Automatic Detection and Correction of Errors in Dependency Tree-banks

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

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

Abstract

Annotated corpora are essential for almost all NLP applications. Whereas they are expected to be of a very high quality because of their importance for the followup developments, they still contain a considerable number of errors. With this work we want to draw attention to this fact. Additionally, we try to estimate the amount of errors and propose a method for their automatic correction. Whereas our approach is able to find only a portion of the errors that we suppose are contained in almost any annotated corpus due to the nature of the process of its creation, it has a very high precision, and thus is in any case beneficial for the quality of the corpus it is applied to. At last, we compare it to a different method for error detection in treebanks and find out that the errors that we are able to detect are mostly different and that our approaches are complementary.

1 Introduction

Treebanks and other annotated corpora have become essential for almost all NLP applications. Papers about corpora like the Penn Treebank [1] have thousands of citations, since most of the algorithms profit from annotated data during the development and testing and thus are widely used in the field. Treebanks are therefore expected to be of a very high quality in order to guarantee reliability for their theoretical and practical uses. The construction of an annotated corpus involves a lot of work performed by large groups. However, despite the fact that a lot of human post-editing and automatic quality assurance is done, errors can not be avoided completely [5].

In this paper we propose an approach for finding and correcting errors in dependency treebanks. We apply our method to the English dependency corpus – conversion of the Penn Treebank to the dependency format done by Richard Johansson and Mihai Surdeanu [2] for the CoNLL shared tasks [3]. This is probably the most used dependency corpus, since English is the most popular language among the researchers. Still we are able to find a considerable amount of errors in it. Additionally, we compare our method with an interesting approach developed by a different group of researchers (see section 2). They are able to find a similar number of errors in different corpora, however, as our investigation shows, the overlap between our results is quite small and the approaches are rather complementary.

2 Related Work

Surprisingly, we were not able to find a lot of work on the topic of error detection in treebanks. Some organisers of shared tasks usually try to guarantee a certain quality of the used data, but the quality control is usually performed manually. E.g. in the already mentioned CoNLL task the organisers analysed a large amount of dependency treebanks for different languages [4], described problems they have encountered and forwarded them to the developers of the corresponding corpora. The only work, that we were able to find, which involved automatic quality control, was done by the already mentioned group around Detmar Meurers. This work includes numerous publications concerning finding errors in phrase structures [5] as well as in dependency treebanks [6]. The approach is based on the concept of "variation detection", first introduced in [7]. Additionally, [5] presents a good method for evaluating the automatic error detection. We will perform a similar evaluation for the precision of our approach.

3 Variation Detection

We will compare our outcomes with the results that can be found with the approach of "variation detection" proposed by Meurers et al. For space reasons, we will not be able to elaborately present this method and advise to read the referred work, However, we think that we should at least briefly explain its idea.

The idea behind "variation detection" is to find strings, which occur multiple times in the corpus, but which have varying annotations. This can obviously have only two reasons: either the strings are ambiguous and can have different structures, depending on the meaning, or the annotation is erroneous in at least one of the cases. The idea can be adapted to dependency structures as well, by analysing the possible dependency relations between same words. Again different dependencies can be either the result of ambiguity or errors.

4 Automatic Detection of Errors

We propose a different approach. We take the English dependency treebank and train models with two different state of the art parsers: the graph-based MSTParser [9] and the transition-based MaltParser [10]. We then parse the data, which we have used for training, with both parsers. The idea behind this step is that we basically try to reproduce the gold standard, since parsing the data seen during the training is very easy (a similar idea in the area of POS tagging is very broadly described in [8]). Indeed both parsers achieve accuracies between 98% and 99% UAS (Unlabeled Attachment Score), which is defined as the proportion of correctly identified dependency relations. The reason why the parsers are not able to achieve 100% is on the one hand the fact that some of the phenomena are too rare and are not captured by their models. On the other hand, in many other cases parsers do make correct predictions, but the gold standard they are evaluated against is wrong.

We have investigated the latter case, namely when both parsers predict dependencies different from the gold standard (we do not consider the correctness of the dependency label). Since MSTParser and MaltParser are based on completely different parsing approaches they also tend to make different mistakes [11]. Additionally, considering the accuracies of 98-99% the chance that both parsers, which have different foundations, make an erroneous decision simultaneously is very small and therefore these cases are the most likely candidates when looking for errors.

5 Automatic Correction of Errors

In this section we propose our algorithm for automatic correction of errors, which consists out of the following steps:

1. Automatic detection of error candidates, i.e. cases where two parsers deliver results different to gold-standard.
2. Substitution of the annotation of the error candidates by the annotation proposed by one of the parsers (in our case MSTParser).
3. Parse of the modified corpus with a third parser (MDParser).
4. Evaluation of the results.
5. The modifications are only kept for those cases when the modified annotation is identical with the one predicted by the third parser and undone in other cases.

For the English dependency treebank we have identified 6743 error candidates, which is about 0.7% of all tokens in the corpus.

The third dependency parser, which is used is MDParser[1] - a fast transition-based parser. We subsitute the gold standard by MSTParser and not MaltParser in order not to give an advantage to a parser with similar basics (both MDParser and MDParser are transition-based).

During this experiment we have found out that the result of MDParser significantly improves: it is able to correctly recgonize 3535 more dependencies than before the substitution of the gold standard. 2077 annotations remain wrong independently of the changes in the gold standard. 1131 of the relations become wrong with the changed gold standard, whereas they were correct with the old unchanged version. We then undo the changes to the gold standard when the wrong cases remained wrong and when the correct cases became wrong. We suggest that the 3535 dependencies which became correct after the change in gold standard are

[1] http://mdparser.sb.dfki.de/

errors, since a) two state of the art parsers deliver a result which differs from the gold standard and b) a third parser confirms that by delivering exactly the same result as the proposed change. However, the exact precision of the approach can probably be computed only by manual investigation of all corrected dependencies.

6 Estimating the Overall Number Of Errors

The previous section tries to evaluate the precision of the approach for the identified error candidates. However, it remains unclear how many of the errors are found and how many errors can be still expected in the corpus. Therefore in this section we will describe our attempt to evaluate the recall of the proposed method.

In order to estimate the percentage of errors, which can be found with our method, we have designed the following experiment. We have taken sentences of different lengths from the corpus and provided them with a “gold standard” annotation which was completely (=100%) erroneous. We have achieved that by substituting the original annotation by the annotation of a different sentence of the same length from the corpus, which did not contain dependency edges which would overlap with the original annotation. E.g consider the following sentence in the (slightly simplified) CoNLL format:

1	Not	RB	6	SBJ
2	all	PDT	1	NMOD
3	those	DT	1	NMOD
4	who	WP	5	SBJ
5	wrote	VBD	1	NMOD
6	oppose	VBP	0	ROOT
7	the	DT	8	NMOD
8	changes	NNS	6	OBJ
9	.	.	6	P

We would substitute its annotation by an annotation chosen from a different sentence of the same length:

1	Not	RB	3	SBJ
2	all	PDT	3	NMOD
3	those	DT	0	NMOD
4	who	WP	3	SBJ
5	wrote	VBD	4	NMOD
6	oppose	VBP	5	ROOT
7	the	DT	6	NMOD
8	changes	NNS	7	OBJ
9	.	.	3	P

This way we know that we have introduced a well-formed dependency tree (since its annotation belonged to a different tree before) to the corpus and the exact number of errors (since randomly correct dependencies are impossible). In case of our example 9 errors are introduced to the corpus.

In our experiment we have introduced sentences of different lengths with overall 1350 tokens. We have then retrained the models for MSTParser and MaltParser and have applied our methodology to the data with these errors. We have then counted how many of these 1350 errors could be found. Our result is that 619 tokens (45.9%) were different from the erroneous gold-standard. That means that despite the fact that the training data contained some incorrectly annotated tokens, the parsers were able to annotate them differently. Therefore we suggest that the recall of our method is close to the value of 0.459. However, of course we do not know whether the randomly introduced errors in our experiment are similar to those which occur in real treebanks.

7 Comparison with Variation Detection

The interesting question which naturally arises at this point is whether the errors we find are the same as those found by the method of variation detection. Therefore we have performed the following experiment: We have counted the numbers of occurrences for the dependencies $B \rightarrow A$ (the word B is the head of the word A) and $C \rightarrow A$ (the word C is the head of the word A), where $B \rightarrow A$ is the dependency proposed by the parsers and $C \rightarrow A$ is the dependency proposed by the gold standard. In order for variation detection to be applicable the frequency counts for both relations must be available and the counts for the dependency proposed by the parsers should ideally greatly outweigh the frequency of the gold standard, which would be a great indication of an error. For the 3535 dependencies that we classify as errors the variation detection method works only 934 times (39.5%). These are the cases when the gold standard is obviously wrong and occurs only few times, most often - once, whereas the parsers propose much more frequent dependencies. In all other cases the counts suggest that the variation detection would not work, since both dependencies have frequent counts or the correct dependency is even outweighed by the incorrect one.

8 Examples

We will provide some of the example errors, which we are able to find with our approach. Therefore we will provide the sentence strings and briefly compare the gold standard dependency annotation of a certain dependency within these sentences.

Together, the two stocks wreaked havoc among takeover stock traders, and caused a 7.3% drop in the DOW Jones Transportation Average, second in size only to ***the stock-market crash*** *of Oct. 19 1987.*

In this sentence the gold standard suggests the dependency relation $market \rightarrow the$, whereas the parsers correctly recognise the dependency $crash \rightarrow the$. Both dependencies have very high counts and therefore the variation detection would not work well in this scenario.

Actually, it ***was*** *down only a few* ***points at*** *the time.*

In this sentence the gold standard suggests $points \rightarrow at$, whereas the parsers predict $was \rightarrow at$. The gold standard suggestion occurs only once whereas the temporal dependency $was \rightarrow at$ occurs 11 times in the corpus. This is an example of an error which could be found with the variation detection as well.

Last October, Mr. Paul paid out $12 million of CenTrust's cash – plus ***a $1.2 million commission*** *– for "Portrait of a Man as Mars".*

In this sentence the gold standard suggests the dependency relation $\$ \rightarrow a$, whereas the parsers correctly recognise the dependency $commission \rightarrow a$. The interesting fact is that the relation $\$ \rightarrow a$ is actually much more frequent than $commission \rightarrow a$, e.g. as in the sentence *he cought up an additional $1 billion or so*. ($\$ \rightarrow an$) So the variation detection alone would not suffice in this case.

9 Conclusion

The quality of treebanks is of an extreme importance for the community. Nevertheless, errors can be found even in the most popular and widely-used resources. In this paper we have presented an approach for automatic detection and correction of errors and compared it to the only other work we have found in this field. Our results show that both approaches are rather complementary and find different types of errors.

We have only analysed the errors in the head-modifier annotation of the dependency relations in the English dependency treebank. However, the same methodology can easily be applied to detect irregularities in any kind of annotations, e.g. labels, POS tags etc. In fact, in the area of POS tagging a similar strategy of using the same data for training and testing in order to detect inconsistencies has proven to be very efficient [8]. However, the method lacked means for automatic correction of the possibly inconsistent annotations. Additionally, the method off course can as well be applied to different corpora in different languages.

Our method has a very high precision, even though we could not compute the exact value, since it would require an expert to go through a large number of cases. It is even more difficult to estimate the recall of our method, since the overall number of errors in a corpus is unknown. We have described an experiment which to our mind is a good attempt to evaluate the recall of our approach. On the one hand the recall we have achieved in this experiment is rather low (0.459), which means that our method would definitely not guarantee to find all errors in a corpus. On the other hand it has a very high precision and thus is in any case beneficial, since the quality of the treebanks increases with the removal of errors. Additionally, the low recall suggests that treebanks contain an even larger number of errors, which could not be found. The overall number of errors thus seems to be over 1% of the total size of a corpus, which is expected to be of a very high quality. A fact that one has to be aware of when working with annotated resources and which we would like to emphasize with our paper.

10 Acknowledgements

The presented work was partially supported by a grant from the German Federal Ministry of Economics and Technology (BMWi) to the DFKI Theseus project TechWatch—Ordo (FKZ: 01M-Q07016).

References

[1] Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz , 1993. *Building a Large Annotated Corpus of English: The Penn Treebank*. In Computational Lingustics, vol. 19, pp. 313-330.

[2] Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluis Marquez and Joakim Nivre. *The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies*. In Proceedings of the 12th Conference on Computational Natural Language Learning (CoNLL-2008), 2008

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

[4] Sabine Buchholz and Darren Green, 2006. *Quality control of treebanks: documenting, converting, patching*. In LREC 2006 workshop on Quality assurance and quality measurement for language and speech resources.

[5] Markus Dickinson and W. Detmar Meurers, 2005. *Prune Diseased Branches to Get Healthy Trees! How to Find Erroneous Local Trees in a Treebank and Why It Matters*. In Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories, pp. 41—52

[6] Adriane Boyd, Markus Dickinson and Detmar Meurers, 2008. *On Detecting Errors in Dependency Treebanks*. In Research on Language and Computation, vol. 6, pp. 113-137.

[7] Markus Dickinson and Detmar Meurers, 2003. *Detecting inconsistencies in treebanks*. In Proceedings of TLT 2003

[8] van Halteren, H. (2000). T*he detection of inconsistency in manually tagged text*. In A. Abeillé, T. Brants, and H. Uszkoreit (Eds.), Proceedings of the Second Workshop on Linguistically Interpreted Corpora (LINC-00), Luxembourg.

[9 R. McDonald, F. Pereira, K. Ribarov, and J. Haji˘c . 2005. *Non-projective Dependency Parsing using Spanning Tree Algorithms.* In Proc. of HLT/EMNLP 2005.

[10] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gulsen Eryigit, Sandra Kubler, Svetoslav Marinov and Erwin Marsi. 2007. *MaltParser: A Language-Independent System for Data-Driven Dependency Parsing,* Natural Language Engineering Journal, 13, pp. 99-135.

[11] Joakim Nivre and Ryan McDonald, 2008. *Integrating GraphBased and Transition-Based Dependency Parsers*. In Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies.