Automatic Grammar Induction: Combining, Reducing and Doing Nothing

Eric Brill
Microsoft Research
One Microsoft Way
Redmond, Wa. 98052 USA
brill@microsoft.com

John C. Henderson
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730 USA
jhndrsn@mitre.org

Grace Ngai
Department of Computer Science
The Johns Hopkins University
Baltimore, Md. 21218 USA
gyn@cs.jhu.edu

Abstract

This paper surveys three research directions in parsing. First, we look at methods for both automatically generating a set of diverse parsers and combining the outputs of different parsers into a single parse. Next, we will discuss a parsing method known as transformation-based parsing. This method, though less accurate than the best current corpus-derived parsers, is able to parse quite accurately while learning only a small set of easily understood rules, as opposed to the many-megabyte parameter files learned by other techniques. Finally, we review a recent study exploring how people and machines compare at the task of creating a program to automatically annotate noun phrases.

1 Introduction

This paper briefly surveys three research directions we hope will positively impact the field of parsing: parser output combination, automatic rule sequence learning, and manual rule sequence creation.

2 Parser Combination

2.1 Combining Off-the-Shelf Parsers

There has been a great deal of recent research in the machine learning community on combining the outputs of multiple classifiers to improve accuracy. In [4], we demonstrated that taking three off-the-shelf part of speech taggers and combining their outputs results in a significant performance improvement over any single tagger. More recently [8], we have demonstrated that the same is true

	Precision	Recall	(P+R)/2	F-Measure
Best Original Parser	89.6	89.7	89.7	89.7
Constituent Voting	92.4	90.1	91.3	91.3
Parser Switching	90.8	90.7	90.7	90.7

Table 1: Comparison of Single Parser with Combination Techniques

for parsing. We took three statistical parsers that had been trained on a portion of the Penn Treebank [5, 6, 10]. We explored two methods of combination: constituent voting and parser switching.

In constituent voting, each parser posits a set of constituents with weights (votes). We take the union of all posited constituents for a sentence, accumulating their weights, and then discard all constituents with a weight below a threshold. In the simplest version of constituent voting, each parser gives a weight of 1 to any constituent it posits, and then we retain all constuents posited by more than half of the parsers. We can also weight the vote according to the accuracy of the individual parser, or according to an estimate of the accuracy of an individual parser on a particular constituent type in a particular context. It is possible that, for instance, one parser will be very accurate at predicting noun phrases and very inaccurate at predicting prepositional phrases that occur before the main verb. In fact, in studying conditional weights, we were unable to achieve better performance than that achieved using the simplest weighting scheme. This could be either due to our lack of creativity, or an indication that the three parsers we used do not differ significantly in behavior across linguistic types or contexts.

In parser switching, each parser outputs a parse and our algorithm chooses to keep one of these parses. We can define a distance measure over trees, and then given a set T of parser outputs for a particular sentence, we would like to output the centroid tree for that set. In other words, we want to output: $\text{argmin}_{T_i} \sum_{T_j \in T} \text{Dist}(T_i, T_j)$. The motivation for this is that we can think of there being a *true* parse for a sentence, and then each parser makes random errors, independent of the errors made by the other parsers. Since finding the true centroid is intractible, we instead output: $\text{argmin}_{T_i \in T} \sum_{T_j \in T} \text{Dist}(T_i, T_j)$.

In table 1, we show the results obtained using both constituent voting and parser switching. We see that both combination techniques improve upon the best statistical parser, and indeed the results obtained using constuent voting are the highest accuracy numbers reported for the Wall Street Journal to date.

2.2 Generating A Set of Parsers

We have also explored how we can exploit the advantages of classifier combination when we have access to only one trainable parser [7, 9]. One technique for doing so is known as bagging [1]. In bagging, we take a single training set of size M and from it generate a new training set by sampling with replacement M times from the original set. We can do this multiple times to create multiple training sets. Then each derived training set is used to train a separate parser.

We generated 18 bags from a Czech treebank [7] and then used each bag to train a Collins Parser [6]. We then used simple constituent voting to combine the outputs of these 18 parsers on test data. Figure 1 shows that the number of bagged parsers that posit a constituent correlates nicely with the accuracy of that constituent. When only one of the 18 parsers outputs a constituent, that constituent is correct less than 10% of the time. When all 18 parsers output a constituent, that constituent is

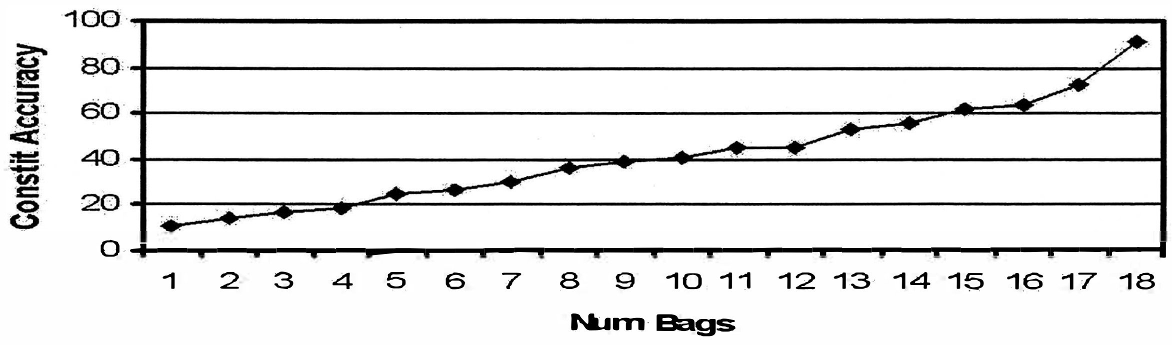

Figure 1: Accuracy versus number of parsers positing a constituent

Parser	Precision	Recall	P+R	F-Measure
Original	79.1	79.1	158.3	79.1
Bagged	79.9	79.9	159.8	79.9

Table 2: Czech Test Set Bagging Results

correct more than 90% of the time. Table 2 shows the results on Czech test data of a single Collins parser trained on the entire training corpus, compared to using bagging to generate 18 parsers and then combining the outputs of these parsers. Similar results have been obtained for English.

3 Transformation-Based Parsing

There have been great advances recently in the accuracy of parsers that are automatically trained from parsed corpora. One disadvantage of these grammar induction methods is that the derived linguistic knowledge is captured in opaque parameter files, typically many megabytes large. This makes it a challenge to capitalize on human intuitions to improve the machine-derived grammars. An alternative to these statistical induction methods is a method called Transformation-Based Parsing (TBP) [2, 11, 12]. In TBP, a grammar consists of an ordered sequence of tree transform rules. To learn the rules, we begin with some initial annotation of the training corpus (for instance, every sentence parsed as a flat structure under an S-node), and then we iteratively search for the transform rule whose application will bring the training set closest to the set of true parses, we append that rule to our rule list and apply it to the training corpus. Transform rules can add, delete or rearrange structure. An example of a learned rule is:

If a sequence begins with a Determiner and ends with a Noun and has a Verb or Modal immediately to the right of it, then bracket that sequence as an NP

Parser	F-Measure
PCFG	73.4
Transformations	83.0
Collins	86.7

Table 3: Transformation-Based Parsing Test Set Results

While there is still much work to be done in improving TBP, we have found that in its current state we achieve significantly better performance than a corpus-derived PCFG, but worse performance than the best statistical methods. In table 3 we show results from training on a 20k-sentence subset of the Penn Treebank WSJ corpus.

While the transformation-based system underperforms the best statistical systems, we believe it is an exciting path worth pursuing for the following reason: the Collins parser needs a trained parameter file about two hundred megabytes in size, while the transformation-based system achieves its accuracy with just a few hundred (mostly) readily understood rules. A human expert can easily study the rule list, and edit or add rules as appropriate.

4 Manual Rule Writing

In parsing, machine-learned rule-based systems achieve accuracy near that obtained by statistical systems. In many other natural language tasks, such as part of speech tagging, word sense disambiguation, and noun phrase chunking, machine-learned rule-based systems achieve performance on par with the best statistical systems, always learning a relatively small sequence of simple rules. This raises the question of whether we are really gaining anything by using machine learning techniques in NLP. While clearly a 200MB file of parameters is not something a person could create manually, creating small rule lists certainly is something a person could do.

In [3], we addressed this question for noun phrase chunking. We built a system to allow people to easily write rule sequences by hand and provided tools for effective manual error analysis. The advantage of a rule sequence, compared to a CFG for instance, is that the human does not have to be concerned about how rules interact. In manually generating a weighted CFG, there are complex interactions between rules, which makes manual grammar adjustment difficult. With rule sequences, at any stage of development the list of rules currently in the sequence can be ignored. In composing the i+1st rule, the person need only study the corpus and try to derive a rule to improve the accuracy of the training corpus in its current state, having been processed by the previous i rules.

We had students in a Natural Language Processing class write rule sequences for NP-bracketing. We found that the best students wrote rules that achieved test-set performance comparable to the best machine-learning system, and were able to achieve this performance after just a few hours of rule writing. We believe this raises a number of interesting issues, such as: Does machine learning really help? Can we effectively combine the (hopefully complementary) abilities of machine learning algorithms and human rule-writers?

5 Summary

We have briefly surveyed three lines of research for parsing: combining the outputs of multiple trained systems, reducing the problem to learning a small set of simple rules, and doing nothing automatically.

Acknowledgements

This work was partly funded by NSF grant IRI-9502312.

References

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[2] E. Brill. Transformation-based error-driven parsing. In *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, The Netherlands, 1993.

[3] Eric Brill and Grace Ngai. Man vs. machine: A case study in base noun phrase learning. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 1999.

[4] Eric Brill and Jun Wu. Classifier combination for improved lexical disambiguation. In *Proceedings of the 17th International Conference on Computational Linguistics*, 1998.

[5] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artifial Intelligence*, 1997.

[6] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Assocation of Computational Linguistics*, 1997.

[7] Jan Hajič, E. Brill, M. Collins, B. Hladka, D. Jones, C. Kuo, L. Ramshaw, O. Schwartz, C. Tillmann, and D. Zeman. Core natural language processing technology applicable to multiple languages. *Prague Bulletin of Mathematical Linguistics*, 70, 1999.

[8] John Henderson and Eric Brill. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[9] John Charles Henderson. *Exploiting Diversity for Natural Language Parsing*. PhD thesis, Johns Hopkins University, August 1999.

[10] Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, 1997.

[11] Giorgio Satta and Eric Brill. Efficient transformation-based parsing. In *Proceedings of ACL*, 1996.

[12] Marc Vilain and David Palmer. Transformation-based bracketing: fast algorithms and experimental results. In *Proceedings of the Workshop on Robust Parsing (at ESSLLI-96)*, 1996.