Parsing with a Mixture of PCFGs

Leon Barrett and Slav Petrov
University of California at Berkeley
{lbarrett, petrov}@eecs.berkeley.edu
Spring 2005

Abstract

We modify the PCFG language model by introducing a mixture over grammars. We then investigate the performance of such a mixture and analyze what information is gained by its introduction. We examine two different types of parsers and find optimal parameters (number of grammars, training iterations, etc.) for our language model.

1 Introduction

In the area of natural language processing, one attempts to allow computers to harvest information from human language. Humans are very good at this, while historically, computers have performed very poorly. In order to extract meaning from an utterance or piece of text, it may be useful to know how that text is structured grammatically. For instance, the it may be helpful to identify the main verb of the sentence. This extraction of grammar from words is called parsing.

Probabilistic Context Free Grammars (PCFGs) have proven very useful in the area of natural-language parsing. Both lexicalized [2, 3] and unlexicalized [6] parsers can produce impressive results. In this paper, we enrich the PCFG language model by removing the assumption that all language is generated from a single grammar. Instead, we assume that each sentence comes from a grammar chosen from a mixture of grammars. Using a mixture of estimators instead of a single estimator was first suggested in the statistics literature [7] and has since been adopted in machine learning since [5].

The general idea is that not all language has the same grammar. Clearly, speech usually comes from a different grammar than written text; it would be rare for someone to speak in the same manner as they would write a research paper. Within text, too, grammars often change, such as a paper's abstract being written differently than the body of the paper. Furthermore, grammar switches may come quite rapidly; consider the case of quoting other text. Even a single individual may generate language while switching grammars. Our model works by constructing several possible grammars and attempting to parse text with each of them.

The paper is structured as follows: After a brief background review in the next section, a detailed presentation of our method is given in section three. In section four we show our empirical results for the various experiments we ran. We conclude in section five and give an outlook into future work in section six.

2 Background

Lexicalized parsing, in which words influence the resulting structure by means other than just their parts of speech, generally outperforms unlexicalized parsing. However, according to [6], unlexicalized parsing has several advantages. In particular, the grammar is simpler and more compact, and it is easier to understand its inner workings. Therefore, while investigating this new mixture of grammars technique, we implement our grammar mixing on top of a simple unlexicalized parser.

In order to remove some of the well understood imperfections of a raw unlexicalized PCFG read from training trees, Klein et al. [6] add parent annotations and remove sibling annotations from their rules. They call this technique vertical and horizontal markovization. The approach is based on the observation that most categories are expanded in different ways depending on their position within the sentence. For example, a NP in subject position is 8.7 times more likely than an object NP to expand into a single pronoun. This external context can be captured by adding parent information to the category. The second imperfection of a raw PCFG is that many of the complicated and specific rules have been seen only once or not at all during training. By binarizing the grammar and removing sibling annotation, one is able to soften this sparsity problem.

3 Method

There are two components to our method: The language model consisting of a mixture of PCFGs and the parsers

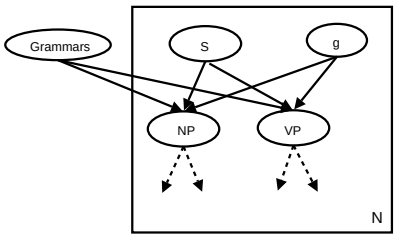

Figure 1: A model of a simple mixture of grammars

that we use to parse according to our language model. Both will be introduced in this section.

3.1 Language Model

We enrich the PCFG language model by removing the assumption that all language is generated from the same grammar. Instead, we assume that each sentence comes from a grammar chosen from a mixture of grammars. This is visible in the model in Fig. 3.1.

We train our parser via the EM algorithm [4]. First, we assign each training sentence a particular mixture of grammars. Then, we iteratively:

- Train each grammar, weighting the value of each sentence by the probability the sentence came from that grammar.
- Update the probability of each sentence coming from each grammar.

We want each of our grammars to be able to parse the same sentences. Therefore we require that all sentences have a fractional assignment of at least 10^{-6} to each grammar.

3.2 Parser Types

An interesting question is how to use the different grammars when trying to parse a previously unseen sentence. In our experiments we used two closely related parsing models, which we refer to as sum parser and max parser, for reasons that will become apparent very soon. While there are many ways to make use of the different grammars in parsing, we hope that the two we chose are easy enough to understand and sophisticated enough in order to produce high quality results.

3.2.1 The Sum Parser

Both our parser are modified versions of the Cocke-Kasami-Younger parser [8]. The "sum" parser computes the most likely sentence structure for all grammars. The CKY parser uses a chart where information about incomplete parses, called edges is stored. Each edge contains information about a possible structure covering a certain span and the probability associated with this structure. In each step the most likely structure is taken from a priority queue of edges called the agenda, and all possible derivation rules from the grammar are applied in order to produce new edges with easy-to-compute probabilities.

Since the different grammars contain the same rules (but with different probabilities), it is easy to modify the standard CKY parser so that it parses with several grammars. To do so we add an array of probabilities to each edge. In this array, we store the likelihood of the edge dependent on the different grammars. The total likelihood is just the sum (hence the name) of all these likelihoods and is used as the priority for the agenda. The first edge of type S taken off the agenda that spans over the whole sentence will therefore be the most likely sentence structure for all grammars:

$$P(w_{1n}|G_1, G_2, ...G_K) \propto \sum_{k=1}^{K} P(G_k) \cdot P(w_{1n}|G_k)$$

where w_{1n} denotes the sentence (words 1 through n) and the G_k are the different grammars. This can be derived quite simply using standard probabilistic rules.

$$P(w_{1n}) = \sum_k P(w_{1n}, G_k) = \sum_k P(w_{1n}|G_k)P(G_k)$$

3.2.2 The Max Parser

The idea behind the "max" parser is to rely only on the grammar that produces the sentence structure with the highest likelihood. The max parser differs from the sum parser in the way that the agenda priority is computed. As one could have guessed, the maximum of the different likelihoods is used instead of the sum:

$$P(w_{1n}|G_1, G_2, ...G_K) \propto \max_{k=1..K} P(G_k) \cdot P(w_{1n}|G_k)$$

This has some of the same advantages as the Viterbi algorithm. Essentially, there are some sentences which are not likely in any particular grammar, but which can be generated by any of several grammars. Thus, we would expect to see them often only if the speaker is changing grammars frequently. However, it is reasonable to expect that a person will use a consistent grammar for a period of time, which means that these sentences are being overvalued by the sum algorithm. Admittedly, this parser abandons the purity of our probabilistic model of a mixture of grammars; however, the hope that it might improve performance makes it worthwhile to examine.

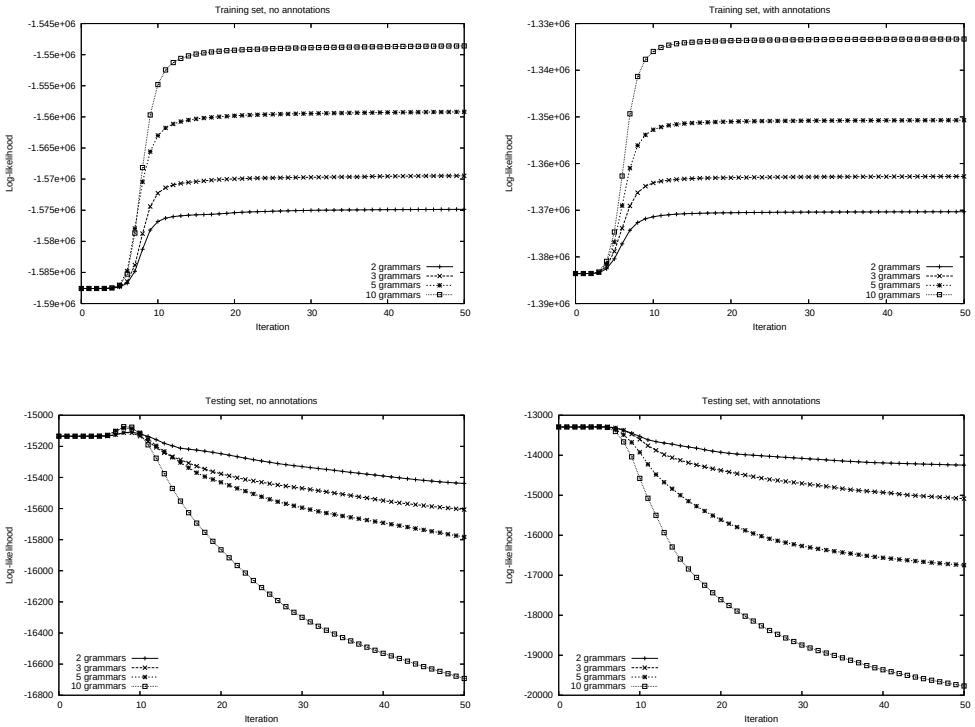

Figure 2: Log likelihood and accuracy of training data and testing data during training

4 Results

While our method is relatively easy to explain and understand, there are several parameters that need to set in order to make it work well. Probably the most important question is: how many grammars are there in the Penn Treebank? Since we are using an EM-approach we need to decide for how many iterations to train our model. Additionally it would be interesting to see whether our model behaves differently depending on whether the grammar is annotated or not. Finally we would like to know whether there is a significant difference between the sum and max parser and if so, which one produces higher accuracies. We will present results from the experiments and the optimal settings in this section.

4.1 Experimental Setup

We use the same setup that has been used by previous work in order to facilitate comparison. Our training set consists of sections 2 to 21 of the WSJ section of the Penn Treebank. We used the first 20 files (393 sentences) of section 22 as a validation set that was used for tuning the parameters of our models. We tested our model on the first 20 files (158 sentences) of section 23. (This is a noticeably smaller test set than most previous work has used; in particular, [6] used the entirety of section 23 as a test set. However, we found ourselves restricted in the amount of computation time available to us.) The training trees were annotated or transformed in some way and then (unsmoothed) maximum-likelihood estimates were used for rule probabilities. For the tagging probabilities we used add-one smoothing which was applied to unknown words and words that were seen 10 times or less during training. Parsing was done with a simple array-based Java implementation of a generalized CKY parser.

To evaluate parsers, we use the *F1* score of their parses. Two standard measures of accuracy are the *precision* and *recall*. The precision is the proportion of tags in the computer-parsed structure (the "guessed parse") that match the tags in the human-generated parse (the "gold standard"). The recall is the proportion of tags in the gold standard which appear in the guessed parse. So, there is often a trade-off; one can improve the precision of a parser at the expense of its recall, and vice versa. As a result, we use the F1 score, which is the geometric mean of the precision and recall.

$$\text{F1} = \sqrt{PR}$$

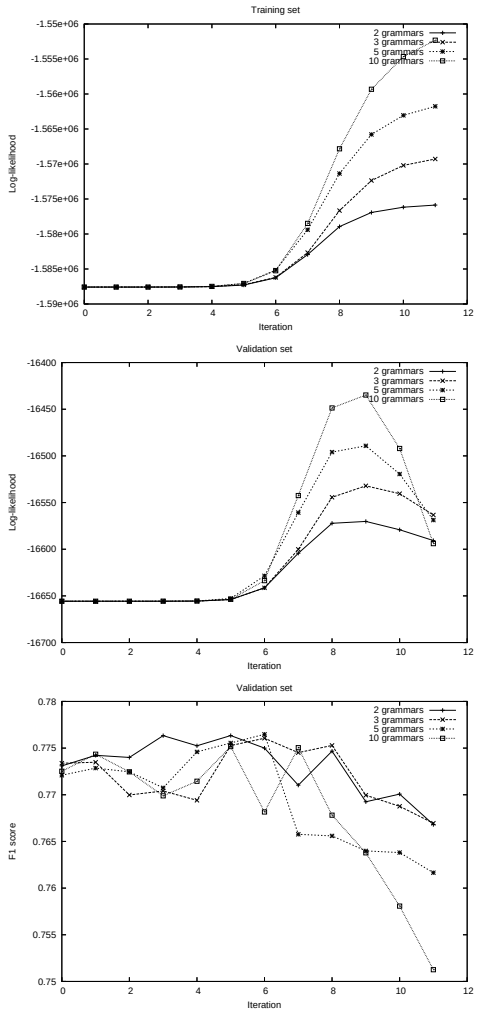

Figure 3: Log likelihood and accuracy of training data and validation data during training

4.2 Training

We trained our mixture of grammars in the standard way for a probabilistic model; we used the Expectation Maximization (EM) algorithm. Although it is usually used to learn parameters to give the best accuracy, the EM algorithm actually promises only one thing. It always attempts to increase the likelihood of the training data. In general, one hopes that because log likelihood tends to correlate with accuracy, this maximum corresponds to the most accurate set of parameters. In practice, the EM algorithm first increases the accuracy (fitting), but then decreases it (overfitting).

In Fig. 2, we trained our model for many iterations and plotted the log likelihood of the training data along with that of the testing data. As expected, the log likelihood of the testing data increased at first, but then fell off sharply as we began to overtrain. What was surprising was the speed with

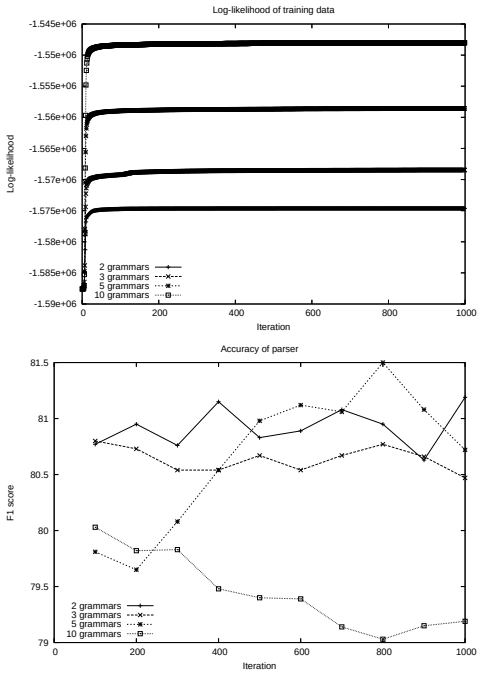

Figure 4: The accuracy changes surprisingly much with additional training, though the log likelihood seems to indicate convergence

which this happened; we expected that our model could take hundreds of iterations to converge to a good solution, while in fact this happened in less than ten iterations.

We were also curious how log likelihood matched up with accuracy. As it turns out, the accuracy begins to dip even before the log likelihood of the testing data peaks does. In Fig. 3, one can see that the accuracy fluctuates fairly randomly, but it clearly falls off as overfitting begins.

In a nutshell, EM works because increasing the likelihood of some correct data should decrease the likelihood of incorrect data, since there is only a fixed amount of probability to spread among all possible parse trees. Because the log likelihood on the testing data does not correlate well with the parser's accuracy, we conclude that while the parser is correctly eliminating some incorrect parses, it must also be increasing the accuracy of some of the bad parses. This means that the parser must be learning something that only indirectly indicates a correct parse.

4.2.1 Accuracy fluctuation

As we train our mixture of grammars, its accuracy fluctuates dramatically–as much as 1% every iteration! However, the log likelihood of the correct parse trees changes very smoothly. Furthermore, the log likelihood of the training data converges quite swiftly. (See Fig. 4.) How can these facts be reconciled?

Annotation	Parser	1g	2g	3g	5g	10g
without annotation	max	77.07	77.33 ±0.17	77.22 ±0.30	77.03 ±0.32	76.97 ±0.66
	sum		77.40 ±0.47	77.32 ±0.48	77.28 ±0.17	77.40 ±0.75
with annotation	max	80.76	80.86 ±0.12	80.61 ±0.35	80.90 ±0.17	80.33 ±0.70
	sum		81.12 ±0.24	81.05 ±0.21	81.00 ±0.27	80.79 ±0.45

Table 1: Overview of average F1-score and standard deviation of the F1-score (for 4 runs) on the test set for the different configurations depending on the number of grammars.

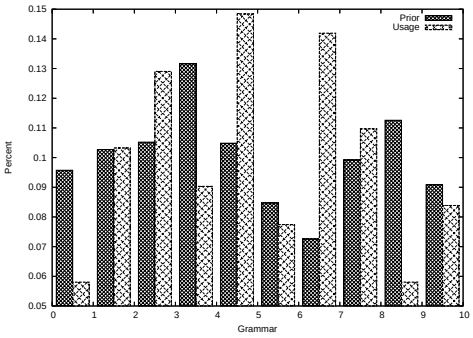

Figure 5: Grammar prior versus actual usage

The most reasonable explanation is that there are many trees with similar likelihood but with very different accuracy. As the mixture of grammars trains, and the rule probabilities change slightly, the most likely parse changes quickly, resulting in random fluctuations in the F1 score. This leads us to believe that the mixture of grammars is not "homing in" on a language model as well as we would like. If it were, then one parse (hopefully the correct one) would become noticeably more probable than any other. The mixture of grammars is not even focusing in on a bad language model, because that would lead to one parse, of poor but consistent accuracy, dominating. Instead, many parse trees remain nearly equally probable.

4.3 Number of Grammars

In order to determine the optimal number of grammars we ran our algorithm with different settings and computed the accuracy of the obtained parsers on the test set. For each set of parameters we trained the parser four times (each time with a slightly different initialization). We are showing the the average F1-score and it's standard deviation in table 4.3. Several phenomena can be observed:

- As expected, annotating the grammar always increases the accuracy, independent of the other parameters.
- The difference between the sum parser, which returns the most likely sentence structure for all grammars, and the max parser, which returns the structure that had the highest likelihood for a single grammar, is very small. But, for every single setting, the sum parser scores slightly higher.
- The F1-score stays almost constant when the number of grammars is varied, but one can also see that there it has a tendency to decrease when the number of grammars is increased.

It should be noted that not only the F1-scores are very similar, but in fact most of the produced parse trees are exactly the same even with different number of grammars. This is somewhat surprising since we expected different numbers of grammars to capture different aspects of language.

4.4 Grammar Priors

When we calculate the probability of generating a sentence, we consider both the probability of the sentence given each grammar and the probability of choosing that grammar. That is, we also use the *prior* over the grammars. Since the max parser presumes that a sentence is generated from exactly one grammar, one interesting question is how well the prior matches the grammar's frequency of use.

It turns out that grammars are not used in particularly close approximation to their priors. There is some correlation, but it is not very strong, as can be seen in Fig. 5. One possible reason is *grammar drift*. We evaluate on data which chronologically follows the training data from the Wall-Street Journal portion of the Penn Treebank. As a result, the test data may have a different subject matter, or even use slightly different grammar, than the training data.

However, the other, less appealing possibility is that the mixture of grammars is simply learning something particular to the training data alone. That is, the mixture is overtrained and simply does not learn any generalizing features. Given the performance of this language model, we are forced to conclude that this is most likely case the case.

5 Conclusions

In the end, we had some rather unexpected results. The accuracy and log likelihood on the validation set peaked after an astonishingly low number of training iterations. We anticipated several hundred training iterations to be needed, but we found that eight or nine iterations were optimal. Furthermore, this number was independent of the number of grammars in the mixture.

In addition, we found that training with different numbers of grammars produced identical parse trees. This is surprising because theoretically more grammars would capture a more detailed model.

Finally, the accuracy of the model seems to fluctuate even as the log likelihood converges. This is also an indicator that log likelihood and parsing accuracy are not closely related.

Unfortunately we have to conclude that no general improvement of statistical significance could be observed after the introduction of our language model. However, if we cut off the training at the right point, the model performs comparably to the unlexicalized single grammar parser. This leads us to the conclusion that the mixture of grammars is learning something that is unrelated to parsing accuracy.

6 Future Work

Although we could not significantly increase the accuracy of parsing, we still believe that the introduction of a mixture of PCFGs could be beneficial. There are several directions of future work that seem worthwile pursuing:

A relatively simple extension to our current model would be the introduction of an additional common grammar. In this case, there would be a shared grammar and many specific grammars. Then, productions would be chosen based not only on its probability within the specific grammar, but on the sum of this with the probability of the production within the shared grammar. The gain here is that the resulting language model could still model the variation in the training data, but without being so tightly focused on these variations.

Our mixture of PCFGs model is very restrictive on the way that the different grammars can be combined; in fact, they cannot be combined at all. A way to break this inflexibility would be a Latent Dirichlet Allocation (LDA) type of model [1], in which each constituent can be generated by a different grammar. The advantage of this approach is that now grammar shifts can be dealt with on a much finer grain. Instead of changing grammars only on sentence boundaries, now the model could accommodate such things as in-sentence quotations and constructs with differing grammars, such as idiomatic phrases. In the LDA setting, the grammar for each constituent is chosen independently from the parent grammar. However, in our context, we would expect it to be better to draw the new grammar depending on the parent grammar and thereby enforce fewer grammar changes.

It is also possible that our model is already sufficient to capture some variation in grammar, but that the data we are looking at does not have enough variation to justify the use of several grammars. Applying our parser to a different corpus (e.g. the Brown corpus, where the data is drawn from five different genres) could show the superiority of the mixture of PCFGs method to the basic PCFG approach.

Acknowledgments

We would like to acknowledge Dan Klein's plentiful help. He suggested this mixture of grammars idea and was full of good ideas when results were less promising than we hoped.

We also appreciate the help of Srini Narayanan, who offered us an interesting project idea which, ultimately, we did not choose.

References

[1] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. In *NIPS'02*, 2002.

[2] E. Charniak, S. Goldwater, and M. Johnson. Edge-based best-first chart parsing. *Sixth Workshop on Very Large Corpora*, pages 127–133, 1998.

[3] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, Univ. of Pennsylvania, 1999.

[4] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, 1977.

[5] Z. Ghahramani and M. I. Jordan. Supervised learning from incomplete data via an EM approach. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 120–127. Morgan Kaufmann Publishers, Inc., 1994.

[6] D. Klein and C. Manning. Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL 2003)*, pages 423–430, 2003.

[7] D. Titterington, A. Smith, and U. Makov. *Statistical Analysis of Finite Mixture Distributions*. Wiley, 1962.

[8] D.H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208, 1967.