CUNI: Feature Selection and Error Analysis of a Transition-Based Parser

Daniel ZEMAN
(1) Charles University in Prague, Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics, Malostranské náměstí 25, Praha, Czechia
zeman@ufal.mff.cuni.cz

ABSTRACT
We describe the parsing system used at the Charles University (CUNI) for the Hindi Parsing Shared Task 2012. We used the publicly available Malt Parser, which is highly configurable. A substantial part of the paper describes the configuration that we selected. The parser performs reasonably well in identifying the head nodes. The main weakness is in labeling the dependency relations. We identify the most prominent error types, which should help to improve the parsing accuracy in future.

TITLE AND ABSTRACT IN CZECH

CUNI: Výběr rysů a analýza chyb parseru založeného na přechodech

Popisujeme systém pro syntaktickou analýzu použitý na Univerzitě Karlově (CUNI) pro Hindi Parsing Shared Task 2012. Použili jsme veřejně dostupný nástroj Malt Parser, který poskytuje mnoho možností konfigurace. Podstatná část článku se zabývá právě konfigurací, kterou jsme zvolili. Parser dosahuje dobré úspěšnosti při identifikaci rodičovských uzlů. Jeho hlavní slabinou je značkování závislostních vztahů. Popisujeme nejběžnější druhy chyb, což by mělo pomoci v budoucnosti zvýšit úspěšnost parseru.

KEYWORDS: parsing, dependency, SOV language, malt parser.

CZECH KEYWORDS: syntaktická analýza, závislost, jazyk SOV, malt parser.

Proceedings of the Workshop on Machine Translation and Parsing in Indian Languages (MTPIL-2012), pages 143–148,
COLING 2012, Mumbai, December 2012.

1 Introduction

Dependency parsing has gradually become standard way of parsing for many languages during the past decade. There have been three multilingual shared tasks in dependency parsing at CoNLL (2006, 2007 and 2009) and two focused on Indian languages, organized at ICON (2009 and 2010). This work on parsing Hindi loosely follows from our contribution to ICON (Zeman, 2009).

2 Data

The data provided are split to three parts: training, development and test. During the preparatory stage, we trained the parser on the training dataset and tested it on the development dataset. Then the blind test data were released. We retrained the parser on both the training and the development datasets, and applied it to the test data. Note that at this stage, we did not have access to the gold-standard syntactic annotation of the test data and we could not measure the accuracy on the test set. Later on, after the official results were announced, the full test set was released.

From now on, we will refer to the original training data as *dtrain*, to the development data as *dtest*, to the combination of dtrain + dtest as *etrain* and to the final test data as *etest*.

	Tokens	Sentences
dtrain	268096	12041
dtest	26416	1233
etrain	294512	13274
etest	39775	1828

Table 1: Size of the various datasets.

The data were provided in two file formats and two encodings of the Devanāgarī script. We work exclusively with the CoNLL data format (Buchholz and Marsi, 2006) and the UTF-8 encoding.

Two versions of morphological annotation were provided, corresponding to two tracks of the shared task: *Gold* and *Auto*. As suggested by its name, the Gold version contains more information and more accurate information. In the Auto version, values of some token attributes were assigned using automated tools, and values of the (many!) remaining attributes were empty because no automatic tool was available to assign them.

An example of one token from the Gold training data follows (including transliteration):

Index	Token	Lemma	CPOS	POS	Features	Head index	Dep label
32	किया	कर	VM	v	lex-कर|cat-…	0	main
32	*kiyā*	*kara*	VM	v	lex-*kara*|cat-…	0	main

Wherein the features consist of a long, vertical-bar-delimited list (here displayed as a table):

lex	cat	gen	num	pers	case	vib	tam	chunkId	chunkType	stype
कर	v	m	sg	any		या	yA	VGF	head	declarative’>

voicetype
active

The corresponding line of the same token in the Auto data is much simpler:

Index	Token	Lemma	CPOS	POS	Features	Head index	Dep label
32	किया	_	VM	_	_	0	main
32	*kiyā*	_	VM	_	_	0	main

Slightly more than 1% of the tokens in Hindi are attached non-projectively. As a distinct feature of the Hyderabad treebank, there are special NULL nodes covering words deleted from surface, e.g. in deficient coordinations, as in:

दीवाली	के	दिन	जुआ	खेलें	मगर	NULL	घर	में	या	होटल	में.
dīvālī	*ke*	*din*	*juā*	*khelẽ*	*magar*	NULL	*ghar*	*mẽ*	*yā*	*hoṭal*	*mẽ.*
Diwali	of	day	gambling	play	but	[do-so]	house	in	or	hotel	in

"They do gamble on Diwali but they do so in house or in hotel."

The NULL nodes are present also in test input and it is not the task of the parser to introduce them. Approximately 0.4% of all nodes are NULL nodes.

3 Parser

We use Malt Parser v. 1.7.1[1] (Nivre et al., 2007). It is a deterministic shift-reduce parser where input words can be either put to the stack or taken from the stack and combined to form a dependency. The decision which operation (transition) to perform is made by an oracle based on various features of the words in the input buffer and the stack. The default machine learning algorithm used to train the oracle is a sort of SVM (support vector machine) classifier (Cristianini and Shawe-Taylor, 2000).

Malt Parser has participated in several CoNLL shared tasks in multilingual dependency parsing (2006, 2007 and 2009), as well as in the ICON 2009 NLP tools contest in parsing Indian languages. It achieves state-of-the-art accuracy (or close to it) for many languages, provided its settings, algorithm and feature space are optimized for the given language and dataset. Malt Parser is reasonably fast and it is open-source, freely available for download.

Malt Parser provides several parsing algorithms. They differ in the data structures they use, in the sets of transition operations they allow, and in the precedence of transition types when converting a training tree to a sequence of transitions. The algorithms are as follows: *nivrestandard, nivreeager, covproj, covnonproj, stackproj, stackeager, stacklazy, planar, 2planar.* We need one of the non-projective algorithms because there are occasional non-projective dependencies in the data. Based on dtest scores, we selected the *stacklazy* algorithm.

Malt Parser is also highly configurable with respect to the features considered when selecting the next transition. Table 2 gives an overview of features that we used.

Training times depend on the number of features, thus it is much faster to train on the Auto data than on the Gold data. Training Malt parser on the full etrain dataset takes about 2 hours in the Auto version and about 6 hours in the Gold version (tested on 2GHz 64bit Intel Xeon processors with 29GB RAM dedicated to the Java Virtual Machine). Parsing etest took between 30 (Auto) and 45 (Gold) minutes. For the sake of quick probing experiments

[1] http://www.maltparser.org/

		FORM	LEMMA	POS	FEATS	DEPREL
Stack:	top	1	2	3	4	
Stack:	$top - 1$		5	6	7	
Stack:	$top - 2$			8	9	
Stack:	$top - 3$			10		
Input:	$next$			11		
Lookahead:	$next$	12	13	14	15	
Lookahead:	$next + 1$	16	17	18	19	
Lookahead:	$next + 2$	20		21	22	
Lookahead:	$next + 3$			23		
Lookahead:	$next + 4$			24		
Tree:	leftmost dep of top			25		26
Tree:	leftmost dep of $top - 1$					27
Tree:	leftmost dep of $top - 2$					28
Tree:	rightmost dep of top					29
Tree:	rightmost dep of $top - 1$					30
Tree:	rightmost dep of $top - 2$					31
String:	predecessor of top	32	33			

Table 2: Feature pool for optimization with columns representing data fields and rows representing tokens relative to the stack, input buffer / lookahead, partially built tree, and input string. Features are numbered.

we also used the subset of the first 1000 training sentences. With such limited training corpus, accuracy dropped about 12 percent points down and processing times dropped to just a few minutes.

4 Results

Morphology	UAS	LA	LAS
Auto	89.96	84.09	81.48
Gold	95.19	91.27	89.48

Table 3: The official results as measured by the organizers of the shared task. UAS = unlabeled attachment score; LA = labeling accuracy; LAS = labeled attachment score.

5 Error Analysis

In our assessment of typical error patterns we will focus on the Gold track. The (unlabeled) attachment score is very high, which implies that it is difficult to pull out frequent errors. Nevertheless, there is some evidence that coordinations, commonly perceived as hard to parse, still correlate with errors in our data.

If we break up tokens by their coarse part of speech, then the most successful classes (in terms of finding their correct parent) will be PSP (postposition), VAUX (auxiliary verb) and DEM (demonstrative), all scoring (almost) 100%. At the other end of the scale, coordinating conjunctions (CC) only achieve 83%. The conjunction serves as the head node of the coordination. The parser probably feels uncertain about the type of the subtree of the conjunction, which makes it difficult to find the correct parent for the conjunction.

Besides conjunctions, the second most difficult part of speech is RB (adverb of manner; 89%). If we look at word forms, again, conjunctions are most risky: और, कि, व.

Not surprisingly, attachments to the artificial root node are more difficult (96%) to recognize that dependencies inside the tree. For tree-internal arcs, length 1 has 99% accuracy and longer links decrease down to 92%.

While the parser rarely confuses parent selection, its weakness lies in labeling of the dependencies. The nature of the label set used in the Hyderabad treebank excludes certain combinations of dependencies under one parent; in particular, there should not be two siblings both labeled *k1*. However, the parser has only limited means of learning such constraints. We cannot define a feature that would tell whether we already labeled *k1* anything anywhere in the current tree. Labels with the lowest precision and recall are k1 (87%), k2 (76%), k7 (80%), k7t (88%), pof (82%), r6 (92%) and ccof (93%; again, this is coordination). Most of these are typical verb complements.

The most frequent label confusion occurred between r6-k2 (karma of a conjunct verb) and r6 (possessive), presumably because both occur with the postposition का. Another relatively frequent confusion is between k7p (location-place) and k7 (location other than place).

6 Conclusion

We described our configuration of the Malt Parser used to parse Hindi part of the Hyderabad Dependency Treebank ver. 0.51. Feature engineering is the key to success in discriminative parsing. One of the main advantages of Malt Parser is that many different features can be used to describe tokens in the various data structures of the parser. Later in the paper, we evaluated the output of the parser and identified the most frequent error types. Future work should focus on two separate problems: 1. How to improve attachment of coordinations, and 2. How to improve labeling accuracy. For example, coordinations might improve by introducing features that would better port the phrase type to the higher levels. As for dependency labels, a self-standing tagger applied during post-processing could be tested.

Acknowledgements

The work on this project was supported by the grants P406/11/1499 of the Czech Science Foundation (GAČR), FP7-ICT-2009-4-249119 (MetaNet) and by research resources of the Charles University in Prague (PRVOUK).

References

Buchholz, S. and Marsi, E. (2006). CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City. Association for Computational Linguistics.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge University Press, Cambridge, UK.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryiğit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Zeman, D. (2009). Maximum spanning malt: Hiring world's leading dependency parsers to plant indian trees. In *Proceedings of ICON09 NLP Tools Contest: Indian Language Dependency Parsing*, pages 18–23, Hyderabad, India. International Institute of Information Technologies, Hyderabad.