372:Comparing the Benefit of Different Dependency Parsers for Textual Entailment Using Syntactic Constraints Only

Alexander Volokh
alexander.volokh@dfki.de
DFKI
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany

Günter Neumann
neumann@dfki.de
DFKI
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany

Abstract

We compare several state of the art dependency parsers with our own parser based on a linear classification technique. Our primary goal is therefore to use syntactic information only, in order to keep the comparison of the parsers as fair as possible. We demonstrate, that despite the inferior result using the standard evaluation metrics for parsers like UAS or LAS on standard test data, our system achieves comparable results when used in an application, such as the SemEval-2 #12 evaluation exercise PETE. Our submission achieved the 4th position out of 19 participating systems. However, since it only uses a linear classifier it works 17-20 times faster than other state of the parsers, as for instance MaltParser or Stanford Parser.

1 Introduction

Parsing is the process of mapping sentences to their syntactic representations. These representations can be used by computers for performing many interesting natural language processing tasks, such as question answering or information extraction. In recent years a lot of parsers have been developed for this purpose.

A very interesting and important issue is the comparison between a large number of such parsing systems. The most widespread method is to evaluate the number of correctly recognized units according to a certain gold standard. For dependency-based units unlabeled or labeled attachment scores (percentage of correctly classified dependency relations, either with or without the dependency relation type) are usually used (cf. Buchholz and Marsi, 2006).

However, parsing is very rarely a goal in itself. In most cases it is a necessary preprocessing step for a certain application. Therefore it is usually not the best option to decide which parser suits one's goals best by purely looking on its performance on some standard test data set. It is rather more sensible to analyse whether the parser is able to recognise those syntactic units or relations, which are most relevant for one's application.

The shared task #12 PETE in the SemEval-2010 Evaluation Exercises on Semantic Evaluation (Yuret, Han and Turgut, 2010) involved recognizing textual entailments (RTE). RTE is a binary classification task, whose goal is to determine, whether for a pair of texts T and H the meaning of H is contained in T (Dagan et al., 2006). This task can be very complex depending on the properties of these texts. However, for the data, released by the organisers of PETE, only the syntactic information should be sufficient to reliably perform this task. Thus it offers an ideal setting for evaluating the performance of different parsers.

To our mind evaluation of parsers via RTE is a very good additional possibility, besides the usual evaluation metrics, since in most cases the main thing in real-word applications is to recognize the primary units, such as the subject, the predicate,

the objects, as well as their modifiers, rather than the other subordinate relations.

We have been developing our own a multilingual dependency parser (called MDParser), which is based on linear classification[1]. Whereas the system is quite fast because the classification is linear, it usually achieves inferior results (using UAS/LAS evaluation metrics) in comparison to other parsers, which for example use kernel-based classification or other more sophisticated methods.

Therefore the PETE shared task was a perfect opportunity for us to investigate whether the inferior result of our parser is also relevant for its applicability in a concrete task. We have compared our system with three state of the art parsers made available on the PETE web page: MaltParser, MiniPar and StandfordParser. We have achieved the total score of 0.6545 (200/301 correct answers on the test data), which is the 4th rank out of 19 submissions.

2 MDParser

MDParser stands for multilingual dependency parser and is a data-driven system, which can be used to parse text of an arbitrary language for which training data is available. It is a transition-based parser and uses a deterministic version of the Covington's algorithm (Covington, 2000).

The models of the system are based on various features, which are extracted from the words of the sentence, including word forms and part of speech tags. No additional morphological features or lemmas are currently used in our models, even if they are available in the training data, since the system is especially designed for processing plain text in different languages, and such components are not available for every language.

The preprocessing components of MDParser include a.) a sentence splitter[2], since the parser constructs a dependency structure for individual sentences, b.) a tokenizer, in order to recognise the elements between which the dependency relations will be built[3], and c.) a part of speech tagger, in order to determine the part of speech tags, which are intensively used in the feature models[4].

MDParser is an especially fast system because it uses a linear classification algorithm L1R-LR(L1 regularised logistic regression) from the machine learning package LibLinear (Lin et al., 2008) for constructing its dependency structures and therefore it is particularly suitable for processing very large amounts of data. Thus it can be used as a part of larger applications in which dependency structures are desired.

Additionally, significant efforts were made in order to make the gap between our linear classification and more advanced methods as small as possible, e.g. by introducing features conjunctions, which are complex features built out of ordinary features, as well as methods for automatically measuring feature usefulness in order to automate and optimise feature engineering.

3 Triple Representation

Every parser usually produces its own somehow special representation of the sentence. We have created such a representation, which we will call *triple representation* and have implemented an automatic transformation of the results of Minipar, MaltParser, Stanford Parser and of course MDParser into it (cf. Wang and Neumann, 2007).

The triple representation of a sentence is a set of triple elements of the form <*parent*, *label*, *child*>, where *child* and *parent* elements stand for the head and the modifier words and their parts of speech, and *label* stands for the relation between them. E.g. <have:VBZ, SBJ, Somebody:NN>. This information is extractable from the results of any dependency parser.

4 Predicting Entailment

Whereas the first part of the PETE shared task was to construct syntactic representations for all T-H-pairs, the second important subtask was to determine whether the structure of H is entailed by the structure of T. The PETE guide[5] states that the following three phenomena were particularly important to recognise the entailment relation:

[1]http://www.dfki.de/~avolokh/mdparser.pdf
[2]http://morphadorner.northwestern.edu/morphadorner/sentencesplitter/
[3]http://morphadorner.northwestern.edu/morphadorner/wordtokenizer/
[4]The part of speech tagger was trained with the SVMTool http://www.lsi.upc.edu/~nlp/SVMTool/
[5]http://pete.yuret.com/guide

1. subject-verb dependency (*John kissed Mary.* → *John kissed somebody.*)
2. verb-object dependency (*John kissed Mary* → *Mary was kissed.*)
3. noun-modifier dependency (*The big red boat sank.* → *The boat was big.*)

Thus we have manually formulated the following **generic decision rule** for determining the entailment relation between T and H:

1. identify the root triple of H <null:null, ROOT, x>

2. check whether the subject and the complements(objects, verb complements) of the root word in H are present in T. Formally: all triples of H of the form <x, z, y> should be contained in T(x in 1 and 2 is thus the same word).

3. if 2 returns false we have to check whether H is a structure in passive and T contains the same content in active voice(a) or the other way around(b). Formally:

3a. For triples of the form <be:VBZ, SBJ, s> and <be:VBZ, VC, t> in H check whether there is a triple of the form <s, NMOD, t> in T.

3b. For triples of the form <u, OBJ,v> in H check whether there is a triple of the form <v, NMOD, u> in T.

It turned out that few **additional modifications** to the base rule were necessary for some sentences: 1.) For sentences containing conjunctions: If we were looking for a subject of a certain verb and could not find it, we investigated whether this verb is connected via a conjunction with another one. If true, we compared the subject in H with the subject of the conjunct verb. 2.) For sentences containing special verbs, e.g. modal verbs *may* or *can* or auxiliary verbs like *to have* it turned out to be important to go one level deeper into the dependency structure and to check whether all of their arguments in H are also present in T, the same way as in 3.

A triple <x,z,y> is contained in a set of triples S, when there exists at least one of the triples in S <u,w,v>, such that x=u, w=z and y=v. This is also true if the words *somebody, someone* or *something* are used on one of the equation sides. Moreover, we use an English lemmatizer for all word forms, so when checking the equality of two words we actually check their lemmas, e.g., *is* and *are* are also treated equally.

5 Results

We have parsed the 66 pairs of the development data with 4 parsers:[6] MiniPar, Stanford Parser, MaltParser and MDParser. After applying our rule we have achieved the following result:

	Accuracy	Parsing Speed
MiniPar	45/66	1233 ms
Stanford Parser	50/66	32889 ms
MaltParser	51/66	37149 ms
MDParser	50/66	1785 ms

We used the latest versions of MiniPar[7] and Stanford Parser[8]. We did not re-test the performance of these parsers on standard data, since we were sure that these versions provide the best possible results of these systems.

As far as the MaltParser is concerned we had to train our own model. We have trained the model with the following LibSVM options: "-s_0_-t_1_-d_2_-g_0.18_-c_0.4_-r_0.4_-e_1.0". We were able to achieve a result of 83.86% LAS and 87.25% UAS on the standard CoNLL English test data, a result which is only slightly worse than those reported in the literature, where the options are probably better tuned for the data. The training data used for training was the same as for MDParser.

The application of our rule for MDParser and MaltParser was fully automated, since both use the same training data and thus work over the same tag sets. For MiniPar and Stanford Parser, which construct different dependency structures with different relation types, we had to go through all pairs manually in order to investigate how the rule should be adopted to their tag sets and structures. However, since we have already counted the number of structures, for which an adoptation of the rule would work during this investigation, we did not implement it in the end. Therefore these results might be taken with a pinch of salt, despite the fact that we have tried to stay as fair as possible and treated some pairs as correct, even if a quite large modification of the

[6]For all results reported in this section a desktop PC with an Intel Core 2 Duo E8400 3.00 GHz processor and 4.00 GB RAM was used.

[7]http://webdocs.cs.ualberta.ca/~lindek/minipar

[8]http://nlp.stanford.edu/downloads/lex-parser.shtml

rule was necessary in order to adopt it to the different tag set and/or dependency structure.

For test data we were only able to apply our rule for the results of MDParser and MaltParser, since for such a large number of pairs (301) only the fully automated version of our mechanism for predicting entailment could be applied. For MiniPar and Stanford Parser it was too tedious to apply it to them manually or to develop a mapping between their dependency annotations and the ones used in MDParser or MaltParser. Here are the official results of our submissions for MaltParser and MDParser:

	Accuracy	Parsing Speed
MDParser	197/301	8704 ms
MaltParser	196/301	147938 ms

6 Discussion

We were able to show that our parser based on a linear classification technique is especially fast compared to other state of the art parsers. Furthermore, despite the fact, that it achieves an inferior result, when using usual evaluation metrics like UAS or LAS, it is absolutely suitable for being used in applications, since the most important dependency relations are recognized correctly even with a less sophisticated linear classifier as the one being used in MDParser.

As far as the overall score is concerned we think a much better result could be achieved, if we would put more effort into our mechanism for recognizing entailment using triple representations. However, many of the pairs required more than only syntactical information. In many cases one would need to extend one's mechanism with logic, semantics and the possibility to resolve anaphoric expressions, which to our mind goes beyond the idea behind the PETE task. Since we were primarly interested in the comparison between MaltParser and MDParser, we have not tried to include solutions for such cases. Here are some of the pairs we think require more than only syntax:

(4069 entailment="YES") <t>Mr. Sherwood speculated that the leeway that Sea Containers has means that Temple would have to "substantially increase their bid if they're going to top us."</t>

<h>Someone would have to increase the bid.</h>

(7003 entailment="YES") <t>After all, if you were going to set up a workshop you had to have the proper equipment and that was that.</t>

<h>Somebody had to have the equipment.</h>

(3132.N entailment="YES") <t>The first was that America had become -- or was in danger of becoming -- a second-rate military power.</t>

<h>America was in danger.</h>

→ 4069, 7003 and 3132.N are examples for sentences were beyond syntactical information logic is required. Moreover we are surprised that sentences of the form "if **A**, then **B**" entail **B** and a sentence of the form "**A** or **B**" entails **B**, since "*or"* in this case means uncertainty.

(4071.N entailment="NO") <t>Interpublic Group said its television programming operations -- which it expanded earlier this year -- agreed to supply more than 4,000 hours of original programming across Europe in 1990.</t>

<h>Interpublic Group expanded.</h>

(6034 entailment="YES") <t>"Oh," said the woman, "I've seen that picture already."</t>

<h>The woman has seen something.</h>

→ In 4071.N one has to resolve "*it*" in "*it expanded*" to *Interpublic Group*. In 6034 one has to resolve "*I*" in "*I've seen*" to "*the woman*". Both cases are examples for the necessity of anaphora resolution, which goes beyond syntax as well.

(2055) <t>The Big Board also added computer capacity to handle huge surges in trading volume.</t>

<h>Surges were handled.</h>

→ If something is added in order to do something it does not entail that this something is thus automatically done. Anyways pure syntax is not sufficient, since the entailment depends on the verb used in such a construction.

(3151.N) <t>Most of them are Democrats and nearly all consider themselves, and are viewed as, liberals.</t>

<h>Some consider themselves liberal.</h>

→ One has to know that the semantics of "consider themselves as liberals" and "consider themselves liberal" is the same.

Acknowledgements

The work presented here was partially supported by a research grant from the German Federal Ministry of Economics and Technology (BMWi) to the DFKI project Theseus Ordo TechWatch (FKZ: 01MQ07016). We thank Joakim Nivre and Johan Hall for their support and tips when training models with MaltParser. Additionally, we are very grateful to Sven Schmeier for providing us with a trained part of speech tagger for English and for his support when using this tool.

References

Michael A. Covington, 2000. *A Fundamental Algorithm for Dependency Parsing*. In Proceedings of the 39th Annual ACM Southeast Conference.

Dan Klein and Christopher D. Manning, 2003. *Accurate Unlexicalized Parsing*. Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.

Lin D, 2003. *Dependency-Based Evaluation Of Minipar*. In Building and using Parsed Corpora Edited by: Abeillé A. Dordrecht: Kluwer; 2003.

Sabine Buchholz and Erwin Marsi. 2006. *CoNLL-X shared task on multilingual dependency parsing*. In Proceedings of CONLL-X, pages 149–164, New York.

Ido Dagan, Oren Glickman and Bernardo Magnini. *The PASCAL Recognising Textual Entailment Challenge*. In Quinonero-Candela, J.; Dagan, I.; Magnini, B.; d'Alche-Buc, F. (Eds.), Machine Learning Challenges. Lecture Notes in Computer Science, Vol. 3944, pp. 177-190, Springer, 2006.

Nivre, J., J. Hall and J. Nilsson, 2006. *MaltParser: A Data-Driven Parser-Generator for Dependency Parsing*. In Proceedings of the fifth international conference on Language Resources and Evaluation (LREC2006), May 24-26, 2006, Genoa, Italy, pp. 2216-2219.

Rui Wang and Günter Neumann, 2007. *Recognizing Textual Entailment Using a Subsequence Kernel Method*. In Proceedings of AAAI 2007.

R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin, 2008. *LIBLINEAR: A Library for Large Linear Classification*. Journal of Machine Learning Research, 9(4): 1871–1874.

Deniz Yuret, Aydın Han and Zehra Turgut, 2010. *SemEval-2010 Task 12: Parser Evaluation using Textual Entailments.* In Proceedings of the SemEval-2010 Evaluation Exercises on Semantic Evaluation.

The Stanford Parser: A Statistical Parser. http://nlp.stanford.edu/downloads/lex-parser.shtml

Maltparser. http://maltparser.org/

Minipar. http://webdocs.cs.ualberta.ca/~lindek/minipar.htm

MDParser: Multilingual Dependency Parser. http://mdparser.sb.dfki.de/