Dependency Parser Combination with Weighted Voting

Abstract

We investigate the combination of dependency parsers using voting and three weighting schemes. The recent development of accurate deterministic classifier-based parsers (Yamada and Matsumoto, 2003; Nivre 2004) makes this type of combination particularly attractive. We show that deterministic parsers in combination offer accuracy equal to that of statistical constituent parsers, and higher than that of any of the deterministic parsers in isolation. This is also true for root accuracy, which has so far been a challenge for deterministic parsers. When statistical constituent parsers are included in the combination, we obtain higher accuracy than previously published results on dependency parsing using the Penn Treebank: an overall dependency accuracy of 93.9%, and root accuracy of 97.6%.

1 Introduction

In recent years, dependency-based formalisms have been shown to be useful in a number of scenarios, such as machine translation (Ding and Palmer, 2004), semantic role labeling (Hacioglu, 2004) and question answering (Punyakanok et al., 2004), to name a few. In addition, recently developed classifier-based dependency parsers (Nivre and Scholz, 2004; Yamada and Matsumoto, 2003) are almost as accurate in producing dependency structures as state-of-the-art constituent parsers, such as those of Collins (1997) and Charniak (2000), although the identification of dependency tree root nodes remains a challenge.

Having different parsers that produce dependency structures with high accuracy raises the question of whether we can combine the output of these parsers to achieve even higher accuracy. In a way similar to how Henderson and Brill (1999) combined three statistical constituent parsers with voting, where each parser votes for constituents that make up a combined parse tree, we explore the combination of different dependency parsers in a weighted dependency voting scheme.

2 Dependency Voting

In this work we focus on unlabeled dependencies, also referred to as barebones dependency structures and described in Eisner (1996). In this scheme, the syntactic analysis of an *n* word sentence is represented by *n-1* dependencies, each containing one head word and one dependent word. Every word in the sentence except for one is assigned exactly one head, and words may be heads of multiple dependents. The single word that remains without a head is the root of the dependency tree. Figure 1 shows an unlabeled dependency tree. For practical reasons, we assign the root word a special head called the LeftWall. This is simply a word appended to the beginning of every sentence, so that every word in the original sentence has exactly one head.

If *m* different parsers output a set of dependencies (forming a dependency structure) for a given sentence, combining these dependencies through voting is straightforward. Each of the *m* parsers specifies exactly one head for each of the *n* words in the sentence. So, for each word, there are *m* votes (one by

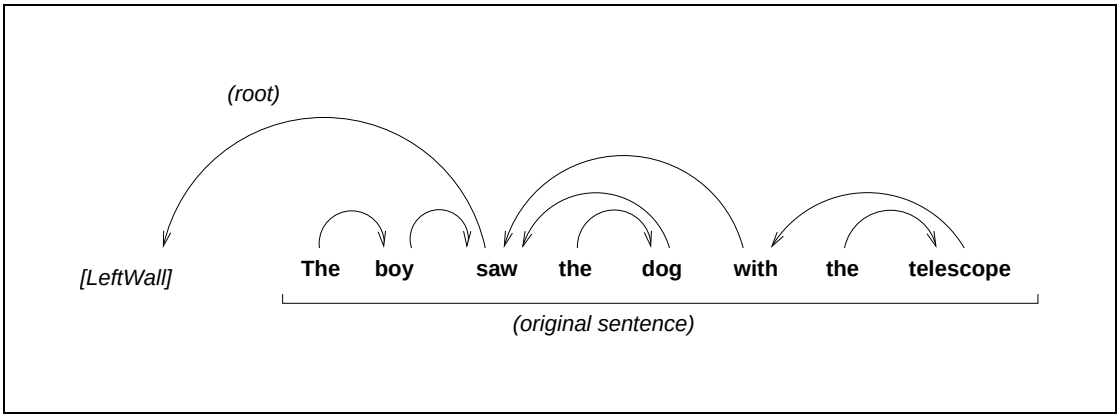

Figure 1: An unlabeled dependency structure. The LeftWall is inserted to be the head of the root word, so that every word in the original sentence has a head.

each parser) for what the head of that word should be. This means that voting is done on a word-by-word basis, and there is no guarantee that the final voted set of dependencies for a sentence will be a well-formed dependency tree.

One simple improvement to the basic dependency voting scheme is to allow parsers to have different weights associated with their votes. In other words, if we know in advance that parser A is more accurate on average than parser B, we can allow parser As vote to be more influential that parser Bs vote simply by giving it a larger weight.

A second improvement to the voting scheme, discussed later in section 4.4, is to enforce the construction of well-formed dependency trees. This can be done by abandoning the word-local view of voting and maximizing the votes for an entire dependency tree using either a maximum spanning tree algorithm or dynamic programming.

As in the case of Henderson and Brills constituent voting, a potential problem with our dependency voting scheme is the assumption of independence among the errors generated by the different parsers. If multiple parsers systematically vote for the same incorrect dependencies, the accuracy of the voted dependency structure may be lower than that of one or several of the voting parsers.

3 Classifier-Based Dependency Parsing

Recent work on dependency parsing by Nivre and Scholz (2004) and Yamada and Matsumoto (2003) shows that dependency structures can be built accurately in a deterministic fashion, where a classifier decides on a single parsing action to take at each step of a parsing algorithm with no backtracking. We briefly describe these two approaches, and present our own classifier-based parsers used in voting.

3.1 Yamada and Matsumoto (2003)

Yamada and Matsumotos parser achieves high dependency accuracy (90.7%) with an algorithm of quadratic run-time complexity and support vector machines (SVMs) for parser action classification. In a nutshell, the algorithm consists of moving a sliding window of two words over the input string from left to right, and at each position deciding whether there is a dependency with the word on the left as the head, with the word on the right as the head, or no dependency. If a dependency is found (and no further dependencies are expected to have the dependent word as a head), the dependent word is removed from the input string. Once the moving window reaches the end of the sentence, the process starts again at the beginning of the sentence. Parsing ends when a single word is left in the input string (and a complete dependency structure is built with the remaining word as the root), or no more dependencies can be formed (in which case the input string is rejected, although partial dependency structures can be recovered).

The classifier used to make the parsing decisions is trained by running the algorithm on sentences for which the dependency structure is known in advance, and associating a set of features from the

parsers configuration with the correct action. Features used for classification include the words in the sliding window and their parts-of-speech (POS), words to the left and right of the window (with POS), and already determined dependents of the words in the window.

3.2 Nivre and Scholz (2004)

Nivre and Scholzs parser has lower accuracy (about 87.7%) on unlabeled dependencies, but its algorithm is linear on the size of the input string. Nivre and Scholz use a simple left-to-right stack-based shift/reduce algorithm. This is simply an instance of the LR parsing algorithm for binary branching trees, but a classifier that decides on the parsing action replaces the LR table. The classifier used is k-nearest neighbors, and features include the word on top of the stack and its POS, the next word in the input string and its POS, the next n words in the input string (with POS), and the previously determined dependents of the word on top of the stack. Although we are interested in unlabeled dependencies, it is interesting to note that Nivre and Scholzs parser is actually designed for labeled dependencies, where the classifier also decides the dependency labels, and the labels can also be used as features. This results in having several different actions for the classifier to choose from. Training of the k-NN classifier is also done by running the algorithm on sentences with known dependency structures, and associating the correct parser actions with features that correspond to the parsers current state.

3.3 Our Dependency Parsers

Despite the similarities in Yamada and Matsumotos parser and (an unlabeled dependency version of) Nivre and Scholzs parser, they differ in two important ways. First, they differ in their choice of classifier. Second, their algorithms are different. While Yamada and Matsumoto use a quadratic run-time complexity algorithm, Nivre and Scholz use a linear algorithm. Despite its advantage in complexity, Nivre and Scholzs parser is less accurate. Nivre and Scholz argue that the difference in accuracy is explained by the use of a slightly less accurate part-of-speech tagger, the absence of true dependency labels in the training data, and the use of a linear algorithm that performs a single pass over the input string, as opposed to Yamada and Matsumotos multiple passes. Our experiments indicate that, although these factors may account for a small part of the difference, none of them are nearly as important for parser accuracy as the choice of classifier and features. Based on this observation, we describe the parsers we used for voting. The main difference we exploited among the parsers was the directionality of processing (left-to-right, right-to-left, bi-directional). Because of the deterministic nature of these parsers, the order in which words are processed makes a significant difference in the state of the parser at any given point. By varying directionality, we hope to achieve the diversity in results that is necessary for successful voting, while preserving the accuracy of each individual parser (which is, clearly, also important for successful results in voting).

LR-SVM

Our first parser (called LR-SVM) runs in linear time, like the one of Nivre and Scholz, but is about as accurate as Yamada and Matsumotos.

The LR-SVM parser uses the same basic left-to-right algorithm as Nivre and Scholz parser, but because we are dealing with unlabeled dependencies, we simply ignore the dependency types. Although the same algorithm is used, we have additional features that are used for classifying parser actions. In addition to the features used by Nivre and Scholz, we use:

- the word and POS of the item immediately below the top of the stack, and the item immediately below that;
- the words and POS surrounding (to the left and to the right, in the original sentence) the item on top of the stack;
- the words and POS surrounding (to the left and to the right, in the original sentence) the next word in the input;
- a distance feature that simply reflects how many words apart the word on the top of the stack and the next word in the input are.

Finally, instead of using k-NN for classification, we use SVMs with a third degree polynomial kernel in an all-against-all scheme for multi-class classification.

RL-SVM

Our second parser, RL-SVM, works in the same way as LR-SVM, except that (as the name suggests) processing is done right-to-left, instead of left-to-right. As we will see in section 4.3, the accuracy of RL-SVM is slightly lower than that of LR-SVM.

BIDIR-SVM

Our third parser, BIDIR-SVM, uses an algorithm that is very similar to the one used by Yamada and Matsumoto. The main difference is that instead of moving the sliding window from left-to-right at each iteration over the input string, we alternate between moving the window from left-to-right and right-to-left. The features used correspond to Yamada and Matsumotos feature set with a context length of two words to the left and two words to the right, with the addition of the distance feature also used in LR-SVM and RL-SVM.

LR-kNN, RL-kNN and BIDIR-kNN

Although the use of k-NN instead of SVMs in the three parsers described above results in a consistent and significant drop in accuracy, it is possible that their inclusion in the weighted voting scheme could be beneficial. It is important to note that even a single different classification decision in the LR and RL parsers could result in a significant difference in the dependencies generated, once again because of the deterministic nature of the parsers. This effect is lessened in the BIDIR parser, since it makes multiple passes over the input, and a single difference has more ample opportunity to remain isolated. These are important considerations, since the inclusion of parsers that make similar errors could hurt the voting scheme, as discussed in the end of section 3.3.

4 Experiments

We combined the parsers described in section 3 using the weighted dependency voting scheme described in section 2. In addition, we present a combination approach that guarantees a dependency tree as output. The details of these experiments are described below.

4.1 Data

As it is common in efforts to develop and evaluate corpus-based parsers for English, we use the standard split of the Wall Street Journal corpus of the Penn Treebank for training (sections 02 to 21) and testing (section 23). Section 22 was used during parser development, especially for feature selection and parameter tuning, and section 00 was used to set the voting weights, as explained in sections 4.3. Like in other recent work on dependency parsing using the Penn Treebank, constituent trees are converted to unlabeled dependencies automatically using a slightly modified version of the Collins (1996) head-table for lexicalizing constituent trees.

4.2 Single Parser Evaluation

Each parser described in section 3.3 was evaluated on section 23. We used two evaluation metrics for dependencies, also used by Nivre and Scholz (2004):

- Attachment Accuracy, also called Unlabeled Attachment Score (UAS): the number of correct dependencies (including root dependencies to LeftWall), divided by total number of dependencies (we do not include punctuation in the calculation of UAS);
- Root Accuracy (RA): the number of correct root assignments, divided by the total number of root assignments.

Table 1 shows the results for each of our parsers, the parsers of Nivre and Scholz (N&S) and Yamada and Matsumoto (Y&M) [1], and the dependencies extracted from the output of the parsers of Charniak (2000) and Collins (1997)[2].

A few interesting points are worth noting. First, the right-to-left parsers are significantly less accurate than their left-to-right counterparts. Also, the bidirectional parsers are less accurate than the left-to-right parsers, despite their many passes over the input string. As expected, each parser that used SVMs outperformed their k-NN counterparts. In particular, LR-SVM was outperformed in attachment score and dependency accuracy only by the Charniak parser. Finally, the root accuracy of all of the deterministic parsers is well below the root accuracy for the two statistical parsers.

[1] N&S was retrained for these experiments, and Y&M is our implementation of Yamada and Matsumoto's parser.

[2] We use Bikel's implementaion of the Collins parser in our experiments (Bikel, 2002).

Parser	UAS	RA
LR-SVM	91.2	92.6
RL-SVM	90.1	86.3
BIDIR-SVM	89.6	89.1
LR-kNN	88.1	85.1
RL-kNN	86.3	83.9
BIDIR-kNN	87.8	86.9
N&S	87.7	85.0
Y&M	90.7	91.7
Charniak	92.3	97.3
Collins	91.2	96.0

Table 1: Evalulation single parsers on dependencies from section 23 of the WSJ corpus of the Penn Treebank, using unlabeled attachment score (UAS) and root accuracy (RA).

4.3 Voting Evaluation

We evaluate dependency voting under three conditions: simple voting (no weights), weighted voting (each parser is assigned a single weight), and POS-weighted voting (each parser is assigned a weight for every part-of-speech of dependent words). In addition, we evaluate a combination strategy that combines the familiar CKY parsing algorithm with the idea of voting, and guarantees well-formed dependency trees.

Simple Voting

The simplest voting scheme assigns equal weight to the dependencies generated by every parser. Voting is done on a word-by-word basis (or dependency-by-dependency basis, since there is one dependency per word when we count the root dependency to the LeftWall). For each word, we check the head chosen by each of the parsers. Each of these heads receives one vote. The head with most votes is chosen as the head for the current word (ties are broken randomly).

It is easy to see how combining parsers in this scheme can produce poor results. For example, consider parser RL-kNN. Although it has much lower accuracy than LR-SVM, its vote is worth just as much. Additionally, if two or more of the parsers make many of the same mistakes, the voting can easily lead to erroneous analyses. To avoid these problems, the set of parsers used with simple voting was determined by trying every subset of our six parsers on section 00. Not surprisingly, the subsets containing LR-SVM, RL-SVM and BIDIR-SVM generally performed better. The optimal subset on section 00 was LR-SVM, RL-SVM, BIDIR-SVM and LR-kNN.

On the test set (section 23), the unlabeled attachment score of simple voting with the optimal subset is 91.9%, and root accuracy is 96.0%. In other words, using only deterministic classifier-based parsers, we obtain dependencies that are just as good as the ones obtained with the Collins parser. More surprisingly, the root accuracy of the combined parsers is also just as high, showing that it is possible to achieve high root accuracy with deterministic parsers. If we restrict the voting only to parsers that run in linear time (disqualifying BIDIR-SVM, but making the entire parsing process linear), we still obtain 91.7% attachment score and 95.6% root accuracy.

Weighted Voting

The weighted voting scheme is similar to simple voting, but each parser is assigned a weight for the votes it casts. Instead of just adding the number of votes for each dependency, we add the weights of each vote. We determine these weights by looking at individual parser accuracy a development set. More specifically, the weights for each parser are set to the unlabeled attachment score for the parser on section 00.

Combining the same optimal set of parsers as before, each now weighted by their performance on section 00, in the weighted voting scheme gives us an attachment score of 92.2%, and root accuracy of 94.1%. It is interesting that although dependencies are slightly more accurate overall than in simple voting, root accuracy is lower. This is not entirely surprising, given that the parser weights were optimized on all dependencies (the UAS metric).

POS-Weighted Voting

Instead of assigning a single weight to each parser, we can assign a set of weights, each associated with the part-of-speech of the dependent word of the dependency in question. The idea behind POS-weighted voting is that certain parsers may be better at attaching items with certain parts-

Combination	UAS	RA
Simple Voting	91.9	96.0
Weighted	92.2	94.1
POS-Weighted	92.3	94.9
Simple(+Ch/Co)	93.5	97.8
Weighted(+Ch/Co)	93.8	97.3
POS-Weighted(+Ch/Co)	93.9	97.6

Table 2: Results for voting experiments, evaluated as unlabeled attachment score (UAS) and root accuracy (RA). +Ch/Co indicated the inclusion of the Charniak and Collins parsers in the voting.

of-speech. By allowing weights to vary according to parts-of-speech, we attempt to capitalize on these differences. Once again these weights are set according to accuracy on section 00. In this case, accuracy for each part of speech is measured separately.

This scheme, in fact, generates the best overall results, with 92.3% attachment score and 94.9% root accuracy.

Summary of Results

The results for voting are summarized in table 2. In addition, we also show the results for voting when the Charniak parser and the Collins parser are included in the voting (denoted by +Ch/Co).

4.4 Obtaining Well-Formed Dependency Trees

As mentioned in section 2, voting decisions are local to each word, and there is no guarantee that a dependency set resulting from voting for a given sentence will form a dependency tree, or that the dependencies will even be completely connected or free of cycles.

We present two ways to combine multiple dependency parses of a sentence to produce a well-formed dependency tree. In both cases, we start by creating a graph where each word in the sentence is a node. We then create directed edges between nodes corresponding to words for which dependencies are obtained from any of the parsers. The edges are weighted according to what parser is responsible for generating the edge. These weights are the same weights used in weighted voting or POS-weighted voting. In cases where more than one parser indicates that the same edge should be created, the weights are added, just as in the voting scheme. As long as any of the parsers creates a valid dependency tree for the sentence, the directed weighted graph created this way will be fully connected.

Once the graph is created, we can simply find its maximum spanning tree, using, for example, the Chu-Liu/Edmonds directed MST algorithm (Chu and Liu, 1965; Edmonds, 1967). The maximum spanning tree maximizes the votes for dependencies given the constraint that the resulting structure must be a tree. However, there is no guarantee against crossing branches. While this may seem undesirable, the resulting tree generated from the combination of parsers should rarely contain crossing branches (for English). In addition, this would be a suitable scheme for combining structures in free-word-order languages, where branches are expected to cross.

A second option is to use dynamic programming to reparse the sentence. We proceed just as if we were parsing the sentence using the CKY algorithm, but we restrict the creation of new items in the CKY chart to pairs of words connected by the appropriate edges in the directed weighted graph, and assign these items the weight of their respective edges. Instead of the usual multiplication of probabilities, we simply add the values associated with each item used in the creation of a new item, and the value of the graph edge that allowed that item to be created.

Although it may seem that this is a lot of extra work, the search space described by the weighted directed graph is usually small, limiting the search for the best tree. The results obtained with either method are very close to results obtained voting with the same weighting scheme. For example, if the parser weights are set in the same way as for weighted voting, we obtain 92.3% attachment accuracy and 94.3% root accuracy, when we combine deterministic parsers only. When we combine deterministic and statistical parsers, we obtain 93.9% attachment accuracy and 97.3% root accuracy.

4.5 Conclusion

We have presented a dependency parser combination scheme based on weighted voting. We used a number of deterministic classifier-based dependency parsers, exploiting their differences in the order they process the input, and the method used for classification. We have also discussed how to select parsers

to be included in the voting process by using a development set.

The results obtained by combining deterministic parsers are quite good, surpassing previously published results for dependency parsers, and closely matching the results obtaining by extracting dependencies from state-of-the-art statistical constituent parsers. The identification of root words, considered a challenge for deterministic parsers, can be done accurately by voting, even when we use just three parsers that only make a single pass over the input. When we combine deterministic parsers with statistical parsers, we achieve even higher accuracy, close to 94% over all dependencies.

Finally, we have also described ways to obtain fully consistent dependency trees without sacrificing the accuracy obtained with voting, by using either maximum spanning trees or dynamic programming.

As future work, we plan to leverage on the differences between parsers to perform domain adaptation, possibly using semi-supervised learning.

5 Acknowledgments

This work was supported in part by the National Science Foundation under grant IIS-0414630.

References

Dan M. Bikel. 2002. Design of a multi-lingual, parallel-processing statistical parsing engine. *Proceedings of HLT (Human Language Technology Conference) 2002.* San Diego, CA.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. *In Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics.* Seattle, WA.

Yoeng-jin Chu and Tseng-hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14 (pp. 1396-1400).

Michael Collins. 1996. A new statistical parser based on bigram lexical dependencies. *Proceedings of the 34th Meeting of the Association for Computational Linguistics* (pp. 184-191). Santa Cruz, CA.

Michael Collins. 1997. Three generative, lexicalized models for statistical parsing. *Proceedings of the 35th Meeting of the Association for Computational Linguistics* (pp. 16-23). Madrid, Spain.

Walter Daelemans, Jacub Zavrel, Ko van der Sloot, and Antal van den Bosch. 2004. TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide. *ILK Research Group Technical Report Series* no. 04-02, 2004.

Yuan Ding and Martha Palmer. 2004. Synchronous dependency insertion grammars: a formalism for syntax based MT. *Proceedings of International Conference on Computational Linguistics*. Geneva, Switzerland.

Jack Edmonds. Optimum branchings. *J. Research of the National Bureau of Standards,* 71B (pp. 233-240).

Jason M. Eisner. 1996. Three new probablistic models for dependency parsing: An exploration. *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)* (pp. 340-345). Copenhagen, Denmark.

Kadri Hacioglu. 2004. Semantic role labeling using dependency trees. *Proceedings of International Conference on Computational Linguistics*. Geneva, Switzerland.

John C. Henderson and Eric Brill. 1999. Exploiting diversity in natural language processing: Combining parsers. *Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing*. College Park, MD.

Mitchel P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewics. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. *Proceedings of International Conference on Computational Linguistics* (pp. 64-70). Geneva, Switzerland.

Vasin Punyakanok, Dan Roth, and Wen-tau Yih. 2004. Natural language inference via dependency tree mapping: an application to question answering. *Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics*. Fort Lauderdale, FL.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines *Proceedings of the Eighth International Workshop on Parsing Technologies*. Nancy, France.