Experiments with a Multilanguage Non-Projective Dependency Parser

Giuseppe Attardi
Dipartimento di Informatica
largo B. Pontecorvo, 3
I-56127 Pisa, Italy
attardi@di.unipi.it

1 Introduction

Parsing natural language is an essential step in several applications that involve document analysis, e.g. knowledge extraction, question answering, summarization, filtering. The best performing systems at the TREC Question Answering track employ parsing for analyzing sentences in order to identify the query focus, to extract relations and to disambiguate meanings of words.

These are often demanding applications, which need to handle large collections and to provide results in a fraction of a second. Dependency parsers are promising for these applications since a dependency tree provides predicate-argument relations which are convenient for use in the later stages. Recently statistical dependency parsing techniques have been proposed which are deterministic and/or linear (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004). These parsers are based on learning the correct sequence of Shift/Reduce actions used to construct the dependency tree. Learning is based on techniques like SVM (Vapnik 1998) or Memory Based Learning (Daelemans 2003), which provide high accuracy but are often computationally expensive. Kudo and Matsumoto (2002) report a two week learning time on a Japanese corpus of about 8000 sentences with SVM. Using Maximum Entropy (Berger, et al. 1996) classifiers I built a parser that achieves a throughput of over 200 sentences per second, with a small loss in accuracy of about 2-3 %.

The efficiency of Maximum Entropy classifiers seems to leave a large margin that can be exploited to regain accuracy by other means. I performed a series of experiments to determine whether increasing the number of features or combining several classifiers could allow regaining the best accuracy. An experiment cycle in our setting requires less than 15 minutes for a treebank of moderate size like the Portuguese treebank (Afonso et al., 2002) and this allows evaluating the effectiveness of adding/removing features that hopefully might apply also when using other learning techniques.

I extended the Yamada-Matsumoto parser to handle labeled dependencies: I tried two approaches: using a single classifier to predict pairs of actions and labels and using two separate classifiers, one for actions and one for labels.

Finally, I extended the repertoire of actions used by the parser, in order to handle non-projective relations. Tests on the PDT (Böhmovà et al., 2003) show that the added actions are sufficient to handle all cases of non-projectivity. However, since the cases of non-projectivity are quite rare in the corpus, the general learner is not supplied enough of them to learn how to classify them accurately, hence it may be worthwhile to exploit a second classifier trained specifically in handling non-projective situations.

1. Summary of the approach

The overall parsing algorithm is an inductive statistical parser, which extends the approach by Yamada and Matsumoto (2003), by adding six new reduce actions for handling non-projective relations and also performs dependency labeling.

Parsing is deterministic and proceeds bottom-up. Labeling is integrated within a single processing step.

Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X), pages 166–170, New York City, June 2006. ©2006 Association for Computational Linguistics

The parser is modular: it can use several learning algorithms: Maximum Entropy, SVM, Winnow, Voted Perceptron, Memory Based Learning, as well as combinations thereof. The submitted runs used Maximum Entropy and I present accuracy and performance comparisons with other learning algorithms.

No additional resources are used.

No pre-processing or post-processing is used, except stemming for Danish, German and Swedish.

2 Features

Columns from input data were used as follows.

LEMMA was used in features whenever available, otherwise the FORM was used. For Danish, German and Swedish the Snowball stemmer (Porter 2001) was used to generate a value for LEMMA. This use of stemming slightly improved both accuracy and performance.

Only CPOSTAG were used. PHEAD/PDEPREL were not used.

FEATS were used to extract a single token combining gender, number, person and case, through a language specific algorithm.

The selection of features to be used in the parser is controlled by a number of parameters. For example, the parameter *PosFeatures* determines for which tokens the POS tag will be included in the context, *PosLeftChildren* determines how many left outermost children of a token to consider, *PastActions* tells how many previous actions to include as features.

The settings used in the submitted runs are listed below and configure the parser for not using any word forms. Positive numbers refer to input tokens, negative ones to token on the stack.

LemmaFeatures	*-2 -1 0 1 2 3*
PosFeatures	*-2 -1 0 1 2 3*
MorphoFeatures	*-1 0 1 2*
DepFeatures	*-1 0*
PosLeftChildren	*2*
PosLeftChild	*-1 0*
DepLeftChild	*-1 0*
PosRightChildren	*2*
PosRightChild	*-1 0*
DepRightChild	*-1*
PastActions	*1*

The context for POS tags consisted of 1 token left and 3 tokens to the right of the focus words, except for Czech and Chinese were 2 tokens to the left and 4 tokens to the right were used. These values were chosen by performing experiments on the training data, using 10% of the sentences as held-out data for development.

3 Inductive Deterministic Parsing

The parser constructs dependency trees employing a deterministic bottom-up algorithm which performs Shift/Reduce actions while analyzing input sentences in left-to-right order.

Using a notation similar to (Nivre and Scholz, 2003), the state of the parser is represented by a quadruple $\langle S, I, T, A\rangle$, where S is the stack, I is the list of (remaining) input tokens, T is a stack of temporary tokens and A is the arc relation for the dependency graph.

Given an input string W, the parser is initialized to $\langle (), W, (), ()\rangle$, and terminates when it reaches a configuration $\langle S, (), (), A\rangle$.

The parser by Yamada and Matsumoto (2003) used the following actions:

Shift	in a configuration $\langle S, n\|I, T, A\rangle$, pushes n to the stack, producing the configuration $\langle n\|S, I, T, A\rangle$.
Right[1]	in a configuration $\langle s_1\|S, n\|I, T, A\rangle$, adds an arc from s_1 to n and pops s_1 from the stack, producing the configuration $\langle S, n\|I, T, A\cup\{(s_1, r, n)\}\rangle$.
Left	in a configuration $\langle s_1\|S, n\|I, T, A\rangle$, adds an arc from n to s_1, pops n from input, pops s_1 from the stack and moves it back to I, producing the configuration $\langle S, s_1\|I, T, A\cup\{(n, r, s_1)\}\rangle$.

At each step the parser uses classifiers trained on treebank data in order to predict which action to perform and which dependency label to assign given the current configuration.

4 Non-Projective Relations

For handling non-projective relations, Nivre and Nilsson (2005) suggested applying a pre-processing step to a dependency parser, which consists in lifting non-projective arcs to their head repeatedly, until the tree becomes pseudo-projective. A post-processing step is then required to restore the arcs to the proper heads.

[1] Nivre and Scholz reverse the direction, while I follow here the terminology in Yamada and Matsumoto (2003).

I adopted a novel approach, which consists in adding six new parsing actions:

`Right2` in a configuration $\langle s_1|s_2|S, n|I, T, A\rangle$, adds an arc from s_2 to n and removes s_2 from the stack, producing the configuration $\langle s_1|S, n|I, T, A\cup\{(s_2, r, n)\}\rangle$.

`Left2` in a configuration $\langle s_1|s_2|S, n|I, T, A\rangle$, adds an arc from n to s_2, pops n from input, pops s_1 from the stack and moves it back to I, producing the configuration $\langle s_2|S, s_1|I, T, A\cup\{(n, r, s_2)\}\rangle$.

`Right3` in a configuration $\langle s_1|s_2|s_3|S, n|I, T, A\rangle$, adds an arc from s_3 to n and removes s_3 from the stack, producing the configuration $\langle s_1|s_2|S, n|I, T, A\cup\{(s_3, r, n)\}\rangle$.

`Left3` in a configuration $\langle s_1|s_2|s_3|S, n|I, T, A\rangle$, adds an arc from n to s_3, pops n from input, pops s_1 from the stack and moves it back to I, producing the configuration $\langle s_2|s_3|S, s_1|I, T, A\cup\{(n, r, s_3)\}\rangle$.

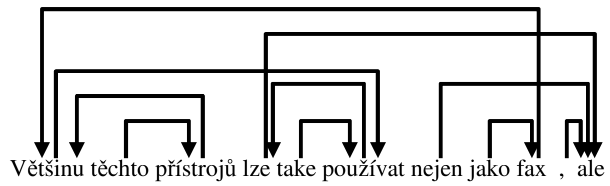
`Extract` in a configuration $\langle s_1|s_2|S, n|I, T, A\rangle$, move s_2 from the stack to the temporary stack, then `Shift`, producing the configuration $\langle n|s_1|S, I, s_2|T, A\rangle$.

`Insert` in a configuration $\langle S, I, s_1|T, A\rangle$, pops s_1 from T and pushes it to the stack, producing the configuration $\langle s_1|S, I, T, A\rangle$.

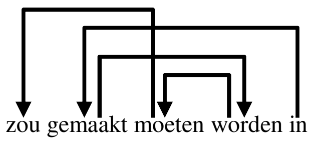
The actions `Right2` and `Left2` are sufficient to handle almost all cases of non-projectivity: for instance the training data for Czech contain 28081 non-projective relations, of which 26346 can be handled by `Left2`/`Right2`, 1683 by `Left3`/`Right3` and just 52 require `Extract`/`Insert`.

Here is an example of non-projectivity that can be handled with `Right2` (*nejen → ale*) and `Left3` (*fax → Většinu*):

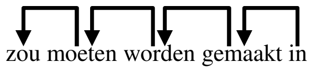
Většinu těchto přístrojů lze take používat nejen jako fax, ale současně …


Většinu těchto přístrojů lze take používat nejen jako fax , ale

The remaining cases are handled with the last two actions: `Extract` is used to postpone the creation of a link, by saving the token in a temporary stack; `Insert` restores the token from the temporary stack and resumes normal processing.



This fragment in Dutch is dealt by performing an `Extract` in configuration $\langle$*moeten*|*gemaakt*|*zou*, *worden*|*in*, *A*$\rangle$ followed immediately by an `Insert`, leading to the following configuration, which can be handled by normal `Shift`/`Reduce` actions:



Another linguistic phenomenon is the anticipation of pronouns, like in this Portuguese fragment:

`Tudo é possivel encontrar em o IX Salão de Antiguidades, desde objectos de ouro e prata, moedas, ...`

The problem here is due to the pronoun `Tudo` (*Anything*), which is the object of `encontrar` (*find*), but which is also the head of `desde` (*from*) and its preceding comma. In order to be able to properly link `desde` to `Tudo`, it is necessary to postpone its processing; hence it is saved with `Extract` to the temporary stack and put back later in front of the comma with `Insert`. In fact the pair `Extract`/`Insert` behaves like a generalized `Rightn`/`Leftn`, when n is not known. As in the example, except for the case where n=2, it is difficult to predict the value of n, since there can be an arbitrary long sequence of tokens before reaching the position where the link can be inserted.

5 Performance

I used my own C++ implementation of Maximum Entropy, which is very fast both in learning and classification. On a 2.8 MHz Pentium Xeon PC, the learning time is about 15 minutes for Portuguese and 4 hours for Czech. Parsing is also very fast, with an average throughput of 200 sentences per second: Table 1 reports parse time for parsing each whole test set. Using Memory Based Learning increases considerably the parsing time, while as expected learning time is quite shorter. On the other hand MBL achieves an improvement up to 5% in accuracy, as shown in detail in Table 1.

Language	Maximum Entropy						MBL				
	LAS %	**Cor-rected LAS**	**UAS %**	**LA %**	**Train time sec**	**Parse time sec**	**LAS %**	**UAS %**	**LA %**	**Train time sec**	**Parse time sec**
Arabic	*53.81*	54.15	*69.50*	72.97	181	2.6	**59.70**	**74.69**	75.49	24	950
Bulgarian	*72.89*	72.90	*85.24*	77.68	452	1.5	**79.17**	**85.92**	83.22	88	353
Chinese	*54.89*	70.00	*81.33*	58.75	1156	1.8	**72.17**	**83.08**	75.55	540	478
Czech	*59.76*	62.10	*73.44*	69.84	13800	12.8	**69.20**	**80.22**	77.72	496	13500
Danish	*66.35*	71.72	*78.84*	74.65	386	3.2	**76.13**	**83.65**	82.06	52	627
Dutch	*58.24*	63.71	*68.93*	66.47	679	3.3	**68.97**	**74.73**	75.93	132	923
German	*69.77*	75.88	*80.25*	78.39	9315	4.3	**79.79**	**84.31**	86.88	1399	3756
Japanese	*65.38*	78.01	*82.05*	73.68	129	0.8	**83.39**	**86.73**	89.95	44	97
Portuguese	*75.36*	79.40	*85.03*	80.79	1044	4.9	**80.97**	**86.78**	85.27	160	670
Slovene	*57.19*	60.63	*72.14*	69.36	98	3.0	**62.67**	**76.60**	72.72	16	547
Spanish	*67.44*	70.33	*74.25*	82.19	204	2.4	**74.37**	**79.70**	85.23	54	769
Swedish	*68.77*	**75.20**	*83.03*	72.42	1424	2.9	74.85	**83.73**	77.81	96	1177
Turkish	*37.80*	**48.83**	*65.25*	49.81	177	2.3	47.58	**65.25**	59.65	43	727

Table 1. Results for the CoNLL-X Shared task (official values in italics).

For details on the CoNLL-X shared task and the measurements see (Buchholz, et al. 2006).

6 Experiments

I performed several experiments to tune the parser.

I also tried alternative machine learning algorithms, including SVM, Winnow, Voted Perceptron.

The use of SVM turned out quite impractical since the technique does not scale to the size of training data involved: training an SVM with such a large number of features was impossible for any of the larger corpora. For smaller ones, e.g. Portuguese, training required over 4 days but produced a bad model which could not be used (I tried both the TinySVM (Kudo 2002) and the LIBSVM (Chang and Lin 2001) implementations).

Given the speed of the Maximum Entropy classifier, I explored whether increasing the number of features could improve accuracy. I experimented adding various features controlled by the parameters above: none appeared to be effective, except the addition of the previous action.

The classifier returns both the action and the label to be assigned. Some experiments were carried out splitting the task among several specialized classifiers. I experimented with:

1. three classifiers: one to decide between *`Shift`*/*`Reduce`*, one to decide which *`Reduce`* action and a third one to choose the dependency in case of *`Left`*/*`Right`* action
2. two classifiers: one to decide which action to perform and a second one to choose the dependency in case of *`Left`*/*`Right`* action

None of these variants produced improvements in precision. Only a small improvement in labeled attachment score was noticed using the full, non-specialized classifier to decide the action but discarding its suggestion for label and using a specialized classifier for labeling. However this was combined with a slight decrease in unlabeled attachment score, hence it was not considered worth the effort.

7 Error Analysis

The parser does not attempt to assign a dependency relation to the root. A simple correction of assigning a default value for each language gave an improvement in the LAS as shown in Table 1.

7.1 Portuguese

Out of the 45 dependency relations that the parser had to assign to a sentence, the largest number of errors occurred assigning *N<PRED* (62), *ACC* (46), *PIV* (43), *CJT* (40), *N<* (34), *P<* (30).

The highest number of head error occurred at the CPOS tags *PRP* with 193 and *V* with 176. In particular just four prepositions (*em*, *de*, *a*, *para*) accounted for 120 head errors.

Most of the errors occur near punctuations. Often this is due to the fact that commas introduce relative phrases or parenthetical phrases (e.g. "*o suspeito, de 38 anos, que trabalha*"), that produce diversions in the flow. Since the parser makes decisions analyzing only a window of tokens of a limited size, it gets confused in creating attachments. I tried to add some global context features, to be able to distinguish these cases, in particular, a count of the number of punctuation marks seen so far, whether punctuation is present between the focus words. None of them helped improving precision and were not used in the submitted runs.

7.2 Czech

Most current parsers for Czech do not perform well on *Apos* (apposition), *Coord* (coordination) and *ExD* (ellipses), but they are not very frequent. The largest number of errors occur on *Obj* (166), *Adv* (155), *Sb* (113), *Atr* (98). There is also often confusion among these: 33 times *Obj* instead of *Adv*, 32 *Sb* instead of *Obj*, 28 *Atr* instead of *Adv*.

The high error rate of *J* (adjective) is expected, mainly due to coordination problems. The error of *R* (preposition) is also relatively high. Prepositions are problematic, but their error rate is higher than expected since they are, in terms of surface order, rather regular and close to the noun. It could be that the decision by the PDT to hang them as heads instead of children, causes a problem in attaching them. It seems that a post-processing may correct a significant portion of these errors.

The labels ending with *_Co*, *_Ap* or *_Pa* are nodes who are members of the Coordination, Apposition or the Parenthetical relation, so it may be worth while omitting these suffixes in learning and restore them by post-processing.

An experiment using as training corpus a subset consisting of just sentences which include non-projective relations achieved a LAS of 65.28 % and UAS of 76.20 %, using MBL.

Acknowledgments. Kiril Ribarov provided insightful comments on the results for Czech.

The following treebanks were used for training the parser: (Afonso et al., 2002; Atalay et al., 2003; Böhmovà et al., 2003; Brants et al., 2002; Chen et al., 2003; Civit Torruella and Martì Antonìn, 2002; Džeroski et al., 2006; Hajiç et al., 2004; Kawata and Bartels, 2000; Kromann, 2003; Nilsson et al., 2005; Oflazer et al., 2003; Simov et al., 2005; van der Beek et al., 2002).

References

A. Berger, S. Della Pietra, and M. Della Pietra. 1996. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1).

S. Buchholz, et al. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proc. of the Tenth CoNLL.*

C.-C. Chang, C.-J. Lin. 2001. LIBSVM: a library for support vector machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm/

W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. 2003. Timbl: Tilburg memory based learner, version 5.0, reference guide. Technical Report ILK 03-10, Tilburg University, ILK.

T. Kudo. 2002. tinySVM. http://www.chasen.org/~taku/software/TinySVM/

T. Kudo, Y. Matsumoto. 2002. Japanese Dependency Analysis using Cascaded Chunking. In *Proc. of the Sixth CoNLL.*

R. McDonald, et al. 2005. Non-projective Dependency Parsing using Spanning Tree Algorithms. In *Proc. of HLT-EMNLP.*

J. Nivre, et al. 2004. Memory-based Dependency Parsing. In *Proc.s of the Eighth CoNLL*, ed. H. T. Ng and E. Riloff, Boston, Massachusetts, pp. 49–56.

J. Nivre and M. Scholz. 2004. Deterministic Dependency Parsing of English Text. In *Proc. of COLING 2004*, Geneva, Switzerland, pp. 64–70.

J. Nivre and J. Nilsson, 2005. Pseudo-Projective Dependency Parsing. In *Proc. of the 43rd Annual Meeting of the ACL*, pp. 99-106.

M.F. Porter. 2001. Snowball Stemmer. http://www.snowball.tartarus.org/

V. N. Vapnik. 1998. The Statistical Learning Theory. Springer.

H. Yamada and Y. Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proc. of the 8th International Workshop on Parsing Technologies* (*IWPT*), pp. 195–206.