is that BASELINE, i.e. projectivizing the training data without trying to do deprojectivize, yields a significant improvement for Danish, Dutch, Portuguese and Turkish. For the rest of the languages, the improvement is small or the decrease is negligible, with Czech as a surprising exception. The BASELINE-encoding did have a statistically significant improvement compared to NON-PROJ in Nivre and Nilsson (2005), with 0.7%-point higher labeled attachment score. It is however important to remember that there are at least four things that are different compared to that study, (1) the machine learning method, (2) division of the training and testing data, (3) the testing data contains gold standard tags in the contest but not in previous study, and (4) the information that the feature model looks at is not the same due to the data representation of the CoNLL-format.

Another thing to note when looking at HEAD, HEAD+PATH and PATH is that HEAD is the winner, even though they on average are virtually equally good at recovering non-projectivity. When we compare these three with NON-PROJ and BASELINE it is clear that the parser is able to correctly assign pseudo-projectivity arcs given in the pseudo-projective training data, and that the inverse transformation works. This indicates that the non-projectivity in several of the treebanks are regular enough for the parser to learn.

A comparison between the individual languages reveals a lot of variation in the effect of pseudo-projective parsing in relation to BASELINE. A first general and expected observation, which seems to hold, is that the more non-projectivity in a treebank, the more pseudo-projective parsing increases accuracy. We recorded the highest improvement for Dutch (5.4 %NPR) and German (2.3 %NPR) of 3.25 and 1.16%-points, respectively.

However, the proportion of non-projectivity does not tell the whole story. The figures indicate that the amount of training data is another important factor. For example Portuguese with 1.3 %NPR and 207k words increases accuracy by 0.49%-point, whereas Slovene with 1.9 %NPR and 27k words only increases by 0.28%-points, and Turkish with 1.5 %NPR and 58k words actually exhibits a decreased accuracy.

The table also shows that FILTERING has on average the lowest accuracy. The accuracy drops the most for Dutch, since the amount of non-projectivity seems to roughly correlate with the decrease in accuracy, with Turkish as an exception (and possibly also Spanish). This is likely due to the fact that the projective arcs in the non-projective sentences help more that the non-projective arcs are harmful.

Given the fact that pseudo-projective parsing using the HEAD-encoding only decreases accuracy marginally for some languages and helps several of them, we have taken the decision to use it for all languages in the final run.

### 4.4 Feature Optimization

An important factor to increase the labeled accuracy is to optimize the feature model for each language. To do a complete exhaustive search for all 13 languages is an impossible task given the time limit of the contest, and even if had more time it would still be hard to search for the optimal feature model. Instead we used two different strategies depending on the size of the training data:

1. Batch testing of new features by forward and backward selection

2. Investigate the properties of the language and use a feature model that we believed could capture these properties

For some languages we used both strategies and for other languages we only used one strategy. We used the feature model from the preliminary study as a

| | NON-PROJ | BASELINE | HEAD | HEAD+PATH | PATH | FILTER |
|---|---|---|---|---|---|---|
| Arabic | 62.97 | 62.95 | 62.66 | 62.72 | 62.73 | 61.98 |
| Bulgarian | 84.67 | 84.76 | 84.75 | 84.75 | - | - |
| Chinese | - | - | - | - | - | - |
| Czech | 72.51 | 72.15 | 73.01 | 73.06 | 72.99 | 71.05 |
| Danish | 80.79 | 81.11 | 81.39 | 81.26 | 81.29 | 80.73 |
| Dutch | 75.80 | 76.54 | 79.05 | 78.94 | 78.92 | 72.50 |
| German | 84.39 | 84.26 | 85.55 | 85.61 | 85.50 | 83.04 |
| Japanese | 90.47 | 90.36 | 90.45 | 90.40 | 90.40 | 89.97 |
| Portuguese | 84.34 | 84.84 | 85.33 | 85.32 | 85.22 | 83.89 |
| Slovene | 64.44 | 64.39 | 64.67 | 64.53 | 64.54 | 63.09 |
| Spanish | 77.09 | 76.80 | 77.10 | 76.83 | 77.01 | 77.19 |
| Swedish | 81.79 | 81.64 | 81.44 | 81.51 | 81.56 | 81.27 |
| Turkish | 63.0 | 63.7 | 63.5 | 63.5 | 63.5 | 63.6 |
| Average | 76.14 | 76.25 | 76.74 | 76.70 | 76.70 | 75.30 |

Table 5: Result for pseudo-projective dependency parsing using SVM. Average is computed without Bulgarian and Chinese

starting point for the feature optimization process. The CoNLL-X shared task format allows three more feature types (lemma, course-grained part-of-speech tags and morphological features) compared to what MaltParser could handle previously. The obvious choice was to start by adding these feature types for the languages that contains these feature types. It turned out that it was a good idea to add these three features for the token on top of the stack $\sigma[0]$ and next input token $\tau[0]$ for all the new feature types where these were present.

The batch testing strategy is a practical way to find an appropriate feature model when learning and parsing time are tractable. We constructed a feature selection program that generates several feature model files. The program could be executed in two modes: *add-one* or *leave-one-out*.[7] The add-one mode adds new features to a feature model one by one and the leave-one-out mode subtract one feature from the feature model at time. The program takes two files, one file with all features which should be kept constant and second file with all features that will be added or subtracted one by one. If there are $n$ features in the second file, the program will construct $n + 1$ feature model files.[8]

[7]Add-one and leave-one-out are in the literature also denoted forward and backward selection, respectively.

[8]The additional feature model is the feature model consisting of only the features that are constant.

We used a script to execute a set of experiments according how many feature model files generated. Thereafter, the script automatically evaluated all the experiments, and even summarized all the experiments into one file. We manually investigated the results of all the experiments, usually results of about 100 experiments. We picked out the feature that gave the best labeled accuracy. This feature was added to the file that contains the feature that should be kept constant and subtracted the feature from the other file, and the process was repeated all over again until there were no improvements. There were several feature candidates that improve the accuracy. In the beginning, we added two or three features to shorten this time consuming process, but this was not always a good idea because it sometimes decreased the accuracy when they were combined.

For languages with large treebanks (e.g. Czech) the batch testing strategy was impossible in practice given the time constraints of the contest. Instead we manually prepared feature model files that we believed could increase the accuracy and ran several experiments using these files.

Tables 7 and 8 show the optimized feature model for each language. Table 7 shows the features used for the feature types: part-of-speech (p), dependency type (d) of the partially built structure and word form (w). Table 8 presents the additional feature types

present in some of languages.[9]

### 4.5 SVM-Parameter Optimization

We decided to use SVM as the learning method for the contest, and specifically we decided to use the LIBSVM implementation of SVM. This library comes with many parameters which are used for optimizing the SVM learner for a specific task, in our case dependency parsing. The parameters that we tuned were briefly explained in section 2.5.

For some languages we did a grid-search for the best combinations of parameter settings. It is striking to see how sensitive these parameter settings are for different feature models. In the best of worlds, we should perform parameter optimization for each possible feature model. For instance, if we add one feature to a feature model, we would be forced to perform an exhaustive parameter search. This was not possible given the time limit, and even if we had several years and access to many powerful computers to perform this task, it would still be intractable do this search.

Unfortunately, we did not manage to perform an exhaustive parameter search for any language. For some languages, we simply used the parameter settings that we used in the preliminary experiments. Table 9 shows the final settings of the SVM learner. For all languages the polynomial kernel function of degree two was used. The kernel parameters $\gamma$, $r$ together with the penalty parameter $C$ and termination criteria $\epsilon$ were tuned for some languages. For six of the languages, the set of training instances was divided into smaller set based on the part-of-speech of next input token. The column $S$ in table 9 shows if the training instances were divided (D) or not (N). If the training instances were divided into smaller set, the $F$ column presents which part-of-speech feature was used: the coarse-grain part-of-speech tagset (C) and the fine-grain part-of-speech tagset (P). The last column $T$ specifies the frequency threshold $t$ that determines if a certain feature value should be pooled together with other values that occur less than $t$ times in the training data.

[9]For German and Swedish, there are no additional feature types and thus these two languages are not present in table 8.

### 4.6 Dry Run

To make sure that no mistakes were made, we performed what we called a dry run. During all experiments presented in the previous sections we used 80% of data for training and 20% for testing.

To perform this dry run we simply pretended that we had the final unparsed test data by using 10% of the data for testing and rest of data as training data. This test was also done to eliminate the risk of overfitting the models. The overall labeled accuracy for all languages was 81.0% for the dry run, compared with 81.1% for the 80/20 split.

For some languages the accuracy increased and for others it decreased. For Arabic it dropped quite a lot with 1.3%-points and for Slovene 1.2%-points, whereas for Bulgarian the accuracy increased 0.6%-points and for Czech and Swedish 0.5%-points. The treebanks for Slovene and Arabic are the smallest and it was not surprising that the results varied more than for the larger treebanks. To be on the safe side we performed some additional experiments for these two languages, but these experiments did not change the settings.

## 5 Final Evaluation

All experiments presented in section 4 led to several decisions. We decided to use these settings:

- Arc-eager version of Nivre's parsing algorithm
- For the data sets that include informative root labels (Arabic, Czech, Portuguese, Slovene) we used the artificial root method explained in section 2.2, but not for the other data sets
- SVM using different parameter settings for each languages, presented in table 9
- Pseudo-projective parsing using the HEAD-encoding for all languages
- One feature model for each language, listed in tables 7 and 8

We created one parser model for each language before the final test set was released by the organizer. In addition, we had created scripts that automatized the final test run. These scripts were also tested during the dry run.

Table 6 shows the final test results for each language as well as the average for all 13 languages based on the labeled attachment score. The fifth column shows our position in the contest, where our position is boldfaced. For example, for Spanish we are in second place, but there is no significant difference to the participant of the first and third position and therefore this is indicated by 1-**2**-3.

The labeled attachment score varies from 91.7 for Japanese to 65.7 for Turkish. We are above the mean results for all languages. Our average score is 80.2 for twelve languages (Bulgarian is excluded in the final evaluation). We have the best reported result for three languages: Japanese, Swedish and Turkish. Furthermore, we share the best reported result for Arabic, Danish, Dutch, Portuguese, Spanish. If we compare to the other participants, we end up in second place, but according to the organizer's significance tests there is no significant difference between us and the first place having the average score 80.3%.

In comparison to the dry run, the average score drops 0.8%-points for the final test run. The score for Dutch is more than 5%-points below the results for the dry run and for Slovene the score is 2%-points above the results obtained during the dry run. The most likely explanation is that the final test sets differ in complexity compared to the dry run test sets. The overall impression still seems to indicate that our models have not been overfitted to the training data.

Unfortunately, it is not possible to do a comparative analysis of our results compared to the other participants because the material needed for such a study will be published after the completion of this paper. One thing we can say about this issue is that we have native speakers of Swedish and Turkish in our group, and that our results are the best for these two languages. This indicates that knowledge about the language is of importance.

We can find some important facts in respect to the different data sets. A small data set is a good indicator of low accuracy, but not always a good indicator of high accuracy. The smallest data sets (Slovene, Arabic and Turkish) have the lowest accuracy below or close to 70%. On the other hand, the largest data sets (Czech and German) do not have the highest accuracy, e.g. there are at least four languages with higher accuracy than German.

| | Res | AV | SD | Pos |
|---|---|---|---|---|
| Arabic | 66.7 | 60.4 | 6.3 | 1-**2** |
| Bulgarian | 87.4 | 80.6 | 5.6 | 1-**2** |
| Chinese | 86.9 | 78.1 | 9.2 | **2**-3 |
| Czech | 78.4 | 67.6 | 9.0 | **2** |
| Danish | 84.8 | 72.6 | 15.9 | 1-**2** |
| Dutch | 78.6 | 70.7 | 6.5 | 1-**2**-3 |
| German | 85.8 | 78.8 | 7.6 | 2-**3** |
| Japanese | 91.6 | 85.9 | 7.0 | **1** |
| Portuguese | 87.6 | 81.1 | 5.6 | **1**-3 |
| Slovene | 70.3 | 65.4 | 7.0 | >3 |
| Spanish | 81.3 | 74.2 | 8.3 | 1-**2**-3 |
| Swedish | 84.6 | 76.7 | 6.0 | **1** |
| Turkish | 65.7 | 56.0 | 8.3 | **1** |
| Average | 80.2 | | | 1-**2** |

Table 6: Evaluation on final test set. Left column lists all the languages; Res: Our results, AV: average results over all participants, SD: Standard derivation, Pos: our position i the contest. Average exclude Bulgarian.

Another factor that has an impact on the accuracy is the average sentence length. Japanese has the highest accuracy consisting of short sentences length of about 9 tokens per sentence. The Chinese treebank has even shorter sentences, about 6 tokens per sentence in average, but has many distinct dependency types. The Arabic treebank has the longest sentences in average, about 37, and in combination with the fact that it is one of the smallest it has the lowest result.

The proportion of non-projective structures also influences the accuracy, even though we perform pseudo-projective parsing. The Dutch data set has the highest proportion of non-projective relations 5.4% and our result is below 80. Most likely, the high proportion of non-projective relations explains the low accuracy for Dutch. Moreover, the two largest treebanks (Czech and German) are also two of the treebanks that contain most non-projective relations (besides Dutch) about 2% and this can maybe contribute to the explanation that they are not amongst the top three.

## 6 Final Remarks

The goal of this project was to participate in the CoNLL-X Shared Task with our labeled pseudo-projective dependency parser (MaltParser), and looking back at the outcome we can see that we achieved a good result. We ended up in second place, or shared the first place when considering that there were no statistically significant difference between us and the winner. Even though many argue (e.g. Hall and Novák (2005)) that a deterministic parsing method is inferior compared to nondeterministic parsing techniques that provide an $n$-best ranking of the set of candidate analyzes, and without knowing the methods of the other participants, these results indicate that the former method is competitive.

This study confirms previous studies in three perspectives. First, SVM outperforms MBL as machine learning method for this kind of task, although the difference is less for the smaller data sets. Second, our conducted experiments strengthen the observation of Daelemans and Hoste (2002), that is, optimizing the feature model together with the machine learning algorithm is important.

Third, we now know that pseudo-projective transformations work for more languages than Czech, containing non-projectivity. The phenomenon non-projectivity exists in more or less all languages and is therefore a transformation technique that we can incorporate in the concept of "one parser". A recent study (Nilsson et al., 2006) on the Czech treebank reveals that other kinds of graph transformations, of coordination and compound verbs, can improve accuracy even more (and possibly also for other treebanks). However, according to our interpretation of the rules we decided to not include this.

To conclude, this project has given us lots of results to further analyze in the future. It will also be interesting to compare our approach to the others. In addition, we have access to and new knowledge of several data resources, which will be very useful in our future research. The final test sets of the CoNLL Shared Task will facilitate comparison when we evaluate new methods.

## References


A. Abeillé, editor. 2003. *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*. Kluwer Academic Publishers, Dordrecht.

S. Afonso, E. Bick, R. Haber, and D. Santos. 2002. "Floresta sintá(c)tica": a treebank for Portuguese. In *Proc. of the Third Intern. Conf. on Language Resources and Evaluation (LREC)*, pages 1698–1703.

N. B. Atalay, K. Oflazer, and B. Say. 2003. The annotation process in the Turkish treebank. In *Proc. of the 4th Intern. Workshop on Linguistically Interpreteted Corpora (LINC)*.

E. Black, F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, and S. 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.

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. 2003. The PDT: a 3-level annotation scenario. In Abeillé (Abeillé, 2003), chapter 7.

Pierre Boullier. 2003. Guided Earley parsing. In Gertjan van Noord, editor, *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 43–54.

S. Brants, S. Dipper, S. Hansen, W. Lezius, and G. Smith. 2002. The TIGER treebank. In *Proc. of the First Workshop on Treebanks and Linguistic Theories (TLT)*.

Chih-Chung Chang and Chih-Jen Lin. 2001. LIBSVM: A library for support vector machines.

K. Chen, C. Luo, M. Chang, F. Chen, C. Chen, C. Huang, and Z. Gao. 2003. Sinica treebank: Design criteria, representational issues and implementation. In Abeillé (Abeillé, 2003), chapter 13, pages 231–248.

M. Civit Torruella and M$^{a}$ A. Martí Antonín. 2002. Design principles for a Spanish treebank. In *Proc. of the First Workshop on Treebanks and Linguistic Theories (TLT)*.

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.

S. Džeroski, T. Erjavec, N. Ledinek, P. Pajas, Z. Žabokrtsky, and A. Žele. 2006. Towards a Slovene dependency treebank. In *Proc. of the Fifth Intern. Conf. on Language Resources and Evaluation (LREC)*.

J. Hajič, O. Smrž, P. Zemánek, J. Šnaidauf, and E. Beška. 2004. Prague Arabic dependency treebank: Development in data and tools. In *Proc. of the NEMLAR Intern. Conf. on Arabic Language Resources and Tools*, pages 110–117.

Keith Hall and Vaclav Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*.

Johan Hall, Joakim Nivre, and Jens Nilsson. 2006. Discriminative classifiers for deterministic dependency parsing. In *Proceedings of 44th Annual Meeting of the Association for Computational Linguistics and 21th International Conference on Computational Linguistics (COLING-ACL 2006)*.

Johan Hall. 2006. *MaltParser – An Architecture for Inductive Labeled Dependency Parsing*. Licentiat thesis, Växjö University.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2004. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University.

S. Kahane, A. Nasr, and O Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proceedings of 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL 1998)*.

Y. Kawata and J. Bartels. 2000. Stylebook for the Japanese treebank in VERBMOBIL. Verbmobil-Report 240, Seminar für Sprachwissenschaft, Universität Tübingen.

Martin Kay. 2000. Guides and oracles for linear-time parsing. In *Proceedings of the 6th International Workshop on Parsing Technologies (IWPT)*, pages 6–9.

M. T. Kromann. 2003. The Danish dependency treebank and the underlying linguistic theory. In *Proc. of the Second Workshop on Treebanks and Linguistic Theories (TLT)*.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings 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.

J. Nilsson, J. Hall, and J. Nivre. 2005. MAMBA meets TIGER: Reconstructing a Swedish treebank from antiquity. In *Proc. of the NODALIDA Special Session on Treebanks*.

Jens Nilsson, Joakim Nivre, and Johan Hall. 2006. Graph transformations in data-driven dependency parsing. In *Proceedings of 44th Annual Meeting of the Association for Computational Linguistics and 21th International Conference on Computational Linguistics (COLING-ACL 2006)*.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

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

J. Nivre, J. Hall, J. Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. 2006a. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006b. Maltparser: A data-driven parser-generator for dependency parsing. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC)*.

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

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer.

K. Oflazer, B. Say, D. Zeynep Hakkani-Tür, and G. Tür. 2003. Building a Turkish treebank. In Abeillé (Abeillé, 2003), chapter 15.

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.

Petr Sgall, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

K. Simov and P. Osenova. 2003. Practical annotation scheme for an HPSG treebank of Bulgarian. In *Proc. of the 4th Intern. Workshop on Linguistically Interpreteted Corpora (LINC)*, pages 17–24.

K. Simov, P. Osenova, A. Simov, and M. Kouylekov. 2005. Design and implementation of the Bulgarian HPSG-based treebank. In *Journal of Research on Language and Computation – Special Issue*, pages 495–522. Kluwer Academic Publishers.

L. van der Beek, G. Bouma, R. Malouf, and G. van Noord. 2002. The Alpino dependency treebank. In *Computational Linguistics in the Netherlands (CLIN)*.

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

Jorn Veenstra and Walter Daelemans. 2000. A memory-based alternative for connectionist shift-reduce parsing. Technical Report ILK-0012, University of Tilburg.

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.

Table 7: The final feature model for each language used for the dry and the final run. This table shows only the part-of-speech (p), the dependency type (d) and the word form (w) feature types.

| Feature | Ara | Bul | Chi | Cze | Dan | Dut | Ger | Jap | Por | Slo | Spa | Swe | Tur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p(\sigma_0)$* | + | + | + | + | + | + | + | + | + | + | + | + | + |
| $p(\tau_0)$* | + | + | + | + | + | + | + | + | + | + | + | + | + |
| $p(\tau_1)$* | + | + | + | + | + | + | + | + | + | + | + | + | + |
| $p(\tau_2)$* | + | + | + | + | + | + | + | + | + | + | + | + | |
| $p(\tau_3)$* | + | + | + | + | + | + | + | | + | | + | | |
| $p(\tau_4)$ | | | | + | | | | | | | | | |
| $p(\sigma_1)$* | + | + | + | + | + | + | + | + | + | + | + | + | + |
| $p(\sigma_2)$ | | | | | | | | | | | + | | |
| $p(rs(lc(\sigma_0)))$ | + | | | | | | | | | | | | |
| $p(rs(lc(\tau_0)))$ | | | | | | | | | | | | + | |
| $p(lc(\sigma_0))$ | | | + | | | | | + | | | + | | |
| $p(rc(\sigma_0))$ | | | + | | | | | | | | + | | |
| $p(lc(\tau_0))$ | | | | + | | | | | | | + | | |
| $p(l(\tau_0))$ | + | | | | + | | + | | | + | + | | |
| $p(l(\sigma_0))$ | | | | | | | + | | | | | | |
| $p(r(\sigma_0))$ | | | | | | | | | | + | + | + | + |
| $p(h(\sigma_1))$ | | | | | | | + | | | | | | |
| $p(h(\sigma_0))$ | | | | | | | + | | | | + | | |
| $d(\sigma_0)$* | + | + | + | + | + | + | + | + | + | + | + | + | + |
| $d(lc(\sigma_0))$* | + | + | | + | + | | + | + | + | + | + | + | + |
| $d(rc(\sigma_0))$* | + | + | + | + | + | + | + | + | + | + | + | + | + |
| $d(rc(\sigma_1))$ | + | | | | | | | | | | | | |
| $d(lc(\tau_0))$* | | + | + | + | + | + | + | + | + | + | + | + | + |
| $d(h(lc(\sigma_0)))$ | | | | | | | | | | | | + | |
| $d(ls(rc(\sigma_0)))$ | | | | | | | | | + | | | + | |
| $d(rs(lc(\sigma_0)))$ | | | | | | | | | | + | | | |
| $d(rs(lc(\tau_0)))$ | | | | | | | | + | | | | + | |
| $d(r(\sigma_0))$ | | | | | | + | | | | | | | |
| $w(\sigma_0)$* | + | + | + | + | + | + | + | + | + | + | + | + | |
| $w(\tau_0)$* | + | + | + | + | + | + | + | + | + | + | + | + | |
| $w(\tau_1)$* | + | + | + | + | + | + | + | + | + | + | | + | |
| $w(\tau_2)$ | | | | | | | + | | | | | | |
| $w(h(\sigma_0))$* | + | + | + | + | + | + | + | + | + | + | | + | |
| $w(l(\tau_0))$ | + | | | | + | | + | | | | | + | |
| $w(ls(rc(\sigma_0)))$ | + | | | | | | | | | | | | |
| $w(lc(\sigma_0))$ | | | + | | | | | + | | | | | |
| $w(lc(\tau_0))$ | | | | | + | | | | | | | + | |
| $w(rc(\sigma_0))$ | | | + | | + | | | | | | | | |
| $w(rs(lc(\sigma_0)))$ | | | | | | | | | | | | + | |
| $w(h(lc(\sigma_0)))$ | | | | | | | | + | | | | | |

Table 8: The final feature model for each language used for the dry and final run

| **Feature** | **Ara** | **Bul** | **Chi** | **Cze** | **Dan** | **Dut** | **Jap** | **Por** | **Slo** | **Spa** | **Tur** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $c(\sigma_0)$ | + | + | + | + | + | + | | + | + | + | + |
| $c(\tau_0)$ | + | + | + | + | + | + | + | + | + | + | + |
| $c(\tau_1)$ | | + | + | | | | | | | + | |
| $c(\sigma_2)$ | | | | | | | | | + | | |
| $c(lc(\sigma_0))$ | | | | | | + | | | | | |
| $c(l(\tau_0))$ | | | + | | + | | | | | | |
| $c(h(\sigma_0))$ | + | + | | | | | | | | + | |
| $fea(\sigma_0)$ | + | + | | + | + | + | | + | + | + | + |
| $fea(\tau_0)$ | + | + | | + | + | + | | + | + | + | + |
| $fea(\tau_1)$ | | | | + | | | | | | + | |
| $fea(\tau_2)$ | | | | + | | | | | + | | |
| $fea(\tau_3)$ | | | | + | | | | | + | | |
| $fea(h(\sigma_0))$ | | + | | | | | | | | + | |
| $fea(l(\tau_0))$ | | + | | | + | | | | | | |
| $fea(r(\tau_0))$ | | + | | | | | | | | | |
| $fea(lc(\sigma_0))$ | | + | | | | | | | | | |
| $fea(rc(\sigma_1))$ | | + | | | | | | | | | |
| $lem(\sigma_0)$ | + | | | + | | + | | + | + | + | + |
| $lem(\tau_0)$ | + | | | + | | + | | | | + | + |
| $lem(\tau_1)$ | | | | | | | | | | | + |
| $lem(l(\tau_0))$ | | | | | | + | | | | | |
| $lem(lc(\sigma_0))$ | | + | | | | + | | | | | |
| $lem(rc(\sigma_0))$ | | | | | | | | + | | | |
| $lem(lc(\tau_0))$ | | | | | | | | + | | + | |

Table 9: The final parameter settings for the SVM learner.

| **Language** | $\gamma$ | $C$ | $r$ | $\epsilon$ | $S$ | $F$ | $T$ |
|---|---|---|---|---|---|---|---|
| Arabic | .16 | .3 | 0 | 1 | N | | |
| Bulgarian | .2 | .3 | .3 | 0.1 | D | C | 1000 |
| Chinese | .2 | .3 | .3 | 0.1 | N | | |
| Czech | .2 | .5 | 0 | 1 | D | P | 200 |
| Danish | .2 | .6 | .3 | 1 | N | | |
| Dutch | .16 | .3 | 0 | 1 | N | | |
| German | .2 | .5 | 0 | 1 | D | P | 1000 |
| Japanese | .19 | .6 | 0 | 0.1 | N | | |
| Portuguese | .2 | .5 | 0 | 0.1 | N | | |
| Slovene | .2 | .1 | .8 | 0.1 | D | C | 600 |
| Spanish | .2 | .5 | 0 | 0.01 | D | P | 1000 |
| Swedish | .2 | .4 | 0 | 0.1 | N | | |
| Turkish | .12 | .7 | .6 | 0.01 | D | C | 100 |

Växjö
universitet

**Matematiska och systemtekniska institutionen**
SE-351 95 Växjö

tel 0470-70 80 00, fax 0470-840 04
www.msi.vxu.se