Decision Tree Parsing using a Hidden Derivation Model

F. Jelinek, J. Lafferty, D. Magerman, R. Mercer*, A. Ratnaparkhi, S. Roukos*

IBM Research Division
Thomas J. Watson Research Center
Yorktown Heights, NY 10598

1. Introduction

Parser development is generally viewed as a primarily linguistic enterprise. A grammarian examines sentences, skillfully extracts the linguistic generalizations evident in the data, and writes grammar rules which cover the language. The grammarian then evaluates the performance of the grammar, and upon analysis of the errors made by the grammar-based parser, carefully refines the rules, repeating this process, typically over a period of several years.

This grammar refinement process is extremely time-consuming and difficult, and has not yet resulted in a grammar which can be used by a parser to analyze *accurately* a large corpus of unrestricted text. As an alternative to writing grammars, one can develop corpora of hand-analyzed sentences (*treebanks*) with significantly less effort[1]. With the availability of treebanks of annotated sentences, one can view NL parsing as simply *treebank recognition* where the methods from statistical pattern recognition can be brought to bear.

This approach divides the parsing problem into two separate tasks: *treebanking*, defining the annotation scheme which will encode the linguistic content of the sentences and applying it to a corpus, and *treebank recognition*, generating these annotations automatically for new sentences .

The treebank can contain whatever information is deemed valuable by the treebanker, as long as it is annotated according to some consistent scheme, probably one which represents the intended meaning of the sentence. The goal of treebank recognition is to produce the exact same analysis of a sentence that the treebanker would generate.

As treebanks became available during the past five years, many "statistical models" for parsing a sentence w_1^n of n words still relied on a grammar. Statistics were used to simply rank the parses that a grammar allowed for a sentence. Unfortunately, this requires the effort of grammar creation (whether by hand or from data) in addition to the Treebank and suffers from the coverage problem that the correct parse may not be allowed by the grammar. Parsing with these models is to determine the most probable parse, T^*, from among all the parses, denoted by $\mathcal{T}_G(w_1^n)$, allowed by the grammar G for the sentence w_1^n:

$$T^* = \arg\max_{T \in \mathcal{T}_G(w_1^n)} p(T \mid w_1^n). \quad (1)$$

The *a posteriori* probability of a tree T given the sentence w_1^n is usually derived by Bayes rule from a generative model, denoted by $p(T, w_1^n)$, based on the grammar. For example, probabilistic CFGs (P-CFG) can be estimated from a treebank to construct such a model [1, 2].

But there is no reason to require that a grammar be used to construct a probabilistic model $p(T \mid w_1^n)$ that can be used for parsing. In this paper, we present a method for contructing a model for the conditional distribution of trees given a sentence without the need to define a grammar. So with this new viewpoint parsing avoids the step of extracting a grammar and is merely the search of the most probable tree:

$$T^* = \arg\max_{T \in \mathcal{T}(w_1^n)} p(T \mid w_1^n) \quad (2)$$

where the maximization is over all trees that span the n-word sentence. While others have attempted to build parsers from treebanks using correctly tagged sentences as input, we present in this paper the first results we know of in building a parser automatically that produces the surface structure directly from a word sequence and does not require a correct sequence of tags.

The probabilistic models we explore are conditional on the *derivational* order of the parse tree. In [4], this type of model is referred to as a *history-based* grammar model where a (deterministic) leftmost derivation order is used to factor the probabilistic model. In this work, we use a set of bottom-up derivations[2] of parse trees. We explore the use of a self-organized hidden derivational model as well as a deterministic derivational model to assign the probability of a parse tree.

In the remaining sections, we discuss the derivation history model, the parsing model, the probabilistic models for node

*F. Jelinek and R. Mercer, formerly of IBM, are now with John Hopkins University and Renaissance Technologies, Inc., respectively.

[1] In addition, these annotated corpora have a more permanent value for future research use than particular grammars

[2] Traditional use of derivation order identifies the order of application of grammar rules; in this work, we extend the notion to identify the order in which edges in a tree are created.

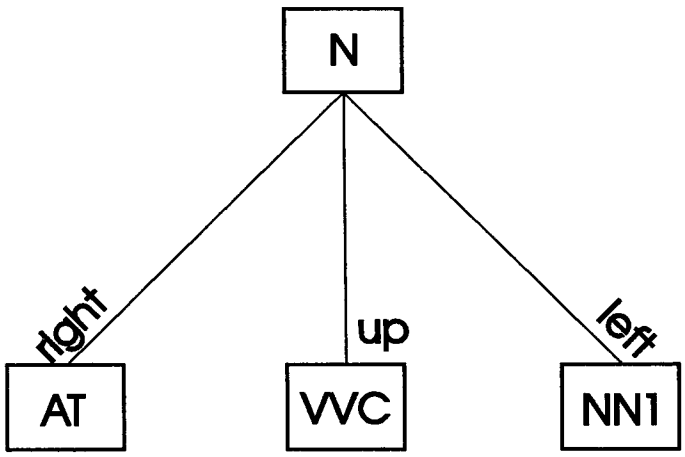

Figure 1: The extensions corresponding to a constituent for a phrase such as "the Enter key".

features, the training algorithms, the experimental results, and our conclusions.

2. A derivation history model

Current treebanks are a collection of n-ary branching trees, with each node in a tree labeled by either a non-terminal label or a part-of-speech label (called a tag). Usually, grammarians elevate constituents to the status of elementary units in a parse, especially in the case of rewrite-rule grammars where each rewrite rule defines a legal constituent. However, if a parse tree is interpreted as a geometric pattern, a constituent is no more than a set of edges which meet at the same tree node. In Figure 1, the noun phrase, "N", which spans the tags "AT VVC NN1", which correspond to an article, a command verb, and a singular noun, respectively, consists of an edge extending to the right from "AT," an edge extending straight up from "VVC," and an edge extending to the left from "NN1" (see Figure 1).

We introduce a new definition for a derivation of a parse tree by using Figure 2 which gives a subtree used in our parser for representing the noun phrase "the Enter key". We associate with every node in the parse tree two features, a name which is either a tag or a non-terminal label, and an extension which indicates whether the edge going to its parent is going right, left, up, or unary. Unary corresponds to a renaming of a non-terminal. By specifying the two features (name and extension) for each node we can reconstruct the parse tree. The order of the nodes in which we specify these two features defines the derivation order. We only consider bottom-up derivations. In a bottom-up derivation, a node is named first, it may be extended only after it's named, and it is not named until all of the nodes beneath it are extended. Naming a node maybe a tagging or labeling action depending on whether or not the node is a leaf in the parse tree.

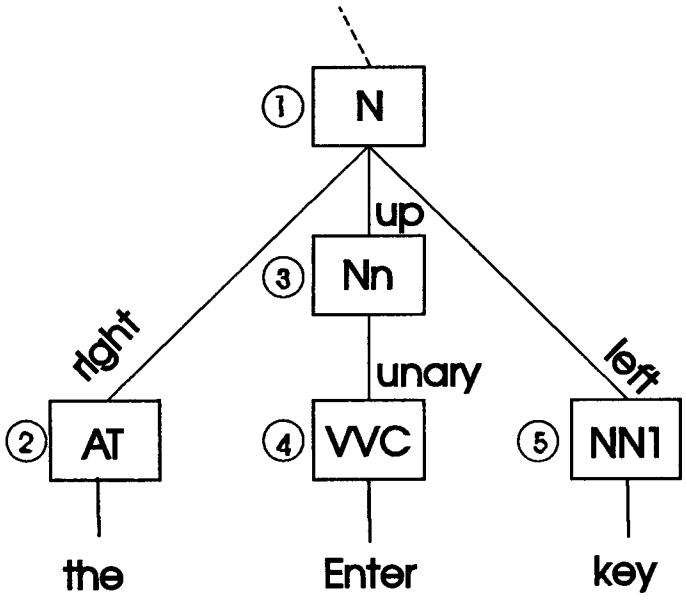

Figure 2: Representation of constituent and labeling of extensions.

Using Figure 2, one derivation is to tag the first word "the" as "AT", then to extend it "right", then to tag the third word "key" as "NN1", then to tag the second word "Enter" as "VVC" (command verb), then to extend the resulting node by a "unary", then to label the resulting node as "Nn" (computer noun), then to extend the resulting node "up", then to extend the "NN1" node by a "left" to yield a node that spans the whole phrase "the Enter key". By our definition of bottom-up derivation, it's only at this point in the derivation that we can label the node that spans the whole phrase as "N", and then extend it "left" as is implied in Figure 2. Using the node numbering scheme in Figure 2, we have at the beginning of this derivation the words with the nodes $\{2, 4, 5\}$ that have unassigned names. These are the active nodes at this point. Suppose that node 2 is picked and then tagged "AT". That corresponds to the derivation [2]; at this point, only nodes $\{2, 4, 5\}$ are active. If we pick node 2 again, then an extension step is required and the derivation is [2 2]. The derivation presented at the beginning of this paragraph corresponds to the sequence of nodes [2 2 5 4 4 3 3 5 1 1].

To derive the tree in Figure 1 when we are given the three-tag sequence, there are 6 possible derivations. We could start by extending any of the 3 tags, then we have either of 2 choices to extend, and we extend the one remaining choice, then we name the resulting node. This leads to 3x2x1=6 derivations for that tree.

If we use a window of 1, then only a single derivation is permitted and we call it the *bottom-up leftmost* derivation. In our example, this leftmost derivation would be [2 2 4 4 3 3 5 5 1].

3. The Parsing Model

We represent a derivation of a parse tree by the sequence of nodes as they are visited by the derivation, denoted by d. Denote by d_i the i-th node of the derivation d. Denote by l_{d_i} the name feature for a node selected at the i-th step in the derivation and by e_{d_i} its extension. A parse derivation is constructed by the following 2-step algorithm:

- select which node to extend among active nodes using $p(active = d_i \mid context)$,
- then either
  - assign a name to the selected node whether it is tagging or labelling a node (constituent) with a non-terminal label using $p(l_{d_i} \mid context)$, or
  - extend the selected node (which adds an edge to the parse graph) using $p(e_{d_i} \mid context)$.

If the node selected has its name identified then an extension step is performed otherwise a naming step is performed. Note that only extension steps change which nodes are active.

We have a different probabilistic model for each type of step in a parse derivation. The probabilistic models do not use the whole derivation history as context; but rather a five node window around the node in question. We will discuss this in more detail later on.

The probability of a derivation of a parse tree is the product of the probabilities of each of the feature value assignments in that derivation and the probability of each active node selection made in that derivation:

$$p(T, d \mid w_1^n) = \prod_{1<j<|d|} p_j$$

where

$$p_j = p(active = d_j \mid context(d_1^{j-1}))p(x_j \mid context(d_1^j))$$

where x_j is either the name l_j of node d_j or its extension e_j and d_1^j is the derivation up to the j-th step. The probability of a parse tree given the sentence is the sum over all derivations of that parse tree:

$$p(T \mid w_1^n) = \sum_d p(T, d \mid w_1^n)$$

Due to computational complexity, we restrict the number of bottom-up derivations we consider by using a *window* of n active nodes. For a window of 2, we can only choose either of the two leftmost nodes in the above process. So for the parse in Figure 1, we only get 4 derivations with a derivation window of 2.

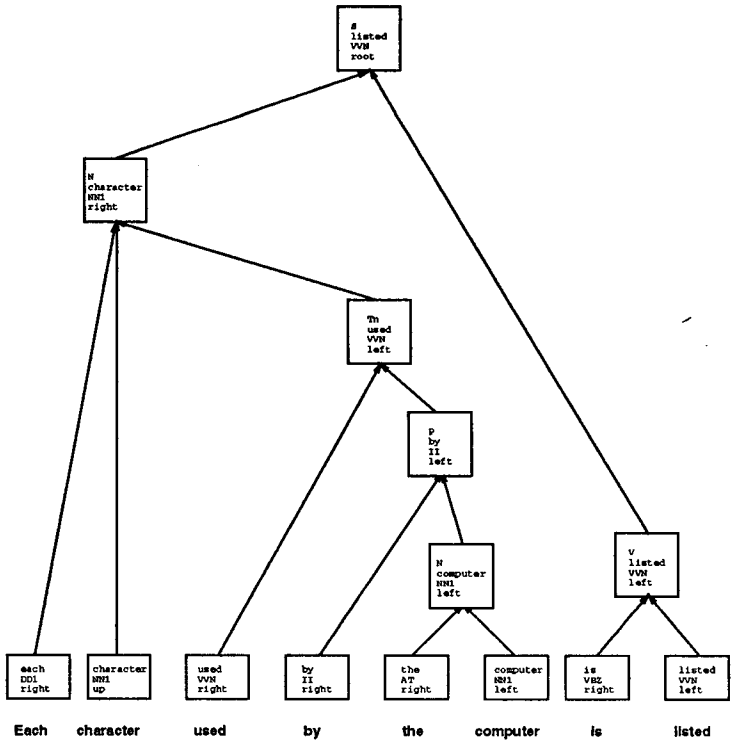

Figure 3: Treebank analysis encoded using feature values. Each internal node contains, from top to bottom, a *label, word, tag,* and *extension* value, and each leaf node contains a *word, tag,* and *extension* value.

4. Probabilistic Models for Node Features

Node Representation We do not use all the subtree information rooted at a node N to condition our probabilistic models. But rather we have an equivalence class defined by the node name (if it's available), we also have for constituent nodes, a word, along with its corresponding part-of-speech tag, that is selected from each constituent to act as a lexical representative. The lexical representative from a constituent corresponds loosely to the linguistic notion of a head word. For example, the lexical representative of a noun phrase is the rightmost noun, and the lexical representative of a verb phrase is the leftmost non-auxiliary verb. However, the correlation to linguistic theory ends there. The deterministic rules (one per label) which select the representative word from each constituent were developed in the better part of an hour, in keeping with the philosophy of avoiding excessive dependence on carefully crafted rule-based methods. Figure 3 illustrates the word and tag features propagated along the parse tree for an example sentence. Each internal node is represented as a 4-feature vector: label, head word, head tag, and extension.

Notation In the remainder of this section, the following notational scheme will be used. w_i and t_i refer to the word corresponding to the ith token in the sentence and its part-of-

speech tag, respectively. N^k refers to the 4-tuple of feature values at the kth node in the current parse state, where the nodes are numbered from left to right. N_l^k, N_w^k, N_t^k, and N_e^k refer, respectively, to the label, word, tag, and extension feature values at the node k. N^{c_j} refers to the jth child of the current node where the leftmost child is child 1. $N^{c_{-j}}$ refers to the jth child of the current node where the rightmost child is child 1. The symbol Q_{etc} refers to miscellaneous questions about the current state of the parser, such as the number of nodes in the sentence and the number of children of a particular node.

The Tagging Model The tag feature value prediction is conditioned on the two words to the left, the two words to the right, and all information at two nodes to the left and two nodes to the right.

$$p(t_i \mid context) \approx p(t_i \mid \quad w_i w_{i-1} w_{i-2} w_{i+1} w_{i+2} t_{i-1} t_{i-2} \\ t_{i+1} t_{i+2} N^{k-1} N^{k-2} N^{k+1} N^{k+2})$$

The Extension Model The extension feature value prediction is conditioned on the node information at the node being extended, all information from two nodes to the left and two nodes to the right, and the two leftmost children and the two rightmost children of the current node (these will be redundant if there are less than 4 children at a node).

$$p(N_e^k \mid context) \quad \approx p(N_e^k \mid \quad N_w^k N_t^k N_l^k N_c^k N^{k-1} N^{k-2} \\ N^{k+1} N^{k+2} N^{c_1} N^{c_2} N^{c_{-1}} N^{c_{-2}})$$

The Label Model The label feature value prediction is conditioned on questions about the presence of selected words in the constituent, all information from two nodes to the left and two nodes to the right, and the two leftmost children and the two rightmost children of the current node.

$$p(N_l^k \mid context) \approx p(N_l^k \mid \quad Q^k N^{k-1} N^{k-2} N^{k+1} N^{k+2} N^{c_1} \\ N^{c_2} N^{c_{-1}} N^{c_{-2}})$$

The Derivation Model In initial experiments, the active node selection process was modelled by a uniform ($p(active) = 1/n$) model with $n = 2$. Our intuition was that by parametrizing the choice of which active node to process, we could improve the parser by delaying labeling and extension steps when the partial parse indicates ambiguity. We used the current node information and the node information available within the five node window.

$$p(active \mid context) \approx p(active \mid Q^k N^k N^{k-1} N^{k-2} N^{k+1} N^{k+2})$$

Statistical Decision Trees The above probability distribution are each modeled as a statistical decision tree with binary questions about the history. We have described in earlier papers, [6, 4], how we use mutual information clustering of words to define a set of classes on words that form the basis of the binary questions about words in the history. We also have defined by the same mutual information on the bigram tag distribution classes for binary questions on tags. We have identified by hand a set of classes for the binary questions on the labels. The decision trees are grown using the standard methods described in [5]. In the case of hidden derivations, the forward-backward algorithms can be used to get partial counts for the different events used in building the decision trees.

5. Expectation Maximization Training

The proposed history-based model cannot be estimated by direct frequency counts because the model contains a *hidden* component: the derivation model. The order in which the treebank parse trees were constructed is not encoded in the treebank, but the parser assigns probabilities to specific derivations of a parse tree. A forward-backward (FB) algorithm can be easily defined to compute a posteriori probabilities for the states. These probabilities can then be used to define counts for the different events that are used to build the decision trees.

To train the parser, all legal derivations of a parse tree (according to the derivational window constraint) are computed. Each derivation can be viewed as a path from a common initial state, the words in the sentence, to a common final state, the completed parse tree. These derivations form a lattice of states, since different derivations of the same parse tree inevitably merge. For instance, the state created by tagging the first word in the sentence and then the second is the same state created by tagging the second word and then the first. These two derivations of this state have different probability estimates, but the state can be viewed as one state for future actions, since it represents a single history.

5.1. Decision Trees and the Forward-Backward Algorithm

Each leaf of decision tree represents the distribution of a class of histories. The parameters of these distributions can be updated using the F-B algorithm.

Initially, the models in the parser are assumed to be uniform. Accordingly, each event in each derivation contributes equally to the process which selects which questions to ask about the history in order to predict each feature value. However, the uniform model is certainly not a very good model of feature value assignments. And, since some derivations of a parse tree are better than others, the events generated by the better derivations should contribute more to the decision tree-growing process. The decision trees grown using the

uniform assumption collectively form a parsing model, M_1. The F-B count for each event in the training corpus using M_1 can be used to grow a new set of decision trees, M_2. The decision trees in M_2 are constructed in a way which gives more weight to the events which contributed most to the probability of the corpus. However, there is no guarantee that M_2 is a better model than M_1. It isn't even guaranteed that the probability of the training corpus according to M_2 is higher than the probability according to M_1. However, based on experimental results, the use of F-B counts in the construction of new decision trees is effective in acquiring a better model of the data.

There is no way of knowing, a priori, which combination of the previously mentioned applications of the forward-backward algorithm will produce the best model. After initial experimentation, the following sequence of training steps proved effective:

1. Grow initial decision trees (M_1) based on uniform models
2. Create M_2 by pruning trees in M_1 to a maximum depth of 10.
3. Grow decision trees (M_3) from F-B counts from M_2.
4. Perform F-B reestimation for leaves of decision trees in M_3.

Smoothing Decision Trees Once the leaf distributions for a set of decision trees are fixed, the model must be smoothed using held-out data to avoid overtraining on the original training corpus.

Each node in a decision tree potentially assigns a different distribution to the set of future values predicted by that tree. The problem of smoothing is to decide which combination of the distributions along a path from a leaf to the root will result in the most accurate model. The decision trees are smoothed by assigning a parameter to each node. This parameter represents the extent to which the distribution at that node should be trusted with respect to the distribution at the parent node.

6. Experimental Results

Task Domain We have chosen computer manuals as a task domain. We picked the most frequent 3000 words from 10 manuals as our vocabulary. We then extracted about 35,000 sentences covered by this vocabulary[3] from 40,000,000 words of computer manuals. This corpus was treebanked by the University of Lancaster. The Treebank uses 17 non-terminal labels and 240 tags[4].

[3] the actual vocabulary is around 7,000 words when we include the many symbols, formulas, and numbers that occur in the manuals

[4] we have projected the tag set to 193

Length	# of Sentences	Words/ Sentence	Constituent/ Sentence
1-10	447	7.5	4.6
1-15	883	10.3	6.4
1-23	1313	13.1	8.0
1-30	1507	14.9	8.9
1-∞	1656	16.9	10.1

Table 1: Distribution of sentences, average words/sentence, and average number of non-terminals per sentence for the blind test set.

A parse produced by the parser is judged to be correct under the "Exact Match" criterion if it agrees with the Treebank parse structurally and all NT labels and tags agree.[5]

Experiment 1 The parser using a stack decoding search which produced 1 parse for each sentence, and this parse was compared to the treebank parse for that sentence. On this test set, the parser produced the correct parse, i.e. a parse which matched the treebank parse exactly, for 38% of the sentences. Ignoring part-of-speech tagging errors, it produced the correct parse tree for 47% of the sentences. Further, the correct parse tree is present in the top 20 parses produced by the parser for 64% of the sentences.

No other parsers have reported results on exactly matching treebank parses, so we also evaluated on the *crossing brackets* measure from [2], which represents the percentage of sentences for which none of the constituents in a parser's analysis violate the constituent boundaries of the treebank parse. The crossing-brackets measure is a very weak measure of parsing accuracy, since it does not verify prepositional phrase attachment or any other decision which is indicated by omitting structure. However, based on analysis of parsing errors, in the current state-of-the-art, increases in the crossing brackets measure appear to correlated with improvements in overall parsing performance. This may not remain true as parsers become more accurate.

The 1100 sentence corpus that we used in this first experiment was one of the test corpora used in several experiments reported in [2]. The grammar-based parser discussed in [2] uses a P-CFG based on a rule-based grammar developed by a grammarian by examining the same training set used above over a period of more than 3 years. This P-CFG parser produced parses which passed the crossing brackets test for 69% of the 1100 sentences. Our decision tree hidden derivation parser improves upon this result, passing the crossing brackets test for 78% of the sentences. The details of this experiment are discussed in [9].

[5] A sample of 5000 sentences (a training set of 4000, a development test of 500, and an evaluation test of 500) is available by request from roukos@watson.ibm.com.

Length	Treebank Consistency	Exact Match	top 20	Crossing Bracket
1-10	69.1%	55.9%	80.8%	91.5%
1-15	64.9%	51.7%	78.7%	86.2%
1-23	58.3%	41.9%	68.9%	76.5%
1-30	—	38.1%	64.0%	70.9%
1-∞	52.5%	34.9%	59.1%	65.7%

Table 2: Performance of leftmost bottom-up derivation for Computer Manuals.

# Sentences in Training Data	Exact Match	Top 20
15000	34.2	61.1
20000	37.4	64.8
25000	37	67.7
30000	38.1	68.4
34000	38.9	73

Table 3: Performance as a function of Training Set Size

Experiment 2 By using a derivation window of 1, we find that Exact Match accuracy decreases by two percentage points with a significant reduction in computational complexity. Using the simpler single derivation model, we built a new set of models. We also combined the naming and extension steps into one, improved some of our processing of the casing of words, and added a few additional questions. Using these models, we ran on all sentences in our blind test set. Table 1 gives some statistics a function of sentence length on our test set of 1656 sentences. Table 2 gives the parser's performance [6]. In Table 2, we show a measure of treebank consistency. During treebanking, a random sample of about 1000 sentences was treebanked by two treebankers. The percentage of sentences for which they both produce the same exact trees (tags included) is shown as Treebank Consistency in Table 2. We also show the percentage of sentences that match the Treebank, the percentage where the Treebank parse is among the top 20 parses produced by the parser, and the percentage of sentences without a crossing bracket. Currently, the parser parses every third sentence exactly as a treebanker and is about 15 percentage points below what the treebankers agree on when they are parsing in production mode. A more carefully treebanked test set may be necessary in the future as we improve our parser.

We also explored the effect of training set size on parsing performance with an earlier version of the parsing model. Table 3 shows the Exact Match score for sentences of 23 words or less. From this data, we see that we have a small improvement in accuracy by doubling the training set size from 15k to 30k sentences.

7. Conclusion

We presented a "linguistically" naive parsing model that has a parsing accuracy rate that we believe is state-of-the-art. We anticipate that by refining the "linguistic" features that can be examined by the decision trees, we can improve the parser's performance significantly. Of particular interest are linguistic features that may be helpful in conjunction and other long distance dependency. We are currently investigating some mehtods for building in some of these features.

[6] While we prefer to use Exact Match for automatic parsing, we computed the PARSEVAL performance measures to be: 80% Recall, 81% Precision, and 10% Crossing Brackets on the unseen test set of Experiment 2. Note: On this test set, 65.7% of the sentences are parsed without any crossing brackets.

Acknowledgement

We wish to thank Robert T. Ward for his measurements of Treebank consistency. This work was supported in part by ARPA under ONR contract No. N00014-92-C-0189.

References

1. Baker, J. K., 1975. Stochastic Modeling for Automatic Speech Understanding. In Speech Recognition, edited by Raj Reddy, Academic Press, pp. 521-542.
2. Black, E., Garside, R., and Leech, G., 1993. *Statistically-driven Computer Grammars of English: The IBM/Lancaster Approach.* Rodopi. Atlanta, Georgia.
3. Black, E., Lafferty, J., and Roukos, S., 1992. Development and Evaluation of a Broad-Coverage Probabilistic Grammar of English-Language Computer Manuals. In Proceedings of the Association for Computational Linguistics, 1992. Newark, Delaware.
4. Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., and Roukos, S., 1993. Towards History-based Grammars: Using Richer Models for Probabilistic Parsing. In Proceedings of the Association for Computational Linguistics, 1993. Columbus, Ohio.
5. Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J., 1984. *Classification and Regression Trees.* Wadsworth and Brooks. Pacific Grove, California.
6. Brown, P. F., Della Pietra, V. J., deSouza, P. V., Lai, J. C., and Mercer, R. L. Class-based n-gram Models of Natural Language. In *Proceedings of the IBM Natural Language ITL,* March, 1990. Paris, France.
7. Magerman, D. M. and Marcus, M. P. 1991. Pearl: A Probabilistic Chart Parser. In Proceedings of the February 1991 DARPA Speech and Natural Language Workshop. Asilomar, California.
8. Magerman, D. M. and Weir, C. 1992. Efficiency, Robustness, and Accuracy in Picky Chart Parsing. In Proceedings of the Association for Computational Linguistics, 1992. Newark, Delaware.
9. Magerman, D., 1994. Natural Language Parsing as Statistical Pattern Recognition. Ph. D. dissertation, Stanford University, California.
10. Sharman, R. A., Jelinek, F., and Mercer, R. 1990. Generating a Grammar for Statistical Training. In Proceedings of the June 1990 DARPA Speech and Natural Language Workshop. Hidden Valley, Pennsylvania.