Assigning Function Tags to Parsed Text*

Don Blaheta and Eugene Charniak
{dpb,ec}@cs.brown.edu
Department of Computer Science
Box 1910 / 115 Waterman St.—4th floor
Brown University
Providence, RI 02912

Abstract

It is generally recognized that the common nonterminal labels for syntactic constituents (NP, VP, etc.) do not exhaust the syntactic and semantic information one would like about parts of a syntactic tree. For example, the Penn Treebank gives each constituent zero or more 'function tags' indicating semantic roles and other related information not easily encapsulated in the simple constituent labels. We present a statistical algorithm for assigning these function tags that, on text already parsed to a simple-label level, achieves an F-measure of 87%, which rises to 99% when considering 'no tag' as a valid choice.

1 Introduction

Parsing sentences using statistical information gathered from a treebank was first examined a decade ago in (Chitrao and Grishman, 1990) and is by now a fairly well-studied problem ((Charniak, 1997), (Collins, 1997), (Ratnaparkhi, 1997)). But to date, the end product of the parsing process has for the most part been a bracketing with simple constituent labels like `NP`, `VP`, or `SBAR`. The Penn treebank contains a great deal of additional syntactic and semantic information from which to gather statistics; reproducing more of this information automatically is a goal which has so far been mostly ignored. This paper details a process by which some of this information—the function tags—may be recovered automatically.

In the Penn treebank, there are 20 tags (figure 1) that can be appended to constituent labels in order to indicate additional information about the syntactic or semantic role of the constituent. We have divided them into four categories (given in figure 2) based on those in the bracketing guidelines (Bies et al., 1995). A constituent can be tagged with multiple tags, but never with two tags from the same category.[1] In actuality, the case where a constituent has tags from all four categories never happens, but constituents with three tags do occur (rarely).

At a high level, we can simply say that having the function tag information for a given text is useful just because any further information would help. But specifically, there are distinct advantages for each of the various categories. Grammatical tags are useful for any application trying to follow the thread of the text—they find the 'who does what' of each clause, which can be useful to gain information about the situation or to learn more about the behaviour of the words in the sentence. The form/function tags help to find those constituents behaving in ways not conforming to their labelled type, as well as further clarifying the behaviour of adverbial phrases. Information retrieval applications specialising in describing events, as with a number of the MUC applications, could greatly benefit from some of these in determining the where-when-why of things. Noting a topicalised constituent could also prove useful to these applications, and it might also help in discourse analysis, or pronoun resolution. Finally, the 'miscellaneous' tags are convenient at various times; particularly the `CLR` 'closely related' tag, which among other things marks phrasal verbs and prepositional ditransitives.

To our knowledge, there has been no attempt so far to recover the function tags in parsing treebank text. In fact, we know of only

* This research was funded in part by NSF grants LIS-SBR-9720368 and IGERT-9870676.

[1] There is a single exception in the corpus: one constituent is tagged with `-LOC-MNR`. This appears to be an error.

ADV	Non-specific adverbial	HLN	Headline	PUT	Locative complement of 'put'
BNF	Benefactive	LGS	Logical subject	SBJ	Subject
CLF	It-cleft	LOC	Location	TMP	Temporal
CLR	'Closely related'	MNR	Manner	TPC	Topic
DIR	Direction	NOM	Nominal	TTL	Title
DTV	Dative	PRD	Predicate	VOC	Vocative
EXT	Extent	PRP	Purpose		

Figure 1: Penn treebank function tags

Grammatical		**53.%**	**Form/Function**		**37.%**	**Topicalisation**		**2.2%**	**Miscellaneous**		**9.5%**
DTV	0.48%	0.25%	NOM	6.8%	2.5%	TPC	100%	2.2%	CLR	94.%	8.8%
LGS	3.0%	1.5%	ADV	11.%	4.2%				CLF	0.34%	0.03%
PRD	18.%	9.3%	BNF	0.072%	0.026%				HLN	2.6%	0.25%
PUT	0.26%	0.13%	DIR	8.3%	3.0%				TTL	3.1%	0.29%
SBJ	78.%	41.%	EXT	3.2%	1.2%						
VOC	0.025%	0.013%	LOC	25.%	9.2%						
			MNR	6.2%	2.3%						
			PRP	5.2%	1.9%						
			TMP	33.%	12.%						

Figure 2: Categories of function tags and their relative frequencies

one project that used them at all: (Collins, 1997) defines certain constituents as complements based on a combination of label and function tag information. This boolean condition is then used to train an improved parser.

2 Features

We have found it useful to define our statistical model in terms of *features*. A 'feature', in this context, is a boolean-valued function, generally over parse tree nodes and either node labels or lexical items. Features can be fairly simple and easily read off the tree (e.g. 'this node's label is X', 'this node's parent's label is Y'), or slightly more complex ('this node's head's part-of-speech is Z'). This is concordant with the usage in the maximum entropy literature (Berger et al., 1996).

When using a number of known features to guess an unknown one, the usual procedure is to calculate the value of each feature, and then essentially look up the empirically most probable value for the feature to be guessed based on those known values. Due to sparse data, some of the features later in the list may need to be ignored; thus the probability of an unknown feature value would be estimated as

$$P(f|f_1, f_2, \ldots, f_n) \approx \hat{P}(f|f_1, f_2, \ldots, f_j), \quad j \leq n\ , \qquad (1)$$

where $\hat{P}$ refers to an empirically observed probability. Of course, if features 1 through i only co-occur a few times in the training, this value may not be reliable, so the empirical probability is usually smoothed:

$$\begin{aligned} P(f|f_1, f_2, \ldots, f_i) \approx & \\ & \lambda_i \hat{P}(f|f_1, f_2, \ldots, f_i) \\ + \quad & (1-\lambda_i) P(f|f_1, f_2, \ldots, f_{i-1})\ . \end{aligned} \qquad (2)$$

The values for λ_i can then be determined according to the number of occurrences of features 1 through i together in the training.

One way to think about equation 1 (and specifically, the notion that j will depend on the values of $f_1 \ldots f_n$) is as follows: We begin with the prior probability of f. If we have data indicating $\hat{P}(f|f_1)$, we multiply in that likelihood, while dividing out the original prior. If we have data for $\hat{P}(f|f_1, f_2)$, we multiply that in while dividing out the $\hat{P}(f|f_1)$ term. This is repeated for each piece of feature data we have; at each point, we are *adjusting* the probability

$$P(f|f_1, f_2, \ldots, f_n) \approx \hat{P}(f)\frac{\hat{P}(f|f_1)}{\hat{P}(f)}\frac{\hat{P}(f|f_1, f_2)}{\hat{P}(f|f_1)}\cdots\frac{\hat{P}(f|f_1, f_2, \ldots, f_j)}{\hat{P}(f|f_1, f_2, \ldots, f_{j-1})}, \quad j \leq n$$
$$\approx \prod_{i=0}^{j}\frac{\hat{P}(f|f_1, \ldots, f_{i-1}, f_i)}{\hat{P}(f|f_1, \ldots, f_{i-1})} \tag{3}$$

we already have estimated. If knowledge about feature f_i makes f more likely than with just $f_1 \ldots f_{i-1}$, the term where f_i is added will be greater than one and the running probability will be adjusted upward. This gives us the new probability shown in equation 3, which is exactly equivalent to equation 1 since everything except the last numerator cancels out of the equation. The value of j is chosen such that features $f_1 \ldots f_j$ are sufficiently represented in the training data; sometimes all n features are used, but often that would cause sparse data problems. Smoothing is performed on this equation exactly as before: each term is interpolated between the empirical value and the prior estimated probability, according to a value of λ_i that estimates confidence. But aside from perhaps providing a new way to think about the problem, equation 3 is not particularly useful as it is—it is exactly the same as what we had before. Its real usefulness comes, as shown in (Charniak, 1999), when we move from the notion of a feature *chain* to a feature *tree*.

These feature chains don't capture everything we'd like them to. If there are two independent features that are each relatively sparse but occasionally carry a lot of information, then putting one before the other in a chain will effectively block the second from having any effect, since its information is (uselessly) conditioned on the first one, whose sparseness will completely dilute any gain. What we'd really like is to be able to have a feature *tree*, whereby we can condition those two sparse features independently on one common predecessor feature. As we said before, equation 3 represents, for each feature f_i, the probability of f based on f_i and all its predecessors, divided by the probability of f based only on the predecessors. In the chain case, this means that the denominator is conditioned on every feature from 1 to $i-1$, but if we use a feature tree, it is conditioned only on those features along the path to the root of the tree.

A notable issue with feature trees as opposed to feature chains is that the terms do *not* all cancel out. Every leaf on the tree will be represented in the numerator, and every fork in the tree (from which multiple nodes depend) will be represented at least once in the denominator. For example: in figure 3 we have a small feature tree that has one target feature and four conditioning features. Features b and d are independent of each other, but each depends on a; c depends directly only on b. The unsmoothed version of the corresponding equation would be

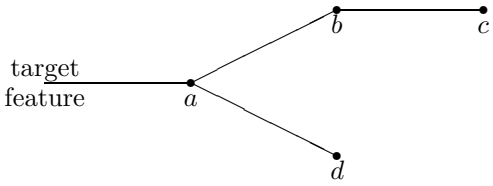

Figure 3: A small example feature tree

$$P(f|a, b, c, d) \approx \hat{P}(f)\frac{\hat{P}(f|a)}{\hat{P}(f)}\frac{\hat{P}(f|a, b)}{\hat{P}(f|a)}\frac{\hat{P}(f|a, b, c)}{\hat{P}(f|a, b)}\frac{\hat{P}(f|a, d)}{\hat{P}(f|a)},$$

which, after cancelling of terms and smoothing, results in

$$P(f|a, b, c, d) \approx \frac{P(f|a, b, c)P(f|a, d)}{P(f|a)}. \tag{4}$$

Note that strictly speaking the result is not a probability distribution. It could be made into one with an appropriate normalisation—the so-called partition function in the maximum-entropy literature. However, if the independence assumptions made in the derivation of equation 4 are good ones, the partition function will be close to 1.0. We assume this to be the case for our feature trees.

Now we return the discussion to function tagging. There are a number of features that seem

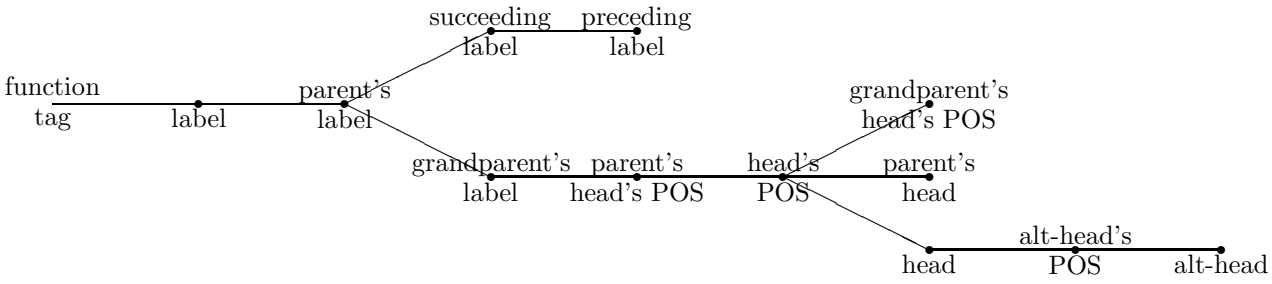

Figure 4: The feature tree used to guess function tags

to condition strongly for one function tag or another; we have assembled them into the feature tree shown in figure 4.[2] This figure should be relatively self-explanatory, except for the notion of an 'alternate head'; currently, an alternate head is only defined for prepositional phrases, and is the head of the object of the prepositional phrase. This data is very important in distinguishing, for example, 'by John' (where John might be a logical subject) from 'by next year' (a temporal modifier) and 'by selling it' (an adverbial indicating manner).

3 Experiment

In the training phase of our experiment, we gathered statistics on the occurrence of function tags in sections 2-21 of the Penn treebank. Specifically, for every constituent in the treebank, we recorded the presence of its function tags (or lack thereof) along with its conditioning information. From this we calculated the empirical probabilities of each function tag referenced in section 2 of this paper. Values of λ were determined using EM on the development corpus (treebank section 24).

To test, then, we simply took the output of our parser on the test corpus (treebank section 23), and applied a postprocessing step to add function tags. For each constituent in the tree, we calculated the likelihood of each function tag according to the feature tree in figure 4, and for each category (see figure 2) we assigned the most likely function tag (which might be the null tag).

[2]The reader will note that the 'features' listed in the tree are in fact not boolean-valued; each node in the given tree can be assumed to stand for a chain of boolean features, one per potential value at that node, exactly one of which will be true.

4 Evaluation

To evaluate our results, we first need to determine what is 'correct'. The definition we chose is to call a constituent correct if there exists in the correct parse a constituent with the same start and end points, label, and function tag (or lack thereof). Since we treated each of the four function tag categories as a separate feature for the purpose of tagging, evaluation was also done on a per-category basis.

The denominator of the accuracy measure should be the maximum possible number we could get correct. In this case, that means excluding those constituents that were already wrong in the parser output; the parser we used attains 89% labelled precision-recall, so roughly 11% of the constituents are excluded from the function tag accuracy evaluation. (For reference, we have also included the performance of our function tagger directly on treebank parses; the slight gain that resulted is discussed below.)

Another consideration is whether to count non-tagged constituents in our evaluation. On the one hand, we could count as correct any constituent with the correct tag as well as any correctly non-tagged constituent, and use as our denominator the number of all correctly-labelled constituents. (We will henceforth refer to this as the 'with-null' measure.) On the other hand, we could just count constituents with the correct tag, and use as our denominators the total number of *tagged*, correctly-labelled constituents. We believe the latter number ('no-null') to be a better performance metric, as it is not overwhelmed by the large number of untagged constituents. Both are reported below.

Table 1: Baseline performance

Category	Baseline 1	Baseline 2 (always choose most likely tag)			
	(never tag)	Tag	Precision	Recall	F-measure
Grammatical	86.935%	SBJ	10.534%	80.626%	18.633%
Form/Function	91.786%	TMP	3.105%	37.795%	5.738%
Topicalisation	99.406%	TPC	0.594%	100.00%	1.181%
Miscellaneous	98.436%	CLR	1.317%	84.211%	2.594%
Overall	94.141%	—	3.887%	66.345%	7.344%

Table 2: Performance within each category

	With-null	———No-null———		
Category	Accuracy	Precision	Recall	F-measure
Grammatical	98.909%	95.472%	95.837%	95.654%
Form/Function	97.104%	80.415%	77.595%	78.980%
Topicalisation	99.915%	92.195%	93.564%	92.875%
Miscellaneous	98.645%	55.644%	65.789%	60.293%

5 Results

5.1 Baselines

There are, it seems, two reasonable baselines for this and future work. First of all, most constituents in the corpus have no tags at all, so obviously one baseline is to simply guess no tag for any constituent. Even for the most common type of function tag (grammatical), this method performs with 87% accuracy. Thus the with-null accuracy of a function tagger needs to be very high to be significant here.

The second baseline might be useful in examining the no-null accuracy values (particularly the recall): always guess the most common tag in a category. This means that every constituent gets labelled with '-SBJ-TMP-TPC-CLR' (meaning that it is a topicalised temporal subject that is 'closely related' to its verb). This combination of tags is in fact entirely illegal by the treebank guidelines, but performs adequately for a baseline. The precision is, of course, abysmal, for the same reasons the first baseline did so well; but the recall is (as one might expect) substantial. The performances of the two baseline measures are given in Table 1.

5.2 Performance in individual categories

In table 2, we give the results for each category. The first column is the with-null accuracy, and the precision and recall values given are the no-null accuracy, as noted in section 4.

Grammatical tagging performs the best of the four categories. Even using the more difficult no-null accuracy measure, it has a 96% accuracy. This seems to reflect the fact that grammatical relations can often be guessed based on constituent labels, parts of speech, and high-frequency lexical items, largely avoiding sparse-data problems. Topicalisation can similarly be guessed largely on high-frequency information, and performed almost as well (93%).

On the other hand, we have the form/function tags and the 'miscellaneous' tags. These are characterised by much more semantic information, and the relationships between lexical items are very important, making sparse data a real problem. All the same, it should be noted that the performance is still far better than the baselines.

5.3 Performance with other feature trees

The feature tree given in figure 4 is by no means the only feature tree we could have used. In-

Table 3: Overall performance on different inputs

	With-null	———No-null———		
Category	Accuracy	Precision	Recall	F-measure
Parsed	98.643%	87.173%	87.381%	87.277%
Treebank	98.805%	88.450%	88.493%	88.472%

deed, we tried a number of different trees on the development corpus; this tree gave among the best overall results, with no category performing too badly. However, there is no reason to use only one feature tree for all four categories; the best results can be got by using a separate tree for each one. One can thus achieve slight (one to three point) gains in each category.

5.4 Overall performance

The overall performance, given in table 3, appears promising. With a tagging accuracy of about 87%, various information retrieval and knowledge base applications can reasonably expect to extract useful information.

The performance given in the first row is (like all previously given performance values) the function-tagger's performance on the correctly-labelled constituents output by our parser. For comparison, we also give its performance when run directly on the original treebank parse; since the parser's accuracy is about 89%, working directly with the treebank means our statistics are over roughly 12% more constituents. This second version does slightly better.

The main reason that tagging does worse on the parsed version is that although the constituent itself may be correctly bracketed and labelled, its exterior conditioning information can still be incorrect. An example of this that actually occurred in the development corpus (section 24 of the treebank) is the 'that' clause in the phrase 'can swallow the premise that the rewards for such ineptitude are six-figure salaries', correctly diagrammed in figure 5. The function tagger gave this SBAR an ADV tag, indicating an unspecified adverbial function. This seems extremely odd, given that its conditioning information (nodes circled in the figure) clearly show that it is part of an NP, and hence probably modifies the preceding NN. Indeed, the statistics give the probability of an ADV tag in this conditioning environment as vanishingly small.

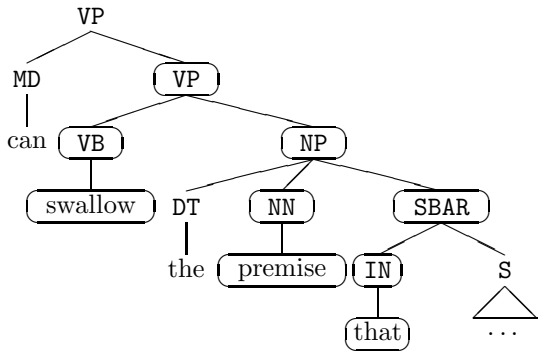

Figure 5: SBAR and conditioning info

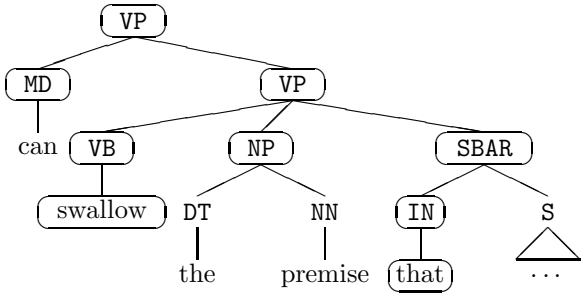

Figure 6: SBAR and conditioning info, as parsed

However, this was not the conditioning information that the tagger received. The parser had instead decided on the (incorrect) parse in figure 6. As such, the tagger's decision makes much more sense, since an SBAR under two VPs whose heads are VB and MD is rather likely to be an ADV. (For instance, the 'although' clause of the sentence 'he can help, although he doesn't want to.' has exactly the conditioning environment given in figure 6, except that its predecessor is a comma; and this SBAR would be correctly tagged ADV.) The SBAR itself is correctly bracketed and labelled, so it still gets counted in the statistics. Happily, this sort of case seems to be relatively rare.

Another thing that lowers the overall performance somewhat is the existence of error and inconsistency in the treebank tagging. Some tags seem to have been relatively easy for the human treebank taggers, and have few errors. Other tags have explicit caveats that, however well-justified, proved difficult to remember for the taggers—for instance, there are 37 instances of a `PP` being tagged with `LGS` (logical subject) in spite of the guidelines specifically saying, '[LGS] attaches to the NP object of *by* and not to the PP node itself.' (Bies et al., 1995) Each mistagging in the test corpus can cause up to two spurious errors, one in precision and one in recall. Still another source of difficulty comes when the guidelines are vague or silent on a specific issue. To return to logical subjects, it is clear that 'the loss' is a logical subject in 'The company was hurt by the loss', but what about in 'The company was unperturbed by the loss'? In addition, a number of the function tags are authorised for 'metaphorical use', but what exactly constitutes such a use is somewhat inconsistently marked. It is as yet unclear just to what degree these tagging errors in the corpus are affecting our results.

6 Conclusion

This work presents a method for assigning function tags to text that has been parsed to the simple label level. Because of the lack of prior research on this task, we are unable to compare our results to those of other researchers; but the results do seem promising. However, a great deal of future work immediately suggests itself:

- Although we tested twenty or so feature trees besides the one given in figure 4, the space of possible trees is still rather unexplored. A more systematic investigation into the advantages of different feature trees would be useful.
- We could add to the feature tree the values of other categories of function tag, or the function tags of various tree-relatives (parent, sibling).
- One of the weaknesses of the lexical features is sparse data; whereas the part of speech is too coarse to distinguish 'by John' (`LGS`) from 'by Monday' (`TMP`), the lexical information may be too sparse. This could be assisted by clustering the lexical items into useful categories (names, dates, etc.), and adding those categories as an additional feature type.
- There is no reason to think that this work could not be integrated directly into the parsing process, particularly if one's parser is already geared partially or entirely towards feature-based statistics; the function tag information could prove quite useful within the parse itself, to rank several parses to find the most plausible.

References

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Ann Bies, Mark Ferguson, Karen Katz, and Robert MacIntyre, 1995. *Bracketing Guidelines for Treebank II Style Penn Treebank Project*, January.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 598–603, Menlo Park. AAAI Press/MIT Press.

Eugene Charniak. 1999. A maximum-entropy-inspired parser. Technical Report CS-99-12, Brown University, August.

Mahesh V. Chitrao and Ralph Grishman. 1990. Statistical parsing of messages. In *DARPA Speech and Language Workshop*, pages 263–266.

Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23.

Adwait Ratnaparkhi. 1997. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Annual Conference on Empirical Methods in Natural Language Processing*, pages 1–10.