The Application of Semantic Classification Trees to Natural Language Understanding

Roland Kuhn, Member, IEEE, and Renato De Mori, Fellow, IEEE

Abstract—This article describes a new method for building a natural language understanding (NLU) system, in which the system's rules are learnt automatically from training data. The method has been applied to design of a speech understanding (SU) system. Designers of such systems rely increasingly on robust matchers to perform the task of extracting meaning from one or several word sequence hypotheses generated by a speech recognizer; a robust matcher processes semantically important islands of words and constituents rather than attempting to parse the entire word sequence. We describe a new data structure, the Semantic Classification Tree (SCT), that learns semantic rules from training data and can be a building block for robust matchers for NLU tasks. By reducing the need for handcoding and debugging a large number of rules, this approach facilitates rapid construction of an NLU system. In the case of an SU system, the rules learned by an SCT are highly resistant to errors by the speaker or by the speech recognizer because they depend on a small number of words in each utterance. Our work shows that semantic rules can be learned automatically from training data, yielding successful NLU for a realistic application.

Index Items—Speech understanding, semantic classification tree, SCT, machine learning, natural language, decision tree.

I. INTRODUCTION

This article describes a new method for building a natural language understanding (NLU) system, in which the system's rules for semantic interpretation are learnt automatically from training data. The rules are encoded in a forest of specialized decision trees called Semantic Classification Trees (SCTs). Application of the SCT method to a speech understanding (SU) task showed that the learned rules are robust to grammatical or lexical errors in the input. By eliminating the need to code and debug a large number of such rules by hand, the methodology facilitates rapid construction of NLU systems for which annotated corpora are available.

SCTs are building blocks that can be used in a variety of ways by a system designer, subject only to the availability of training data; for instance, they can be used in conjunction with a conventional parser. In our particular application, a word string was preprocessed by a bottom-up parser that recognized and labeled certain semantically important phrase constituents while leaving most of the string unchanged [17], [21]. The partly-parsed word string then passed through a forest of SCTs, each of which generated a different aspect of the representation by recognizing patterns made up of constituents, other words, and gaps.

In contrast with a structural pattern recognition approach [12], SCTs make a decision about new patterns on the basis of statistical classification rules learned from patterns seen earlier. These rules involve "gaps": words and groups of words that are ignored. Operations performed on gaps lie at the heart of the SCT-growing algorithms. Several existing parsers use hand-coded rules that involve gaps - in fact, this was our original reason for believing that rules involving gaps might be worth learning (see Section II.A.). However, we know of no other technique (either semantically or syntactically based) that learns rules involving gaps from training data; the grammatical inference literature focuses on production rather than comprehension, and thus assumes implicitly that the goal is to learn rules that account for all symbols in a given string [11]. A focus on gaps and on classification rather than on grammatical inference makes it much easier to learn rules for understanding sentences, though it would be inappropriate for learning rules for producing sentences.

SCTs have the following properties:

- They learn rules for classifying new strings or substrings from a corpus of classified strings or substrings. To apply SCTs to a problem, one must formulate it as a classification problem.
- The questions in the nodes of an SCT involve regular expressions made up of string symbols and a special gap symbol. The string symbols could be words or higher level constituents.
- Generation and selection of questions is completely automatic: any symbol from the symbol lexicon may appear in a question.
- There are two main types of SCT: single-symbol SCTs and set-membership SCTs. Either type can be employed to classify a whole string, or to classify substrings within a string.

II. RELATED WORK

A. Speech Understanding Systems

Every SU system contains a linguistic analyzer that translates word sequence hypotheses generated by the speech rec-
ognition component into a semantic representation. In many SU systems, the linguistic analyzer was built around strict syntactic rules. Word sequences that disobeyed the rules were discarded during the recognition process, so that an incoming utterance could yield only two outcomes: failure or a parse for a complete sequence of words. Unfortunately, many spoken sentences are meaningful but ungrammatical.

A growing number of SU systems rely on robust matching to handle ungrammatical utterances. The robust matcher tries to fill slots in a frame without attempting a sentence-level parse; it skips over words or phrases that do not help it to fill a slot or to decide on the identity of the current frame. The slot-filling phrases themselves still undergo syntactic parsing. Because it does not attempt to generate a parse tree incorporating every word in the utterance, the robust matcher can handle interjections, restarts, incomplete sentences, and many other phenomena typical of speech.

Two of the robust matchers in the literature are the Template Matcher (TM) at the Stanford Research Institute [15] and Phoenix at Carnegie Mellon University [29], [30], [31]. The input to both systems is the top hypothesis produced by the speech recognition component. TM instantiates competing templates, each of which seeks to fill its slots with appropriate words and phrases in the utterance; the template with the highest score yields the semantic representation. Recent improvements have included incorporation of a bottom-up unification-based parser called Gemini. CMU's Phoenix was built in the Recursive Transition Network formalism; word patterns correspond to semantic tokens, some of which appear as slots in frame structures. The system fills slots in different frames in parallel, using a form of dynamic programming beam search; the score for a frame is the number of input words it accounts for.

Other linguistic analyzers, such as MIT's TINA, have also moved away from a purely syntax-driven approach to one that incorporates a strong element of robust matching [27]. Such systems often employ syntax locally and semantics and pragmatics globally. For instance, sentences may be preprocessed by a local parser that brackets and labels important phrases (e.g., NPs and PPs) before they are submitted to a robust matcher. This represents a genuine linguistic discovery: spoken utterances are often made up of islands of syntactically correct phrases separated by verbal "noise," with weak or non-existent global syntactic constraints. In fact, it is arguable that no adequate syntactic theory for spoken English exists. In this view, existing theories are biased by the influence of written English; this influence is reinforced by the Chomskian preference for studying "competence" rather than "performance." SU researchers may have stumbled on phenomena that should be of compelling interest to theoretical linguists.

Recently, some SU researchers have begun to explore linguistic analyzers whose rules are learnt from training data. Among SU groups participating in the ATIS benchmarks in November 1992 (see Sections IV.B. and V.B.), only AT&T and ourselves entered analyzers of this type. AT&T's analyzer, Chronus, carries out unsupervised learning; word sequences are modeled by Hidden Markov Models (HMMs), with the words being the observations and the concepts being the states [22], [23], [24]. The goal is to find the sequence of words \( W \) and the sequence of concepts \( C \) maximizing \( P(W, C|A) \), where \( A \) is the acoustic evidence. The main difference between our linguistic analyzer, CHANEL (described in Section V.) and Chronus is that Chronus carries out a one-to-one mapping between sentence segments and concepts. Each of the SCTs in CHANEL builds part of the semantic representation, and looks at the entire utterance in order to do so. This permits a given word or phrase to contribute to more than one "concept," and also permits words or phrases that are far apart from each other to contribute to the same concept.

In the December 1993 ATIS benchmarks, AT&T chose to use CMU's Phoenix as the linguistic analyzer, rather than Chronus. For these benchmarks, BBN for the first time entered a system, HUM, with learnt rules. The December 1993 version of HUM was only a few months old, a first approximation to the ultimate system. Meaning is represented as a tree structure in which subconcepts are nested within other concepts. A context-free statistical model is nested within other concepts. A context-free statistical model for HUM was available [20].

### B. Applications of Decision Trees to Natural Language

- The pioneering work on the application of decision trees to natural language was carried out at IBM [1]. The task domain was probabilistic language modeling: estimating the probability \( P(w) \) that a word \( w \) will occur, given the history provided by previous words. Bahl et al. employed questions that ask about the identity of a word preceding the current word \( w_j \) by a fixed number of positions—for instance, such a question might ask whether word \( w_{j-5} \) is a month—and also combined these simple questions to obtain compound questions. In practice our approach and the IBM approach have been applied to different problems, but they could be interchanged: SCTs could be adapted to language modeling, and IBM trees could be adapted to learning semantic rules. The principal difference between the two approaches is that the questions in the IBM trees give exact distances between words, while SCT questions specify word order but involve gaps of unspecified nonzero length.

- IBM researchers have also built "History-Based Grammars" based on decision trees that give the name of the parsing rule to be applied next [2].

- In other recent IBM work, decision trees were employed to estimate the probability that a word should be tagged with a certain part of speech, given the surrounding text [4].

- Decision trees were trained to guess the past participles of verbs given the present participle [19]. The paper describes a problem also encountered by us, that of mapping a set of input attributes onto a set of output attributes, and solves it in the same way: by creating a forest of trees, where "each tree takes as input the set of all attributes in the input patterns, but is concerned with determining the value of one attribute in its output pattern" [19].

- Decision trees were applied to the task of partitioning...
newspaper articles into classes [7]. Questions in the tree ask whether a particular word \( w_i \) or set of words \( \{w_i\} \) occurs or does not occur in an article. These trees are simplified SCTs that ignore word order, and thus consider a small subset of the questions that would be considered by SCT-growing algorithms.

III. BUILDING SEMANTIC CLASSIFICATION TREES (SCTs)

A. Introduction

An SCT is a specialized classification tree that learns semantic rules. To grow classification trees, one must supply three elements [5]:

1) A set of possible yes-no questions that can be applied to data items;
2) A rule for selecting the best question at any node on the basis of training data;
3) A method for pruning trees to prevent over-training.

In our application, a data item is a symbol (usually word) sequence. The original aspect of SCTs is the way in which the set of possible questions is generated. These questions ask whether a word sequence matches certain regular expressions involving words and gaps.

To choose a question from this set, we use the Gini “impurity” \( i(T) \) of a node \( T \) [5]. The best question for \( T \) is considered to be the question which brings about the greatest drop in impurity in going from \( T \) to its children. If the two children of \( T \) are denoted YES and NO, and the proportions of items at \( T \) that a question will send to the YES and NO children are denoted \( p_y \) and \( p_n \) respectively, consider the change in impurity defined as

\[
\Delta I = i(T) - p_y i(YES) - p_n i(NO).
\]

The question chosen at node \( T \) will be a question that maximizes \( \Delta I \).

To prevent over-trained trees, we use the algorithm described in [13], which involves iterative cycles of expansion and pruning on two equal-sized disjoint sets of training data.

B. Single-Symbol SCTs

Fig. 1 shows an example of a single-symbol SCT grown on NL data for the ATIS task (see Section IV.B.). Its job is to decide whether a request should result in showing the user the attribute fare.fare_id (found in an air-travel database) or not; sentences that end up in a YES leaf will have fare.fare_id in their “displayed attribute list,” used in generation of an SQL query. The symbols “<” and “>” match the start and end of a sentence, a “*” between two words or symbols indicates a gap of at least one word between them, and the expression “M(w)” (e.g. “M(fares)” in the figure) matches one or more occurrences of the word \( w \); order matters. For instance, the input “Show me first-class fare flights to Boston” matches the pattern \(<+ fare flights +>\) at the root, does not match \(<+ fare code +>\), and does match the pattern \(<+ fare flights +>\)—it thus yields “NO”. “Show me the fare for flights to Boston” matches the root expression but no other expression it encounters, and thus yields “YES” (it does not match \(<+ fare flights +>\) because “for” comes between “fare” and “flights”).

Fig. 2(a)-(c) give a version of the algorithm for growing a single-symbol SCT that assumes words are not repeated in a sentence (the version that handles repetitions differs only slightly, but requires a lengthier explanation; see [18] for details). The SCT is grown by calling the function grow_SCT(), shown at the top of Fig. 2(a), on a collection of strings in which each string is associated with a class label. The strings are made up of words listed in a vocabulary \( V \), where a “word” could be an ordinary word or a symbol representing a constituent.

Each node of the growing single-symbol SCT is associated with a regular expression called the Known Structure consisting of symbols and gaps (denoted “*”); the set of possible questions is generated by operations on the gaps. The known structure for the root of the SCT is \(<+>\); strings entering the root must have length at least one. Expressions of the form \( A := B[u \leftrightarrow v] \) in Fig. 2(b) mean that the string \( A \) is a copy of the string \( B \) in which substring \( u \) has been replaced by substring \( v \).

The expansion phase of the expansion-pruning algorithm employs tolerant stopping rules that encourage growth of a large tree. A SCT node is declared to be a leaf node when no further split is possible. This may happen because all training items in the node belong to the same category, because there are no gaps + in the known structure for the node and thus no questions to be asked, or because none of the possible questions give a split that reduces the Gini impurity.

The original part of the algorithm is contained in the function find_question(). The four expressions generated from a
function grow_SCT (datasettype): nodeptr;
  var root: nodeptr;
  begin
    create (root);
    root.KS := identitystring;
    root.data := datasettype;
    expand_node (root);
    return (root);
  end; (end of grow_SCT())

procedure expand_node (var node: nodeptr);
  begin
    if (stop_cond(node)) then
      set_to_leaf(node);
    else begin
      node.quest := find_question (node);
      create (node.YES);
      create (node.NO);
      node.YES.data := YES_data (node);
      node.NO.data := NO_data (node);
      (* YES_data(), *NO_data()*) return data
      yielding "YES" and "NO" to node.quest
      node.YES.KS := node.quest;
      node.NO.KS := node.KS;
      NOTE: "node.KS" is known structure of
      "node"
      expand_node (node.YES);
      expand_node (node.NO);
    end; (end of expand_node())

function find_question (node: nodeptr): question;
  {returns question maximizing Gini criterion}
  var best_Q, temp_Q: question;
  var best_G, temp_G: real;
  begin
    best_G := -MAX;  {MAX is largest system-defined number}
    for every '+' in node.KS do begin
      for every word w in V do begin
        temp_Q := node.KS [+' e= w];
        temp_G := A Gini (temp_Q, node.data);
        if (temp_G > best_G) then begin
          best_Q := temp_Q;
          best_G := temp_G;
        end;
      end;
    end; (end of find_question())

Fig. 2(a). Growing single-symbol SCTs (start).

Fig. 2(b). Growing single-symbol SCTs (continuation).

given gap + (denoted +) in the known structure KS and a given
lexical item w_i are:

1) The expression obtained by replacing + in KS by w_i;
2) The expression obtained by replacing + in KS by w_i +;
3) The expression obtained by replacing + in KS by + w_i;
4) The expression obtained by replacing + in KS by + w_i +.

At the root, whose known structure is < + >, these four gap
operations generate the expressions < w_i >, < w_i + >, < + w_i >,
and < + w_i + >. Each of these expressions E is turned into a
potential question by asking: "Does the sequence being classified
match the expression E?" If there are V symbols in the lexicon, 4
* V questions are generated by allowing w_i to be any of them.
From these, the algorithm selects the one which achieves the best
split of the training data. As the tree grows, known structures get
longer. If the question “does the sequence match < +w_i + >?” is
selected to fill the root, the known structure for the root’s YES
child is < +w_i + >, and the known structure for the root’s NO
child is < + >. New questions are generated by applying the
four gap operations to each + individually.

C. Set-Membership SCTs

Single-symbol SCTs tend to have more nodes in NO than in
YES subtrees (because a particular sentence will probably not
contain a given word from the lexicon). This implies that sen-
tences that differ only by a synonym will be assigned to differ-
ent subtrees by a single-symbol SCT.

A set-membership question asks about the presence or ab-
sence of any member of a set of words at a given position, thus
allowing system-defined synonyms. Word sets are inserted into
KUHN AND DE MORI: THE APPLICATION OF SEMANTIC CLASSIFICATION TREES

procedure set_to_leaf (var node:nodeptr);
begin
  makes this leaf node - label is set to most common label in node.data
end; (set_to_leaf ()

function _YES data (node:nodeptr): datasettype;
var YES_set: datasettype;
begin
  YES_set := \);
  for every item in node.data do
    if node.quest(item) gives "YES" then
      add item to YES_set;
  return(YES_set);
end; {YES-data ()

function NO_data (node:nodeptr):datasettype;
var NO_set: datasettype;
begin
  NO_set := \);
  for every item in node.data do
    if node.quest(item) gives "NO" then
      add item to NO_set;
  return(NO_set);
end; {NO-data ()

function \( A \) Gini (Q:question, D:datatype): real;
begin
  returns decrease in Gini impurity when split D with question Q
end; (A Gini ()

Fig. 2(c). Growing single-symbol SCTs (final).

the gaps + in the known structure, similar to what happens with individual symbols in the single-symbol SCT. If \( S \) stands for an automatically generated word set, the questions considered at a given + are obtained by replacing it with \( S, S^+, + S, +s+ \).

Several different methods for generating \( S \) were considered. Ultimately, a simple heuristic was adopted (see [18] for details). For each + and each of the four question types listed above, the set \( S \) is initialized to contain the word yielding the best single-symbol question (according to the Gini criterion). Next, a word is chosen from the lexicon so that when it is added to \( S \), the new question diminishes Gini impurity as much as possible. The algorithm iterates until no word from the lexicon, when added to \( S \), reduces the impurity any further. From all set-membership questions generated in this manner, the question yielding the greatest reduction in Gini impurity is placed in the current SCT node.

Fig. 3 shows part of a set-membership SCT grown to make the same decision as the single-symbol SCT in Fig. 1: whether fare.fare_id should be displayed to the user. Here, the symbol \( M(S) \) implies that one or more words in the set \( S \) must occur within a given interval; \( I(S) \) implies that exactly one word in the set \( S \) occurs within a given interval.

Fig. 3. Set-membership SCT for fare.fare_id (9 nodes).

D. Classifying Substrings

The algorithms for growing SCTs (either single-symbol or set-membership) can be adapted to the task of classifying substrings. An example from ATIS will illustrate this case. Consider the sentence "show me flights from Boston no sorry from New York to Chicago stopping over in Pittsburgh." After local parsing, this would be "show me flights from CIT no sorry from CIT to CIT stopping over in CIT." The CITs should be labeled as follows: "show me flights from CIT \( \Leftrightarrow \) SCRAP no sorry from CIT \( \Leftrightarrow \) ORI to CIT \( \Leftrightarrow \) DEST stopping over in CIT \( \Leftrightarrow \) STOP," where "ORI" is flight origin, "DEST" is destination, "STOP" is stopover, and "SCRAP" means the CIT is irrelevant.

The SCT-growing algorithms described above require only minor modification to grow SCTs that classify parts of strings. The key is to submit the same sentence to an SCT as many times as there are substrings to be classified. Each time, the substring being classified is marked with a special symbol such as '*'. Fig. 4 shows a single-symbol SCT for classifying CIT substrings.

To classify the first CIT in the sentence above, we submit "show me flights from *CIT no sorry from CIT to CIT stopping over in CIT " to the SCT. The SCT will label this (and thus the first CIT) "SCRAP." To classify the second city, we submit the sentence with the '*' in front of the second CIT, and so on. How can an SCT for classifying substrings be grown from training data? Note that certain nodes in Fig. 4 including the root, are shaded. These nodes are the compulsory backbone of any SCT for classifying CIT substrings. The growth of such an SCT begins from the compulsory backbone already specified by the system designer; otherwise, the SCT-growing algorithm is exactly the same.
Let \( D \) be the total number of sentences in the training data, \( L \) the upper limit on the number of words in a sentence, and \( V \) the size of the vocabulary. The worst-case time analysis for both single-symbol and set-membership SCTs depends on whether the supply of training data or the supply of possible questions runs out first. For each of at most \( V \) positions in a sentence, at most \( 4V \) meaningful single-symbol or set-membership questions about this position can be asked along a path. A rough upper bound for the depth of a path is therefore \( 4L*V \) (one could establish a tighter upper bound). To obtain a path of this depth, one requires more than \( 4L*V \) sentences in the training data; thus, complexity results depend on whether \( D < 4L*V \) or \( D > 4L*V \).

If \( D < 4L*V \), the worst-case time complexity of growing a single-symbol SCT is \( O(D^2L^2V) \); for growing a set-membership SCT, it is \( O(D^3L^3V) \). Given the reasonable assumption that the number of iterations of the Gelfand-Delp algorithm [13] is a constant (in our experience, convergence has never required more than four iterations) these expressions can be divided by \( D \). Then, growing a single-symbol SCT is \( O(D^2L^2V) \) and growing a set-membership SCT is \( O(D^3L^3V) \). Both single-symbol and set-membership SCTs require \( O(D*L) \) time to classify a sentence of maximum length \( L \).

If \( D > 4L*V \), the time complexity of growing a single-symbol SCT is \( O(D^2L^2V) \) and growing a set-membership SCT is \( O(D^3L^3V) \). Under the assumption of a constant number of expansion-pruning iterations, these expressions are \( O(D^2L^2V^2) \) and \( O(D^3L^3V^3) \) respectively. For both single-symbol and set-membership SCTs, classifying a string of maximum length \( L \) is \( O(L^2V) \).

Detailed proofs can be found in [18].

**E. Computational Complexity of the SCT Algorithms**

**F. Possible Improvements**

With more computation, one could select the question at a node on the basis of its potential grandchildren or great-grandchildren. Alternatively, one could engage in several rounds of SCT-growing on the same data, adding bigrams and trigrams found in the leaves of the round \( i \) SCT to the "vocabulary" for round \( i + 1 \). The heuristic above for generating set-membership questions might be changed—for instance, a variant of the algorithm described in [6] might yield better set-membership SCTs.

**IV. CHANEL: AN SCT-BASED LINGUISTIC ANALYZER**

**A. Introduction**

The previous section describes general-purpose SCTs. To build the linguistic analyzer CHANEL, we employed single-symbol SCTs in various application-specific ways. For instance, "role assignment" SCTs (see below) are substring classification SCTs augmented with CHANEL-specific question types. "CHANEL" stands for "CRIM Hybrid Analyzer for Natural Language"—"hybrid" because this linguistic analyzer consists of a local, chart-based parser plus an SCT-based robust matcher. Since SCTs form only a part of CHANEL, CHANEL performance results (see Section V.B.) yield only tentative conclusions about SCTs. Two aspects of the ATIS task involve pure sentence classification and can be evaluated separately from other components of performance: generation of the set of displayed attributes (Section V.A.) and classification of sentences as ‘A’ or ‘D’ (Section V.B.).

**B. The ATIS Task**

The ATIS task is an ARPA-sponsored benchmark for speech recognition and understanding [8], [9], [10], [14], [25]. A corpus of spoken questions about air travel, their written form, and their "translations" into the SQL database language was obtained and split into training and test corpora. Sentences are classified as ‘D’ (semantically dependent on earlier sentences) or ‘A’ (semantically independent). The following discussion deals almost exclusively with the handling of class ‘A’ sentences. Class ‘D’ sentences were handled by a "context module" consisting of hand-coded rules, whose performance is tangential to the topic of SCTs. The three ATIS benchmarks are SPREC (speech recognition performance), NL (natural language understanding of written sentences), and SLS (spoken language understanding). We participated in two ATIS evaluations, in November 1992 and December 1993. In each evaluation, a single version of CHANEL handled both NL and SLS benchmarks.

**C. CHANEL**

CHANEL translates a word sequence into a semantic representation, as shown in Fig. 5 for "show me 10 a.m. flights from Boston to Denver and how much they cost." Before reaching the robust matcher, the sentence is preprocessed by a local chart parser. The robust matcher itself has two parts: the part that generates a list of displayed attributes (the database columns and functions of them the user wants to see) and the part...
that generates the constraints (which are made up of one or more frames combined by AND or OR). Each part is built of SCTs. In November 1992, the SCTs in the robust matcher were trained on 3248 class ‘A’ sentences from the ATIS 2 training data release; in December 1993, they were trained on 5501 class ‘A’ sentences from ATIS 2, February 1992, November 1992, and ATIS 3. All SCTs are single-symbol for reasons described below (Section V.A.). The average time taken by the chart parser and CHANEL together to process a sentence was about 5 seconds—this could be speeded up significantly (processing time is not an ATIS benchmark, so the current code is not optimized).

C1. The Local Chart Parser

The bottom-up chart parser looks for words or phrases carrying constraints that should be incorporated into the SQL query [21]. These words or phrases are replaced by three-letter symbols in the version of the input sentence sent to the SCT-based robust matcher. The meaning associated with each symbol is stored by the chart parser, since it will have to be recovered to produce the SQL query. For instance, the chart parser would convert the sentence “Delta flights from Boston to Denver between 9 a.m. and noon serving breakfast” into “AIL flights from CIT to CIT TIM serving MEA,” the version seen by the robust matcher. The assignments $\text{AIL} = \text{DL}$, $\text{CIT}_1 = \text{BBOS}$, $\text{CIT}_2 = \text{DDEN}$, $\text{TIM} = 9:00-12:00$, $\text{MEA} = \text{breakfast}$ will be stored until the semantic representation is generated. In the November 1992 CHANEL, the SCT-based robust matcher was entirely responsible for generating the displayed attribute list and for assigning roles to constraints. In the December 1993 version, the chart parser looks for and sometimes finds groups of constraints—for instance, arrival time and date may be grouped with the destination. The resulting role assignments may overrule those made by the SCT component; however, this phase of the chart parser has incomplete coverage.

C2. Choosing the Displayed Attributes

A given query may have any number of displayed attributes. CHANEL contains a forest of SCTs for choosing displayed attributes, each responsible for a particular attribute. The output of each displayed attribute SCT is either 1 (the attribute will be displayed) or 0 (the attribute will not be displayed). Each of the displayed attribute SCTs looks at the input word sequence independently—in principle, all SCTs could output 0 (giving an empty list) or 1 (giving a list containing all possible attributes). Fig. 6 shows part of the training data used to grow the fareId SCT. The November 1992 CHANEL contained 106 displayed attribute SCTs, while the December 1993 CHANEL contained 109. Labeling of the training data for these SCTs was carried out automatically by a program that examined the SQL “translation” of each ATIS sentence to determine whether a given attribute was displayed (label 1) or not (label 0).

C3. Assigning Roles to Constraints

In the constraints component of the robust matcher, role assignment SCTs assign roles to constraints. Properties of the November 1992 constraints component are:

- There were three role assignment SCTs—AIP (airport name), CIT (city name), and TIM (time);
- Possible values for AIP or CIT—origin, destination, stopover, site served by an airline, or ground transport location;
- Possible TIM values—arrival time or departure time;
- Role assignment SCTs were single-symbol substring classification SCTs (Section II.D.).

This version of CHANEL also contained hand-coded metarules that return “NO ANSWER” under the following conditions:

- Local constraint roles clash—e.g., two different CITs in same frame assigned origin role;
- Displayed attributes and constraints clash—e.g., displayed attributes involve the table ground_service, but a CIT is classified as destination of a flight.

Properties of the December 1993 constraints component are:

- There were three role assignment SCTs—one for AIP and CIT, one for TIM, and one for DAT (date) and DAY...
SHOW ME FLIGHTS FROM BOSTON TO DENVER ⇒ 0
ALL RIGHT WHAT I'D LIKE TO DO IS FIND THE CHEAPEST ONE WAY FARE FROM BOSTON TO DENVER ⇒ 1
I WOULD LIKE INFORMATION ON GROUND TRANSPORTATION IN THE CITY OF ATLANTA FROM AIRPORT TO
DOWNTOWN ⇒ 0
SHOW ME ALL THE FLIGHTS BETWEEN DALLAS FORT WORTH AND EITHER SAN FRANCISCO OR DENVER THAT
DEPART BETWEEN FIVE AND SEVEN P M ⇒ 0

Fig. 6. Part of training data for fare_fare_id SCT.

(day of the week);
• Possible values the same as before, with DAT and DAY classified as arrival, stopover, or departure day;
• Role assignment SCTs are single-symbol subsetting classification SCTs augmented with two new question types:
  1) Questions about whether a given attribute is on the displayed attribute list;
  2) For different codes handled by a single SCT, questions about what the current code is—e.g., one of the nodes in the AIP-CIT tree asks whether code is AIP.

The first new question type helps ensure consistency between displayed attributes and constraints (but requires that the displayed attribute list be generated before role assignment occurs). The second new question type allows constraints with similar semantics (AIP and CIT, DAT and DAY) to be merged safely. These new question types illustrate how SCTs can incorporate application-specific information with a modest amount of hand-tuning. Training sentences for these SCTs were hand-labeled, but could have been automatically labeled by a program that analyzed the SQL translations (as was done for displayed attribute SCTs).

V. EXPERIMENTAL RESULTS

A. Preliminary Experiments

One hundred and six SCTs for picking displayed attributes grown on a subset of ATIS 2 class A data were tested on a disjoint subset of the same data. On the test data, a “success” occurs only when the chosen set of displayed attributes is precisely identical to the set of displayed attributes found in the canonical ATIS MIN response.

SCTs for displayed attributes can vary along several orthogonal dimensions:
• Trained on a small vs. a large amount of data;
• Trained on data where the local parser has replaced certain substrings by symbols such as CIT or TIM vs. training on the original word sequences;
• “Well-shuffled” vs. “lumpy” training data;
• Single-symbol vs. set-membership SCTs.

Fig. 7 shows how classification accuracy for sentences depends on these factors, using SCTs trained and tested on NL data. The vertical axis indicates the percentage of sentences each of which has a completely correct set of attributes—thus, the best result is a forest of SCTs which generate exactly the canonical attribute list for 91% of the sentences. This does not mean that CHANFI would produce the canonical answer 91% of the time using these SCTs on these test data, since CHANFI has other components besides the attribute SCTs: the point of the experiment is to study the behaviour of SCTs in isolation. For all SCTs except those in the curve marked

Fig. 7. Classification accuracy for an SCT forest.

“lumpy,” the test data consisted of 542 ATIS ‘A’ sentences—parsed for the SCTs trained on parsed data, unparsed for the other SCTs.

Note that as the amount of training data grows, performance improves. The rate of improvement seems to depend linearly, rather than logarithmically as one might expect, on the number of training sentences. This augurs well for the SCT-based approach.
To obtain the curve labeled "lumpy unparsed single-symbol," the ATIS 2 data were split up into 25 sequential chunks. The training data files and the test data file were made up of random combinations of the chunks. All other curves shown in the figure were trained and tested on concatenations of files obtained by taking every nth sentence in the training data. Both forms of training involve random sampling, but because the ATIS 2 sentences are ordered by speaker, the "lumpy" method groups together in the same file most of the utterances of a given speaker. It may therefore over-train SCTs to deal with speakers in the training files.

Fig. 7 confirms that it is a good idea to carry out local parsing before SCTs classify the input sentences. It is less clear whether single-symbol or set-membership SCTs are better classifiers. In these experiments, the time to grow a displayed attribute SCT was roughly 7 minutes for single-symbol and 110 minutes for set-membership (on a SPARC-2 with 28 MIPS). Thus, we employed single-symbol SCTs trained on parsed, "smoothly" chosen sentences for both ATIS benchmarks. Research into faster, more accurate methods for growing set-membership SCTs is indicated.

### TABLE I

**November 1992 NL 'A' Benchmarks**

<table>
<thead>
<tr>
<th>System</th>
<th>NL 'A' Unweighted Error</th>
<th>NL 'A' Weighted Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATT</td>
<td>19.5</td>
<td>34.7</td>
</tr>
<tr>
<td>BBN</td>
<td>11.1</td>
<td>15.7</td>
</tr>
<tr>
<td>CHANEL</td>
<td>30.2</td>
<td>40.5</td>
</tr>
<tr>
<td>CMU</td>
<td>6.5</td>
<td>12.2</td>
</tr>
<tr>
<td>INRS</td>
<td>42.8</td>
<td>79.9</td>
</tr>
<tr>
<td>MIT-LCS</td>
<td>10.9</td>
<td>18.3</td>
</tr>
<tr>
<td>Neuron</td>
<td>16.6</td>
<td>31.1</td>
</tr>
<tr>
<td>Paramax</td>
<td>34.8</td>
<td>44.0</td>
</tr>
<tr>
<td>SRI1</td>
<td>15.1</td>
<td>22.2</td>
</tr>
<tr>
<td>SRI2</td>
<td>10.2</td>
<td>14.8</td>
</tr>
</tbody>
</table>

### TABLE II

**December 1993 NL 'A' Benchmarks**

<table>
<thead>
<tr>
<th>System</th>
<th>NL 'A' Unweighted Error</th>
<th>NL 'A' Weighted Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATT</td>
<td>7.4</td>
<td>14.7</td>
</tr>
<tr>
<td>BBN1</td>
<td>9.6</td>
<td>19.0</td>
</tr>
<tr>
<td>BBN2</td>
<td>16.1</td>
<td>31.9</td>
</tr>
<tr>
<td>CHANEL (o)</td>
<td>21.7</td>
<td>42.2</td>
</tr>
<tr>
<td>CHANEL (d)</td>
<td>12.3</td>
<td>23.9</td>
</tr>
<tr>
<td>CMU</td>
<td>6.0</td>
<td>12.0</td>
</tr>
<tr>
<td>MIT-LCS</td>
<td>10.0</td>
<td>20.1</td>
</tr>
<tr>
<td>SICS</td>
<td>14.7</td>
<td>28.3</td>
</tr>
<tr>
<td>SRI</td>
<td>14.3</td>
<td>20.8</td>
</tr>
<tr>
<td>Unisys</td>
<td>28.6</td>
<td>44.4</td>
</tr>
</tbody>
</table>

### TABLE III

**‘A’ vs. ‘D’ Results for ATIS 2-Trained SCT**

<table>
<thead>
<tr>
<th>Data</th>
<th>A =&gt; A</th>
<th>D =&gt; D</th>
<th>A =&gt; D</th>
<th>D =&gt; A</th>
<th>PCT Cor.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATIS 3</td>
<td>1308</td>
<td>814</td>
<td>104</td>
<td>147</td>
<td>89.4</td>
</tr>
<tr>
<td>Nov. 92</td>
<td>411</td>
<td>214</td>
<td>33</td>
<td>43</td>
<td>89.2</td>
</tr>
</tbody>
</table>

### B. November 1992 and December 1993 ATIS Results

Tables I and II show ATIS class ‘A’ NL unweighted and weighted error. For the November 1992 evaluation, the official criterion was weighted error (which penalizes a false answer twice as severely as "NO ANSWER"), while the official criterion for December 1993 was unweighted error (which penalizes a false answer and "NO ANSWER" equally). The learning systems described in Section II.A. were AT&T's Chronus (used for November 1992 but not for December 1993) and BBN's HUM (used for the first time in December 1993 and called BBN2). The November 1992 results are for a version of CHANEL that handles only class ‘A’ data. The December 1993 results are for both the CHANEL version used in the official test ("CHANEL (o)"), and a debugged version ("CHANEL (d)"). Immediately after the test, it became apparent that a handful of simple bugs in the non-SCT portions of CHANEL had greatly impaired its performance (the most serious was that the code for "Memphis" had been entered as "MMEN" instead of "MMEM"). In creating the debugged version, great care was taken to avoid studying the test data; the SCTs used in this version were identical to those in the official version.

Currently, the rules in the context module, which handles 'D' sentences by deciding what information to inherit from semantic representations generated earlier, are all hand-coded. We are exploring automatic learning for some of the rules in this module. In particular, it would be nice to have a module that decided with high accuracy whether a sentence is class 'A' or 'D' ('A' sentences inherit no information).

We have grown on 5816 ATIS 2 sentences (each labeled 'A' or 'D') an SCT that classifies a sentence as either 'A' or 'D,' based purely on its word sequence. This SCT is not allowed to ask any questions about the past—it does not even know when a sentence is the first in a scenario (such a sentence must be ‘A’). The test results for 2373 ATIS 3 sentences and 701 November 92 sentences are shown in Table III, (‘A => A’ gives the number of class ‘A’ sentences correctly classified as ‘A’, ‘A => D’ the number of class ‘A’ sentences misclassified as ‘D’, and so on). The percentage of correctly classified sentences is surprisingly high. A learning algorithm which was allowed to ask questions about the contents of semantic representations (current and previous) as well as about word sequences might yield performance competitive with that of hand-coded context modules.
VI. DISCUSSION

This work makes two original contributions:

1) Semantic Classification Trees (SCTs);
2) The SCT-based Robust Matcher.

The ATIS speech understanding task was used as a testbed for these ideas, but they should be widely applicable in natural language understanding systems. As discussed in Section II.A., other SU systems contain robust matchers: linguistic analyzers that grab semantically important phrases and stick them into appropriate slots, skipping over irrelevant words. Like AT&T's Chronus, the SCT-based Robust Matcher contains rules that are grown from data rather than hand-coded. Each SCT in the Robust Matcher looks at the whole sentence, permitting a given segment of the sentence to help build several different parts of the semantic representation (a capability that Chronus does not have).

The motivations for our work were as follows:

- We believe that in the long run, NLU systems whose parameters are learned from data will scale up better, and be more portable to new tasks, than purely hand-coded systems.
- The increasing availability of on-line natural language corpora, and the decreasing cost of computation relative to the cost of programmer time, will also tend to favour the automatic learning approach.
- Given that successful SU systems often incorporate robust matchers that skip irrelevant words, the word patterns learned should permit gaps.
- Decision trees have been successfully applied to other NL problems (Section II.B.), can be modified to learn patterns with gaps, and are tractable building blocks for a complete system.

The first two points are contentious. For instance, on the ATIS task the best hand-coded linguistic analyzers have outperformed those that carry out automatic learning. However, the ATIS results do not address the trade-off between human effort and performance—in the ATIS community as a whole, far more effort has been expended on rule-based than on learning-oriented systems. Conceivably, performance improvements in a hand-coded analyzer require a steeper increase in programmer-hours than such improvements in an analyzer that learns rules from training data; if so, the automatic learning approach is worth pursuing. Consider Fig. 7, where all curves (except the "lumpy" one) show steady improvement with more training data—there is no sign that learning is levelling off. In our view, the advantages of automatic learning should appear when the task is large, the number of programmer-hours is fixed, or the system must quickly be switched to a new domain. Perhaps the ARPA community should consider a "portability test" for NLU systems in which sample data for an entirely new domain are provided only shortly before the benchmarks, as a way of taking some of these factors into account.

In other areas of NL, approaches with a strong automatic learning component have, in the long run, proved more effective than approaches based on hand-coding rules. Speech recognition provides two examples: the triumph of HMMs over the expert system approach to acoustic modeling [26] and the triumph of statistical language modeling over approaches based on deterministic grammars [16]. A recent book that looks at parsing of English text [3] describes experiments in which state of the art, hand-coded deterministic parsers are applied to randomly selected sentences from newspapers and text corpora. Performance (using a very forgiving measure of structural correctness) varied between 16% and 60% correct. A probabilistic parser whose parameters were learned from training data obtained results varying between 92% and 96% on different but equally difficult test corpora [3]. A priori, there seems no reason why an approach that has been extremely effective at most other levels of language should prove ineffective when applied to semantics. The major caveat is the need for training data. One way of dealing with this would be to design the system so that positive and negative feedback are obtained from the end-user, yielding a system that improved over time.

People who find automatic learning of semantic rules intriguing may, nonetheless, have objections to SCTs in particular. One such objection we have encountered is that SCTs embody regular grammars, which are known to be inadequate as a model for the generation of natural language. This ignores the possibility that some problems in computational linguistics are best approached by treating natural language as regular to a first approximation: consider the non-trivial linguistic problems described in Section II.B., all handled by means of decision trees. Furthermore, the objection applies only to systems made up purely of SCTs—in general, SCTs will be used in combination with other components. For instance, in CHANEL the grammar built into the local parser may contain rules of arbitrary power.

SCTs do have genuine limitations. Since the patterns they learn contain gaps, they will tend to work well in contexts where a few key words and phrases contain most of the semantic information. The presence of deeply nested modifiers and complicated conjunctions will confuse them—they are unlikely to deal adequately with the prose of Henry James. Even on the ATIS data, we encountered some difficulty with modifying clauses. In this respect, the properties of SCTs may be complementary to those of the AT&T HMM-based approach. CHANEL performed well on sentences in which a single piece of semantic information is conveyed by different parts of a sentence but had trouble with sentences with complex structure; the AT&T system is good at segmenting sentences but bad at combining information found in different segments.

Class 'D' context-dependent sentences also created difficulties. CHANEL relied on a hand-coded context module to deal with class 'D' sentences; this should be replaced by a module with rules learned from training data. The SCT described in the previous section, which decides whether a sentence is 'A' or 'D,' is a step in this direction. We are working on an enhanced type of SCT for the context module, whose questions would ask not only about the current sentence but
about elements in semantic representations generated by previous sentences. The context module would consist of a forest of such enhanced SCTs, each making the binary decision whether a given element in a previous representation should be inherited by the current representation or not.

It would be desirable to provide an objective comparison between the performance of SCTs and that of other algorithms that learn semantic rules from data—e.g., AT&T's HMM-based approach and RBN's stochastic context-free grammar approach. In all three cases, however, the algorithms are embedded in systems with other components; overall performance may depend more on these other components and interactions between the components than on the learning component alone. Thus, just as the ATIS results are insufficient for deciding between purely hand-coded systems and those incorporating a learning component, so they are insufficient for deciding among learning algorithms. It should be possible to devise a benchmark for NLU that tests only for the ability to learn good semantic rules from training data.

We have presented a new method for learning semantic rules from training data. The rules learned by SCTs are based on regular expressions involving symbols from the lexicon (words or higher-level constituents) and gaps; the questions that can be asked in an SCT node depend on the answers to questions asked earlier. Using our linguistic analyzer, CHANEL, as an example, we have shown how SCTs can form building blocks for an NLU system. They can be used in a wide variety of ways in such a system: in parallel, in series, alone or combined with any number of other components. Future work will explore both new applications of SCTs (we are currently studying their ability to classify messages on the InterNet) and improvements to SCTs (for instance, algorithms yielding better-performing set-membership SCTs).

ACKNOWLEDGMENTS

The CHANEL system described here was built at the Centre de Recherche Informatique de Montreal (CRIM). The research was partially supported by the Institute for Robotics and Intelligent Systems, a network of the Centres of Excellence of the Government of Canada.

REFERENCES


Roland Kuhn (S'91-M'92-S'93-M'93) obtained his PhD in computer science from McGill University, Montréal, Canada, in 1993. He also has degrees in mathematics and biology (BSc and CXVII, Trinity College, University of Toronto) and theoretical biology (SM, University of Chicago).

Dr. Kuhn has been a research scientist at the Centre de Recherche Informatique de Montréal (CRIM) since November 1992. His research interests include speech recognition and understanding, machine learning, and genetic algorithms.

Renato De Mori received a doctorate in electronic engineering from Politecnico di Torino in Torino, Italy, in 1967. He became a full professor in Italy in 1975. Since 1986, he has been a professor and director of the School of Computer Science at McGill University, Montréal, Canada.

In 1991, Dr. De Mori became an associate of the Canadian Institute for Advanced Research and project leader of the Institute for Robotics and Intelligent Systems, a Canadian center of excellence. He is the author of many publications in the areas of computer systems, pattern recognition, artificial intelligence, and connectionist models. His research interests are in stochastic parsing techniques, automatic speech understanding, connectionist models, and program understanding.

Dr. De Mori is a fellow of the IEEE and a member of the IEEE Computer Society. He has been a member of various committees in Canada, Europe, and the United States and is on the board of the following international journals: IEEE Transactions on Pattern Analysis and Machine Intelligence, Signal Processing, Speech Communication, Pattern Recognition Letters, Computer Speech and Language and Computational Intelligence.