Ensemble Models for Dependency Parsing: Cheap and Good?

Mihai Surdeanu and Christopher D. Manning
Computer Science Department
Stanford University, Stanford, CA 94305
{mihais,manning}@stanford.edu

Abstract

Previous work on dependency parsing used various kinds of combination models but a systematic analysis and comparison of these approaches is lacking. In this paper we implemented such a study for English dependency parsing and find several non-obvious facts: (a) the diversity of base parsers is more important than complex models for learning (e.g., stacking, supervised meta-classification), (b) approximate, linear-time re-parsing algorithms guarantee well-formed dependency trees without significant performance loss, and (c) the simplest scoring model for re-parsing (unweighted voting) performs essentially as well as other more complex models. This study proves that fast and accurate ensemble parsers can be built with minimal effort.

1 Introduction

Several ensemble models have been proposed for the parsing of syntactic dependencies. These approaches can generally be classified in two categories: models that integrate base parsers at learning time, e.g., using stacking (Nivre and McDonald, 2008; Attardi and Dell'Orletta, 2009), and approaches that combine independently-trained models only at parsing time (Sagae and Lavie, 2006; Hall et al., 2007; Attardi and Dell'Orletta, 2009). In the latter case, the correctness of the final dependency tree is ensured by: (a) selecting entire trees proposed by the base parsers (Henderson and Brill, 1999); or (b) re-parsing the pool of dependencies proposed by the base models (Sagae and Lavie, 2006). The latter approach was shown to perform better for constituent parsing (Henderson and Brill, 1999).

While all these models achieved good performance, the previous work has left several questions unanswered. Here we answer the following questions, in the context of English dependency parsing:

1. When combining models at parsing time, what is the best scoring model for candidate dependencies during re-parsing? Can a meta classifier improve over unsupervised voting?
2. Are (potentially-expensive) re-parsing strategies justified for English? What percentage of trees are not well-formed if one switches to a light word-by-word voting scheme?
3. How important is the integration of base parsers at learning time?
4. How do ensemble models compare against state-of-the-art supervised parsers?

	Devel	In domain		Out of domain	
	LAS	LAS	UAS	LAS	UAS
MST	85.36	87.07	89.95	80.48	86.08
Malt$^{\rightarrow}_{AE}$	84.24	85.96	88.64	78.74	84.18
Malt$^{\rightarrow}_{CN}$	83.75	85.61	88.14	78.55	83.68
Malt$^{\rightarrow}_{AS}$	83.74	85.36	88.06	77.23	82.39
Malt$^{\leftarrow}_{AS}$	82.43	83.90	86.70	76.69	82.57
Malt$^{\leftarrow}_{CN}$	81.75	83.53	86.17	77.29	83.02
Malt$^{\leftarrow}_{AE}$	80.76	82.51	85.35	76.18	82.02

Table 1: Labeled attachment scores (LAS) and unlabeled attachment scores (UAS) for the base models. The parsers are listed in descending order of LAS in the development partition.

2 Setup

In our experiments we used the syntactic dependencies from the CoNLL 2008 shared task corpus (Surdeanu et al., 2008).

We used seven base parsing models in this paper: six are variants of the Malt parser[1] and the seventh is the projective version of MSTParser that uses only first-order features[2] (or MST for short). The six Malt

[1] http://maltparser.org/

[2] http://sourceforge.net/projects/mstparser/

Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL, pages 649–652,
Los Angeles, California, June 2010.

# of parsers	Unweighted		Weighted by POS of modifier		Weighted by label of dependency		Weighted by dependency length		Weighted by sentence length	
	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS
3	86.03	89.44	86.02	89.43	85.53	88.97	85.85	89.23	86.03	89.45
4	86.79	90.14	86.68	90.07	86.38	89.78	86.46	89.79	86.84	90.18
5	86.98	90.33	86.95	90.30	86.60	90.06	86.87	90.22	86.86	90.22
6	87.14	**90.51**	**87.17**	90.50	86.74	90.22	86.91	90.23	87.04	90.37
7	86.81	90.21	86.82	90.21	86.50	90.01	86.71	90.08	86.80	90.19

Table 2: Scores of unsupervised combination models using different voting strategies. The combined trees are assembled using a word-by-word voting scheme.

parser variants are built by varying the parsing algorithm (we used three parsing models: Nivre's arc-eager (AE), Nivre's arc-standard (AS), and Covington's non-projective model (CN)), and the parsing direction (left to right ($\rightarrow$) or right to left ($\leftarrow$)), similar to (Hall et al., 2007). The parameters of the Malt models were set to the values reported in (Hall et al., 2007). The MST parser was used with the default configuration. Table 1 shows the performance of these models in the development and test partitions.

3 Experiments

3.1 On scoring models for parser combination

The most common approach for combining independently-trained models at parsing time is to assign each candidate dependency a score based on the number of votes it received from the base parsers. Considering that parsers specialize in different phenomena, these votes can be weighted by different criteria. To understand the importance of such weighting strategies we compare several voting approaches in Table 2: in the "unweighted" strategy all votes have the same weight; in all other strategies each vote is assigned a value equal to the accuracy of the given parser in the particular instance of the context considered, e.g., in the "weighted by POS of modifier" model we use the accuracies of the base models for each possible part-of-speech (POS) tag of a modifier token. In the table we show results as more base parsers are added to the ensemble (we add parsers in the order given by Table 1). The results in Table 2 indicate that weighting strategies do not have an important contribution to overall performance. The only approach that outperforms the LAS score of the unweighted voting model is the model that weighs parsers by their accuracy for a given modifier POS tag, but the improvement is marginal. On the other hand, the number of base parsers in the ensemble pool is crucial: performance generally continues to improve as more base parsers are considered. The best ensemble uses 6 out of the 7 base parsers.[3]

	POS(m)	POS(m) $\times$ POS(h)	length(s)
MST	38	56	26
Malt$^{\rightarrow}_{AE}$	0	6	6
Malt$^{\rightarrow}_{CN}$	0	14	7
Malt$^{\rightarrow}_{AS}$	0	61	0
Malt$^{\leftarrow}_{AS}$	0	0	3
Malt$^{\leftarrow}_{CN}$	0	9	0
Malt$^{\leftarrow}_{AE}$	0	0	0

Table 3: Total number of minority dependencies with precision larger than 50%, for different base parsers and most representative features (m - modifier, h - head, s - sentence). These are counts of tokens, computed in the development corpus of 33,368 dependencies.

It is often argued that the best way to re-score candidate dependencies is not through voting but rather through a meta-classifier that selects candidate dependencies based on their likelihood of belonging to the correct tree. Unlike voting, a meta-classifier can combine evidence from multiple contexts (such as the ones listed in Table 2). However, in our experiments such a meta-classifier[4] did not offer any gains over the much simpler unweighted voting strategy. We explain these results as follows: the meta-classifier can potentially help only when it proposes dependencies that disagree with the majority vote. We call such dependencies *minority dependencies*.[5] For a given parser and context instance (e.g., a modifier POS), we define precision of minority dependencies as the ratio of minority dependencies in this group that are correct. Obviously, a

[3] We drew similar conclusions when we replaced voting with the re-parsing algorithms from the next sub-section.

[4] We implemented a L2-regularized logistic regression classifier using as features: identifiers of the base models, POS tags of head and modifier, labels of dependencies, length of dependencies, length of sentence, and combinations of the above.

[5] (Henderson and Brill, 1999) used a similar framework in the context of constituent parsing and only three base parsers.

group of minority dependencies provides beneficial signal only if its precision is larger than 50%. Table 3 lists the total number of minority dependencies in groups with precision larger than 50% for all our base parsers and the most representative features. The table shows that the number of minority dependencies with useful signal is extremely low. All in all, it accounts for less than 0.7% of all dependencies in the development corpus.

3.2 On re-parsing algorithms

To guarantee that the resulting dependency tree is well-formed, most previous work used the dynamic programming algorithm of Eisner (1996) for re-parsing (Sagae and Lavie, 2006; Hall et al., 2007).[6] However, it is not clear that this step is necessary. In other words, how many sentences are not well-formed if one uses a simple word-by-word voting scheme? To answer this, we analyzed the output of our best word-by-word voting scheme (six base parsers weighted by the POS of the modifier). The results for both in-domain and out-of-domain testing corpora are listed in Table 4. The table shows that the percentage of badly-formed trees is relatively large: almost 10% out of domain. This indicates that the focus on algorithms that guarantee well-formed trees is justified.

However, it is not clear how the Eisner algorithm, which has runtime complexity of $O(n^3)$ (n – number of tokens per sentence), compares against approximate re-parsing algorithms that have lower runtime complexity. One such algorithm was proposed by Attardi and Dell'Orletta (2009). The algorithm, which has a runtime complexity of $O(n)$, builds dependency trees using a greedy top-down strategy, i.e., it starts by selecting the highest-scoring root node, then the highest-scoring children, etc. We compare these algorithms against the word-by-word voting scheme in Table 5.[7] The results show that both algorithms pay a small penalty for guaranteeing well-formed trees. This performance drop is statistically significant out of domain. On the other hand, the difference between the Eisner and Attardi algorithms is not statistically significant out of domain.

[6] We focus on projective parsing algorithms because 99.6% of dependencies in our data are projective (Surdeanu et al., 2008).

[7] Statistical significance was performed using Dan Bikel randomized parsing evaluation comparator at 95% confidence.

	In domain	Out of domain
Zero roots	0.83%	0.70%
Multiple roots	3.37%	6.11%
Cycles	4.29%	4.23%
Total	7.46%	9.64%

Table 4: Percentage of badly-formed dependency trees when base parsers are combined using a word-by-word voting scheme. The different error classes do not sum up to the listed total because the errors are not mutually exclusive.

	In domain		Out of domain	
	LAS	UAS	LAS	UAS
Word by word	88.89	91.52	82.13*	87.51*
Eisner	88.83*	91.47*	81.99	87.32
Attardi	88.70	91.34	81.82	87.29

Table 5: Scores of different combination schemes. * indicates that a model is significantly different than the next lower ranked model.

This experiment proves that approximate re-parsing algorithms are a better choice for practical purposes, i.e., ensemble parsing in domains different from the training material of the base models.

3.3 On parser integration at learning time

Recent work has shown that the combination of base parsers at learning time, e.g., through stacking, yields considerable benefits (Nivre and McDonald, 2008; Attardi and Dell'Orletta, 2009). However, it is unclear how these approaches compare against the simpler ensemble models, which combine parsers only at runtime. To enable such a comparison, we reimplemented the best stacking model from (Nivre and McDonald, 2008) – MST$_{Malt}$ – which trains a variant of the MSTParser that uses additional features extracted from the output of a Malt parser.

In Table 6, we compare this stacking approach against four variants of our ensemble models. The superscript in the ensemble name indicates the runtime complexity of the model ($O(n^3)$ or $O(n)$). The cubic-time models use all base parsers from Table 1 and the Eisner algorithm for re-parsing. The linear-time models use only Malt-based parsers and the Attardi algorithm for re-parsing. The subscript in the model names indicates the percentage of available base parsers used, e.g., ensemble$^{3}_{50\%}$ uses only the first three parsers from Table 1. These results show that MST$_{Malt}$ is statistically equivalent to an ensemble that uses MST and two Malt variants, and both our ensemble$_{100\%}$ models are significantly better than MST$_{Malt}$. While this comparison is somewhat unfair (MST$_{Malt}$ uses two base models, whereas our ensemble models use at least three) it

	In domain		Out of domain	
	LAS	UAS	LAS	UAS
ensemble$^{3}_{100\%}$	88.83*	91.47*	81.99*	87.32*
ensemble$^{1}_{100\%}$	88.01*	90.76*	80.78	86.55
ensemble$^{3}_{50\%}$	87.45	90.17	81.12	86.62
MST$_{Malt}$	87.45*	90.22*	80.25*	85.90*
ensemble$^{1}_{50\%}$	86.74	89.62	79.44	85.54

Table 6: Comparison of different combination strategies.

	In domain		Out of domain	
	LAS	UAS	LAS	UAS
CoNLL 2008, #1	90.13*	92.45*	82.81*	88.19*
ensemble$^{3}_{100\%}$	88.83*	91.47*	81.99*	87.32*
CoNLL 2008, #2	88.14	90.78	80.80	86.12
ensemble$^{1}_{100\%}$	88.01	90.76	80.78	86.55

Table 7: Comparison with state of the art parsers.

does illustrate that the advantages gained from combining parsers at learning time can be easily surpassed by runtime combination models that have access to more base parsers. Considering that variants of shift-reduce parsers can be generated with minimal effort (e.g., by varying the parsing direction, learning algorithms, etc.) and combining models at runtime is simpler than combining them at learning time, we argue that runtime parser combination is a more attractive approach.

3.4 Comparison with the state of the art

In Table 7 we compare our best ensemble models against the top two systems of the CoNLL-2008 shared task evaluation. The table indicates that our best ensemble model ranks second, outperforming significantly 19 other systems. The only model performing better than our ensemble is a parser that uses higher-order features and has a higher runtime complexity ($O(n^4)$) (Johansson and Nugues, 2008). While this is certainly proof of the importance of higher-order features, it also highlights a pragmatic conclusion: in out-of-domain corpora, an ensemble of models that use only first-order features achieves performance that is within 1% LAS of much more complex models.

4 Conclusions

This study unearthed several non-intuitive yet important observations about ensemble models for dependency parsing. First, we showed that the diversity of base parsers is more important than complex learning models for parser combination, i.e., (a) ensemble models that combine several base parsers at runtime performs significantly better than a state-of-the-art model that combines two parsers at learning time, and (b) meta-classification does not outperform unsupervised voting schemes for the re-parsing of candidate dependencies when six base models are available. Second, we showed that well-formed dependency trees can be guaranteed without significant performance loss by linear-time approximate re-parsing algorithms. And lastly, our analysis indicates that unweighted voting performs as well as weighted voting for the re-parsing of candidate dependencies. Considering that different base models are easy to generate, this work proves that ensemble parsers that are both accurate and fast can be rapidly developed with minimal effort.

Acknowledgments

This material is based upon work supported by the Air Force Research Laboratory (AFRL) under prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the Air Force Research Laboratory (AFRL).

We thank Johan Hall, Joakim Nivre, Ryan McDonald, and Giuseppe Attardi for their help in understanding details of their models.

References

G. Attardi and F. Dell'Orletta. 2009. Reverse revision and linear tree combination for dependency parsing. In *Proc. of NAACL-HLT*.

J. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING*.

J. Hall, J. Nilsson, J. Nivre, G. Eryigit, B. Megyesi, M. Nilsson, and M. Saers. 2007. Single malt or blended? A study in multilingual parser optimization. In *Proc. of CoNLL Shared Task*.

J. C. Henderson and E. Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. In *Proc. of EMNLP*.

R. Johansson and P. Nugues. 2008. Dependency-based syntactic semantic analysis with PropBank and NomBank. In *Proc. of CoNLL Shared Task*.

J. Nivre and R. McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proc. of ACL*.

K. Sagae and A. Lavie. 2006. Parser combination by reparsing. In *Proc. of NAACL-HLT*.

M. Surdeanu, R. Johansson, A. Meyers, L. Marquez, and J. Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proc. of CoNLL*.