Inductive Learning of Abstract Role Values Derived from a Constraint Dependency Grammar

Christopher M. White, Mary P. Harper, Terran Lane, and Randall A. Helzerman
{robot, harper, terran, helz}@ecn.purdue.edu
School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University,
West Lafayette, IN 47907

Abstract

Research in the machine learning of the structure of natural language has lead some researchers to look at the PAC learnability of abstract role values (Helzerman & Harper 1997), which are a feature vector representation of constraints. This research has shown that constraints that can be specified by Constraint Dependency Grammar (CDG) are Probably Approximately Correct (PAC) learnable from positive examples. However, the simple PAC-learning algorithm for CDG is too inefficient to be used for large natural language grammars. We show that it is possible to use an inductive logic program, namely C4.5 (Quinlan 1993), to induce decision trees to classify abstract role values as being either positive or negative instances. We further show that both grammatical and ungrammatical training sentences help to specify the space of abstract role values for a grammar.

Descriptive Terms: Machine Learning, Natural Language Processing, Constraint Dependency Grammars, Abstract Role Values, Induction, Decision Trees

Thoughts

Research in the machine learning of the structure of natural language has lead some researchers to look at the PAC learnability of abstract role values (Helzerman & Harper 1997), which are a feature vector representation of constraints. This research shows that constraints that can be specified by Constraint Dependency Grammar (CDG) are Probably Approximately Correct (PAC) learnable from positive examples. However, the simple PAC-learning algorithm for CDG is too inefficient to be used for large natural language grammars. The purpose of our research is to investigate the ability of an inductive logic program, namely C4.5 (Quinlan 1993), to induce decision trees to classify abstract role values as being either positive or negative instances. These abstract role values are provided by parsing sentences in preexisting grammars. This approach provides us with a convenient way to check the validity of the resulting learned grammar by simply comparing the output of the parser on the original and learned grammars. However, once it is shown that the abstract role values can be learned through inductive logic approaches, future experiments can focus on techniques that do not require a previously existing grammar.

Figure 1 shows the general steps that are needed to learn the abstract role values of a grammar. The basic idea is to use a set of training sentences to generate abstract role values that are then used as the training sets to learn the abstract role value space. The decision trees generated by the inductive logic learner will then be used by the parser to generate parses for an input sentence.

Figure 2 shows the two step process that we used for generating abstract role values for the second and third experiments:[1] however, any technique for generating the abstract role values could be used. We use a modified version of PARSEC to accomplish the task of generating the abstract role values. This requires training sentences and a complete CDG, which includes a grammar table, unary constraints, and binary constraints. When there is a preexisting Context-Free Grammar (CFG), it is possible to obtain the CDG by passing the CFG through the CFG-to-CDG-converter as described by White (White 1995) (This paper is currently available at *ftp://transform.ecn.purdue.edu/pub/speech/papers/thesis.ps.gz*). However, if a CDG already exists, this conversion step can be skipped.

Sketches

Before setting out on the investigation of using C4.5 to classify abstract role values, it is necessary to choose some well defined CDGs upon which to do the testing. We chose three grammars for our initial experiments. The first grammar accepts strings in the language $a^n b^n c^n$. The second grammar accepts simple sentences in the form of statements, and the third grammar accepts a subset of twenty sentence types from the Naval Resource (Battleship) Management

[1] A preexisting CDG was used for the first experiment. In this case, it was possible to skip the CFG-to-CDG conversion step.

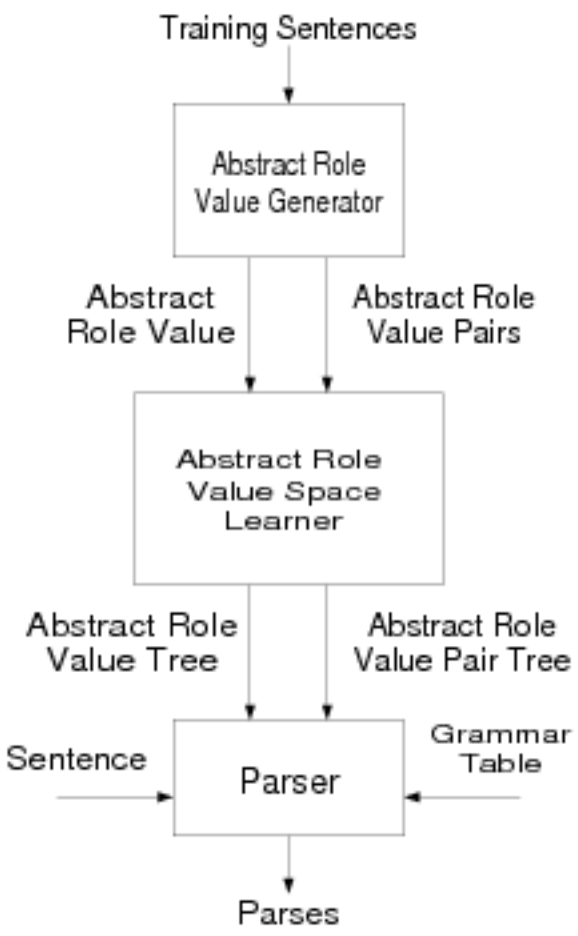

Figure 1: Experiment overview.

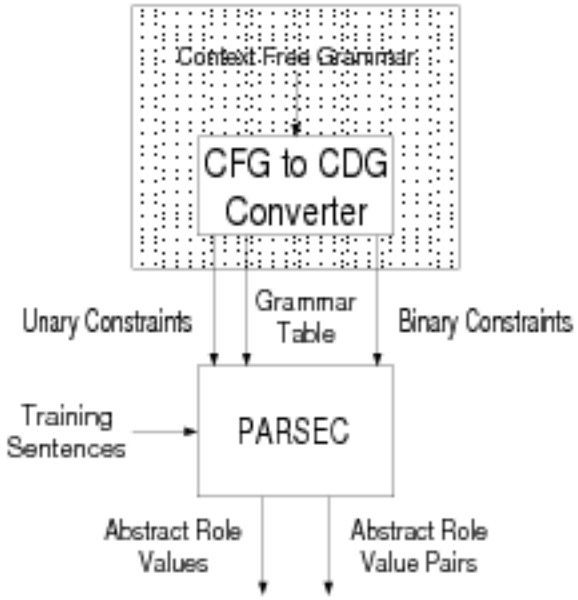

Figure 2: An Abstract Role Value Generator.

Task (NRMT)–as defined within the DARPA community (Price *et al.* 1988). Both the second and third grammars were obtained by converting previously built CFGs which had been tested using our Common Lisp version of a Tomita parser (Tomita 1987). These grammars were converted from CFG to CDG form using the conversion tool defined by White (White 1995) in order to generate grammars consistent with our CDG parser, PARSEC (Parallel ARchitecture SEntence Constrainer) (Harper *et al.* 1995).

Given the CDG constraints for each grammar, it is necessary to develop a method for producing positive and negative example instances (abstract role values) for training the decision trees for both the unary and binary constraints. Each example instance is labeled as either a positive or negative instance based on whether the information in the instance continues to support a possible parse for the target sentence. Due to the large number of possible sentence structures in English (and many other languages), one target sentence is not sufficient to explore the entire space of instances that may be defined by the grammar (the abstract role value space). Thus, it is necessary to repeat the process for many structurally different sentences in order to explore a large portion of the instance space. Once a good representation of the instance space is obtained, two decision trees are induced. One decision tree represents the learned instance space for the unary constraints and the second decision tree represents the learned instance space for the binary constraints. Finally, these decision trees are tested by substituting them for the original constraints within PARSEC.

Having outlined the experiments, we will now review the details of some of the supporting research.

CDG Parsing

Helzerman and Harper (Helzerman & Harper 1997) illustrated that any function over role values that can be specified by CDG constraints are PAC learnable. A CDG, as defined by Harper and Helzerman (Harper & Helzerman 1995a), is defined as a four-tuple, $\langle \Sigma, R, L, C \rangle$, where:

Σ = $\{\sigma_1, \ldots, \sigma_d\}$, a finite set of lexical categories.
R = $\{r_1, \ldots, r_p\}$, a finite set of uniquely named roles (role-ids).
L = $\{l_1, \ldots, l_q\}$, a finite set of labels.
C = finite constraint set that an assignment **A** must satisfy.

A *sentence* $s = w_1 w_2 w_3 \ldots w_n$ is a string of finite length n and is an element of Σ^*. All p roles in R are associated with every w_i of s, yielding $n * p$ roles for the entire sentence. The sentence s is said to be *generated* by the grammar G if there exists an assignment **A** that maps role values to each of the $n * p$ roles for s such that the constraint set C is satisfied. A *role value* is an element of the set $\Sigma \times R \times L \times \{\text{nil}, 1, 2, \ldots\}$. In other words, it is a 4-tuple consisting of a category

from Σ, a role from R, a label from L, and a modifiee, where the modifiee can be the index of a word in the sentence or nil. Role values are usually abbreviated as *label-modifiee* when it is obvious from context what their categories and roles are. The *language* $L(G)$ is the set of all sentences generated by G. Note that the null string ϵ has no roles and is always generated by any grammar by definition.

A *constraint set* is a logical formula of the form:

$$\forall x_1, x_2, \ldots x_a \ (\text{and } \mathrm{P}_1 \ \mathrm{P}_2 \ \ldots \mathrm{P}_m).$$

where each x_i is a variable ranging over all of the role values in each of the roles of each of the words in s. Each subformula P_i in C must be of the form (if *Antecedent Consequent*), where *Antecedent* and *Consequent* are predicates or predicates joined by the logical connectives. Below are the basic components used in forming a constraint:

- **Variables:** x_1, x_2, ... x_a ($a = 2$ in (Maruyama 1990)).
- **Constants:** elements and subsets of $\Sigma \cup L \cup R \cup \{$nil, 1, 2, ..., $n\}$, where n corresponds to the number of words in a sentence.
- **Access Functions:**
  (pos x) returns the position of the word for role value x.
  (rid x) returns the role-id for role value x.
  (lab x) returns the label for role value x.
  (mod x) returns the position of the modifiee for role value x.
  (cat x) returns the category (i.e., the element in Σ) for the role value x.[2]
- **Predicate Symbols:**
  (= x y) returns true if x = y.
  (> x y) returns true if x > y and x, y $\in$ Integers.[3]
  (< x y) returns true if x < y and x, y $\in$ Integers.
- **Logical Connectives:**
  (and p q) returns true if subformulas p and q are true.
  (or p q) returns true if subformula p or q is true.
  (not p) returns true if subformula p is false.

A CDG has two associated parameters: *degree* and *arity*. The degree of a grammar G is the size of R, the set of role-ids. The arity of the grammar corresponds to the maximum number of variables in the subformulas of C.

Figure 3 depicts the process of parsing the sentence *The dog eats* with the following grammar, which has a degree of 1 and an arity of 2:

$\Sigma = \{$noun, verb, det$\}$,
$R = \{$governor (depicted as **G**)$\}$,
$L = \{$DET, SUBJ, ROOT$\}$,
$C =$ see the constraints in Figure 3.

[2] Maruyama does not associate this information with the role value, but to handle lexical ambiguity, this is quite important (see (Harper & Helzerman 1995a)).

[3] For example, (gt 1 nil) is false, because nil is not an integer.

Initially, each of the roles of a word are assigned the set of all role values which are possible for that word. The initial constraint network for the sentence *The dog eats* appears at the top of Figure 3. Since the sentence has three words, the possible labels and modifiees for the role values assigned to each role for each word category in the sentence will be $\{$DET, SUBJ, ROOT$\} \times \{$nil, 1, 2, 3$\} = \{$DET-nil, DET-1, DET-2, DET-3, SUBJ-nil, SUBJ-1, SUBJ-2, SUBJ-3, ROOT-nil, ROOT-1, ROOT-2, ROOT-3$\}$. Because a word cannot modify itself, the set of role values which is initially assigned to each of the roles in the sentence will vary, depending upon the position of the role's word. For example, the governor role for the first word *The* in the example sentence will not be assigned any role values whose modifiee is 1. Therefore, it is assigned the set $\{$DET-nil, DET-2, DET-3, SUBJ-nil, SUBJ-2, SUBJ-3, ROOT-nil, ROOT-2, ROOT-3$\}$. At the end of parsing, the role values will encode the parse relations that are allowed by the grammar constraints.

Unary constraints contain a single variable and test whether role values are legal. A role value is incompatible with a unary constraint iff it satisfies the antecedent, but not the consequent. For example in Figure 3, all of the role values associated with the governor role of *The* satisfy the antecedent of the first unary constraint, but DET-nil, SUBJ-nil, SUBJ-2, SUBJ-3, ROOT-nil, ROOT-2, and ROOT-3 violate the consequent (either because they have an incompatible label or their modifiee is not greater than their position); hence, they are incompatible with the constraints. When a role value violates a unary constraint, node consistency eliminates the role value from its role because it can never participate in a parse for the sentence.

Next, binary constraints are enforced. Binary constraints determine which pairs of role values can legally coexist. A pair of role values is incompatible with a binary constraint iff the pair satisfy the antecedent, but not the consequent. To keep track of pairs of role values, *arcs* connecting each role to all other roles in the network are constructed, and each arc has an associated *arc matrix*, whose row and column indices are the role values associated with the two roles it connects. The entries in an arc matrix can either be a **1** (indicating that the two role values indexing the entry are compatible) or a **0** (indicating that the role values cannot simultaneously exist). Initially, all entries in each matrix are set to **1**. The binary constraint shown in Figure 3 is applied to the pairs of role values indexing the entries in the matrices. In this case when x = DET-3 for *the* and y = ROOT-nil for *eats*, the consequent of the binary constraint fails; hence, the role values are incompatible. This is indicated by replacing the entry of **1** with **0**.

Finally, arc consistency eliminates role values that are inconsistent with all the role values associated with at least one other role (Mackworth 1977; Mohr & Hen-