Voting and Stacking in Data-Driven Dependency Parsing

Mark Fishel
University of Tartu
Tartu, Estonia
fishel@ut.ee

Joakim Nivre
Uppsala University
Uppsala, Sweden
joakim.nivre@lingfil.uu.se

Abstract

We compare the techniques of voting and stacking for system combination in data-driven dependency parsing, using a set of eight different transition-based parsers as component systems. Experimental results show that both methods lead to significant improvements over the best component system, and that voting gives the highest overall accuracy. We also investigate different weighting schemes for voting.

1 Introduction

System combination is a general technique that can be used to boost accuracy in natural language processing tasks. By combining several models for performing the same task, we can exploit the unique advantage of each model and reduce some of the random errors. In this paper, we study two techniques for combining data-driven dependency parsers: *voting* and *stacking*.

In parser combination by *voting*, the outputs of (at least three) independent parsers are combined to produce an analysis supported by a majority of component systems. This technique was first proposed by Zeman and Žabokrtský (2005) and further refined by Sagae and Lavie (2006), who showed that it could be construed as a special form of spanning tree parsing. In parser combination by *stacking*, the outputs of one or more parsers are used as features for a data-driven parser that can learn from the predictions of other models. Parser stacking was recently used by Nivre and McDonald (2008) to advance the state of the art on the multilingual test sets from the CoNLL-X shared task (Buchholz and Marsi, 2006).

We describe a series of experiments, where we first try to optimize the voting strategy, by investigating different schemes for assigning weights to the votes of different systems. We then compare voting to the alternative method of stacking. The paper is organized as follows. Section 2 describes the tools, resources and methods, common to all experiments, as well as the component parsers, used for voting and stacking. Optimizations of voting are introduced and evaluated in Section 3, and stacking is treated in Section 4. The paper is concluded with Section 5.

2 Common Resources and Methodology

We used the corpora from the closed part of the CoNLL 2008 shared task (Surdeanu et al., 2008), including a training corpus (39,279 sentences), a development corpus (1,334 sentences), an in-domain test corpus (labeled WSJ, 2,399 sentences) and an out-of-domain test corpus (labeled Brown, 425 sentences). The available features included word forms, lemmas, and part-of-speech. A more detailed description of the corpora can be found in Surdeanu et al. (2008).

In all voting experiments the training corpus was used to train the component systems and the development corpus was used to learn weights. In the case of stacking, the development corpus was too small to train the joint parsing system. Thus 4-fold cross-validation on the training set was used with the common systems, and the joint system was trained on the resulting training set.

All results are evaluated using the *labeled attachment score*, which is the percentage of tokens with correctly determined heads and dependency relations in the test corpus. Intermediate models are evaluated on the WSJ testing corpus, whereas the final scores are presented for both WSJ and Brown.

All component parsers (as well as the stacking parser) were trained using MaltParser (Nivre et al., 2006), a data-driven dependency parser generator that implements two parsing algorithms: the shift-reduce algorithm proposed by Nivre (2003),

Kristiina Jokinen and Eckhard Bick (Eds.)
NODALIDA 2009 Conference Proceedings, pp. 219–222

in an arc-eager and an arc-standard variant (*Nivre-Eager* and *Nivre-Std*), and the incremental parsing algorithm first described in Covington (2001), in a projective and a non-projective variant (*Cov-Proj* and *Cov-NonProj*).

We used eight component parsers, defined by the four algorithm variants times two directions (forward and reverse), which is the same setup as in Samuelsson et al. (2008). Feature models and parameter settings were taken from Hall et al. (2007). The scores of the eight parsers on the two test corpora are presented in Table 1. The highest score is obtained with Nivre-Eager forward, which may be partly due to the fact that this is the setup used for feature selection and parameter tuning by Hall et al. (2007).

	WSJ	**Brown**
Nivre-Eager forward	86.47%	78.76%
Nivre-Eager reverse	82.94%	76.29%
Nivre-Std forward	84.87%	76.34%
Nivre-Std reverse	84.52%	76.88%
Cov-Proj forward	85.14%	77.61%
Cov-Proj reverse	83.41%	76.59%
Cov-NonProj forward	85.75%	78.09%
Cov-NonProj reverse	83.61%	77.23%

Table 1: Labeled attachment score of the component parsers on WSJ and Brown.

3 Voting

System combination by voting was first proposed for dependency parsing by Zeman and Žabokrtský (2005). Since the task of a dependency parser is to select one head and one dependency relation for each input word, letting component systems vote is a straightforward strategy for combining their predictions. One problem is that using the majority vote for each word may not result in a valid dependency tree – it may result in a graph with cycles, for example – but Sagae and Lavie (2006) showed that this problem can be solved using the maximum spanning tree algorithm previously proposed for dependency parsing by McDonald et al. (2005). If all dependency arcs proposed by some parser are stored in a graph and weighted by their number of votes, then extracting the maximum spanning tree (MST) from this graph yields the optimal dependency tree.

Sagae and Lavie (2006) also showed that accuracy can be further improved if votes are weighted by the accuracy of the component parser on all arcs where the dependent token has the same part of speech. This weighting scheme, which we will refer to as the default model, was later used by Hall et al. (2007) to achieve the best overall score in the CoNLL 2007 shared task by combining six different parsers. Samuelsson et al. (2008) used a variation on the default model, where weights are first set according to accuracy but are then iteratively updated using the following simple principle: at each iteration the MST is compared to the reference parse, after which all weights of correct arcs are given a small increase and all incorrect ones a small decrease. This technique resulted in minor score improvements over the default model.

In order to obtain a baseline, the default model was applied to the eight component systems, resulting in a score of 88.14%, which is a considerable improvement over the best component system (1.67% absolute score improvement, 12.35% error reduction). In addition to the baseline an upper-bound was computed by using the reference parse as an ideal oracle parse, giving correct arcs a weight of 1 and incorrect ones a weight of 0. The resulting upper-bound score is 93.84%.

The first optimization attempt consisted of using other categories or category combinations than the part of speech of the dependent token (POS) to group the individual weights. As a result, three features that improved scores were found: the dependency relation of the dependent token (DEPREL) (88.28%), the part of speech of its head (H-POS) (88.27%), and the dependency relation of its head (H-DEPREL) (88.30%). However, all improvements are rather marginal.

In the next step we tested composite categories, consisting of subsets of the three successful features and the original part of speech, getting improvements for the following combinations: POS, DEPREL (88.48%), POS, H-POS (88.40%), and POS, H-DEPREL (88.45%). We also tried replacing the original part-of-speech tags with more general categories, obtained by taking the first two characters of the original tags (which are two or three characters long). This resulted in 32 tags (instead of 47) and generally improved scores, but again only marginally.

All results from our weight grouping experiments can be found in Table 2. In general, it can

	POS WSJ	PO WSJ	PO Brown
POS (default)	88.14%	88.15%	80.64%
DEPREL	88.28%	-	80.88%
H-POS	88.27%	88.29%	80.84%
H-DEPREL	88.30%	-	80.77%
POS, DEPREL	88.48%	88.49%	80.96%
POS, H-POS	88.40%	88.50%	81.14%
POS, H-DEPREL	88.45%	88.50%	81.05%
DEPREL, H-POS	88.24%	88.28%	80.92%
DEPREL, H-DEPREL	88.26%	-	80.82%
H-DEPREL, H-POS	88.26%	88.27%	80.81%
All but H-DEPREL	88.18%	88.42%	81.05%
All but H-POS	88.21%	88.30%	80.91%
All but DEPREL	88.14%	88.38%	81.12%
All but POS	87.99%	88.11%	80.67%
All four	87.74%	88.15%	80.67%
Upper bound	93.84%	-	88.66%

Table 2: Labeled attachment score for voting systems with weights grouped by different combinations of the token part of speech (POS), the first two letters of the part-of-speech tag (PO), the token dependency relation (DEPREL), the head part of speech (H-POS), and the head dependency relation (H-DEPREL).

be concluded that the part-of-speech of the dependent token, used in the default model, is an important feature for grouping weights, but the system can benefit from combining it with other features of dependency arcs.

The second optimization attempt was to apply gradient descent learning to the problem of finding optimal weights. We defined the error function as follows:

$$\mathcal{E} = \sum_i (w_i^{ref} - w_i^{hyp})^2$$

where w_i^{hyp} are the current weights and w_i^{ref} are the golden reference weights, which equal 1 if the arc is present in the reference parse and 0 otherwise. Thus, minimizing the error function causes the weights to get closer to the golden reference, and the weight of the corresponding category and system is "rewarded" for each correct guess.

Gradient descent learning gave results on a par with the default model but never exceeded them by more than 0.05% despite tweaking the learning rate, replacing categories or switching between initializing the weights to the default model or to random values. In our opinion, this strongly indicates that the default model is either optimal or very close to optimal.

4 Stacking

A completely different way of using the results of several different systems is to include their outputs as input features to a joint system. This is known as *stacking* and has the potential advantage that it allows the joint system to learn from the predictions of the component parsers, as opposed to merely combining the predictions. Stacking for dependency parsing was used by Nivre and McDonald (2008) to combine MaltParser (Nivre et al., 2006) and MSTParser (McDonald et al., 2005). The results showed significant improvements in accuracy when using either of the parsers to generate features for the other, with the largest improvement when MSTParser could learn from features generated by MaltParser.

In our experiments, this approach was tested with the eight component parsers described in Section 2 as input systems. The joint system was essentially the same as the best performing component parser (*Nivre-Eager*) but trained on features that include both the original feature set from Hall et al. (2007) and the new features from the input system outputs. The latter included the hypothesized incoming arcs and dependency relations of the token on top of the stack and the next input token. The joint system achieved a labeled attach-

ment score of 87.67% (1.20% absolute score improvement, 8.87% error reduction).

We then tried removing some of the new feature groups, for example, only using the arc features, or only the dependency relation features. The best combination was achieved by excluding the dependency relation features of the token on the top of the stack (87.76%). The final results for the best models are presented in Table 3. Although some models achieved improvements over the best component system, all of them remained below the best voting system, described above in Section 3.

	WSJ	**Brown**
Baseline	86.47%	78.76%
Stacking, all features	87.67%	80.05%
Stacking, all but input DEPREL	87.15%	79.77%
Stacking, all but stack DEPREL	87.76%	79.83%
Stacking, all but arcs	87.73%	80.07%
Stacking, only input DEPREL	87.71%	79.92%

Table 3: Labeled attachment score of the stacking parsers.

5 Conclusions

This paper focused on the voting technique, which uses the output of many dependency parsers to combine their individual advantages and compute a joint parse. We conducted several experiments, empirically evaluating some adjustments to the technique, and also compared it to the alternative technique of stacking.

The experimental results first of all confirmed that voting may result in considerable quality improvements over their component parser systems. Our attempts to find better ways of grouping arcs when assigning weights showed marginal improvements, in particular when introducing more general part-of-speech categories, while the experiments on replacing the default weighting scheme with gradient descent learning mainly showed that the default model is close to optimal in itself.

The experiments on stacking also showed improvements over its baseline but generally resulted in lower scores than all voting systems. We believe that better results can be achieved by thoroughly selecting the features of the joint parser, but it is also possible that stacking works better when the differences between the input parsers and the joint parser are greater. For example, whereas all our parsers were instantiations of the transition-based approach implemented in MaltParser, Nivre and McDonald (2008) combined one transition-based parser and one graph-based parser, models that have different characteristic error distributions.

References

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

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proc. of the Annual ACM Southeast Conference*, pages 95–102.

Johan Hall, Jens Nilsson, Joakim Nivre, Gülsen Eryigit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single malt or blended? a study in multilingual parser optimization. In *Proc. of CoNLL Shared Task*, pages 933–939.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of HLT/EMNLP*, pages 523–530.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proc. of ACL*, pages 950–958.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. Maltparser: A data-driven parser-generator for dependency parsing. In *Proc. of LREC*, pages 2216–2219.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proc. of IWPT*, pages 149–160.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proc. of NAACL*, pages 129–132.

Yvonne Samuelsson, Oscar Täckström, Sumithra Velupillai, Johan Eklund, Mark Fishel, and Markus Saers. 2008. Mixing and blending syntactic and semantic dependencies. In *Proc. of CoNLL*, pages 248–252.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The conll 2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proc. of CoNLL*, pages 159–177.

Daniel Zeman and Zdeněk Žabokrtský. 2005. Improving parsing accuracy by combining diverse dependency parsers. In *Proc. of IWPT*, pages 171–178.

ISSN 1736-6305 Vol. 4
http://hdl.handle.net/10062/9206