National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Keyword Classification Trees for Speech Understanding Systems

Roland Kuhn

School of Computer Science
McGill University, Montreal

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

copyright ©1993 Roland Kuhn

May 12, 1993

National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-91711-3

Canada

Acknowledgements

I have mixed feelings about this thesis: it forms the tombstone of my agreeable former life as a perpetual student. Nevertheless, I wish to thank the people who made its completion possible.

David Samuel and Louis Vroomen drew dozens of diagrams for me without complaining before I learned Framemaker. During a period when they had many other tasks to carry out, Régis Cardin and Diane Goupil worked very hard to provide data needed for this thesis. Diane also helped me with many minor problems that would have cost me months. Evelyne Millien created the local parser that forms an integral part of CHANEL; her hard work, sense of humour, and good nature made collaboration a pleasure. Caroline Drouin, Ying Cheng, Bassem Khalife, and Charles Snow created the SQL module. Caroline also deserves mention for the large amounts of pepper vodka and moderate amounts of tofu I ingested at her expense in the early years of my PhD, and for being a polyglot hippie vegetarian street musician in the 1990's. Under the leadership of Yves Normandin, the CRIM speech recognition team - Charles, Diane, Evelyne, Louis, Minh, Régis, and Roxane - provided a warm working environment and a stimulating forum for discussion of absolutely anything. Displaying true heroism, the team hired me after experiencing for a year my French, my jokes, and my voice. To all these colleagues and friends, my gratitude.

When I was looking for a supervisor, a vague interest in natural language processing led me to Renato De Mori. I could not have made a wiser choice. Starting at zero, Renato trained me to work on speech recognition and comprehension, currently among the most fascinating areas in any science; for this alone I would be permanently grateful. However, Renato gave me more than this: the example of a researcher who has fiercely resisted mediocrity throughout his career. Renato has always matched himself and his students against the best in the world, even when he commanded far fewer resources than his competitors. Consequently, he and those fortunate enough to work with him have produced world-class work. As a supervisor, he shows an exemplary concern for the well-being of his students, making strenuous efforts to ensure that they have financial support; in Canada, this is a difficult task. Finally, he is a man of great courtesy. Although he constantly devises new ideas of his own, and has an extraordinarily extensive knowledge of the approaches being taken by other researchers all over the world, he has

encouraged me to follow my own path. My thanks to Renato De Mori.

McGill University is a wonderful place to be a graduate student. Beginning with my old friend and room-mate Henry Schultz, far too many people form part of my happy memories of McGill and of Montréal to be listed here. However, I would like to thank the former systems staff of the McGill School of Computer Science, particularly Peter Deutsch, Alan Emtage, Luc Boulianne, Wanda Pierce, and Bill Heelan, for their help with my outlandish and often cretinous requests. I would also like to thank a place: Thomson House, the club and pub for McGill graduate students. Its civilized ambiance, British beers on tap, and wood panelling soothe the soul, its industrial-strength Zombies anaesthetized me for the Comprehensives, but what I appreciate most of all about TH is the way it contrasts with the dank, disagreeable, low-budget rat-hole reserved for undergraduates. As the English know, to relish most deeply one's status as a member of a privileged class, one must be able to contemplate the sufferings of an oppressed lower class.

The members of my family have had a lot to put up with over the years, and also deserve my thanks. My parents-in-law Jane and Lockwood Haight contributed to the thesis indirectly, partly by producing a daughter foolhardy enough to marry me and partly by inspiring me with the desire to rank higher than 4 on a scale of 10. To my brother Oliver: I will never forget the New Zealand apple cores. I enjoyed living with my sister Nicola the year I studied for the Comprehensives - her tolerance of my antics was admirable and lovable. My father and fellow McGill PhD Tillo did everything in his power to help me finish; for instance, he burnt special incense from Mount Athos on the morning of the Comprehensives, and again on the morning of my defense. My mother Naomi provided me with affectionate weekend hospitality, emotional support, and some useful genes for sangfroid and mathematics.

To my beloved wife Susan: what can I say? For most people, writing a PhD thesis is dull misery interrupted by episodes of sheer hell. Because you love me and because you were here with me, writing the thesis was satisfying and occasionally enjoyable. Because you are as unlike a computer geek as anyone can be, you were willing to talk about everything under the sun except the subject of my thesis, and that did me a lot of good. Because you are with me, I'm happy. "Thy sweet love remembered such wealth brings that then I scorn to change my state with kings". My thanks and my love go to you, Susan, most of all.

Abstract

Speech understanding systems try to extract meaning from one or several word sequence hypotheses generated by a speech recognizer. Designers of these systems rely increasingly on *robust matchers* to perform this task; a robust matcher processes semantically important word islands rather than attempting to parse the entire word sequence. This thesis describes a robust matcher for speech understanding whose rules are learnt automatically from training data. Learning is carried out by a new set of algorithms involving a new data structure, the **Keyword Classification Tree (KCT)**. By eliminating the need to handcode and debug a large number of rules, this approach facilitates rapid construction of a speech understanding system. Furthermore, the rules learned by a KCT, which depend on a very small number of words in each utterance, are highly resistant to errors by the speaker or by the speech recognizer. The thesis discusses a speech understanding system built at the Centre de Recherche Informatique de Montréal that incorporates this robust matcher, using the DARPA-sponsored Air Travel Information System (ATIS) task as training corpus and testbed.

Sommaire

Les systèmes de compréhension de la parole ont pour but d'extraire le contenu sémantique des hypothèses de phrases générées par un module de reconnaissance de la parole. Les concepteurs de tels systèmes recourent de plus en plus à des approches robustes pour accomplir cette tâche. Ces approches s'attachent aux groupes de mots sémantiquement importants plutôt que d'essayer d'analyser syntaxiquement toute la phrase. La présente thèse décrit une nouvelle approche pour apprendre des règles robustes automatiquement à partir de données d'entraînement. L'apprentissage se fait à l'aide d'un ensemble d'algorithmes basés sur une nouvelle structure de données : les arbres de classification de mots clés (KCT ; "keyword classification trees"). Cette approche facilite le développement rapide d'un système de compréhension de la parole en évitant l'écriture et la mise au point manuelle d'un grand nombre de règles. De plus, les règles apprises par un KCT dépendent d'un petit nombre de mots dans la phrase et sont donc assez robustes face aux erreurs du locuteur ou du module de reconnaissance. Cette thèse présente aussi le système de compréhension de la parole développé au Centre de recherche informatique de Montréal pour la tâche "Air Travel Information System" (ATIS) parrainée par DARPA, système utilisant le module développé à partir des KCT.

Contents

List of Figures

Chapter 1

Introduction

> On these Papers were written all the Words of their Language in their several Moods, Tenses, and Declensions, but without any Order. The Professor then desired me to observe, for he was going to set his Engine at work. The Pupils at his Command took each of them hold of an Iron Handle, whereof there were Forty fixed round the Edges of the Frame; and giving them a sudden Turn, the whole Disposition of the Words was entirely changed. He then commanded Six and Thirty of the Lads to read the several Lines softly as they appeared upon the Frame; and where they found three or four Words together that might make Part of a Sentence, they dictated to the four remaining Boys who were Scribes... Six Hours a-Day the young Students were employed in this Labour, and the Professor shewed me several Volumes in large Folio already collected, of broken Sentences, which he intended to piece together; and out of those rich Materials to give the World a compleat Body of all Arts and Sciences.

Jonathan Swift, *Gulliver's Travels* [Swi]

1.1 Problem Statement

When someone speaks to a speech recognition system, it tries to guess the sequence of words that best matches the acoustic signal. A typical system will

generate several *word sequence hypotheses*, each with an associated probability. If it is a dictation system, it will display the most probable hypothesis to the user for approval or correction. If it is a speech understanding system, the meaning of the utterance is more important than the precise sequence of words. Word sequence hypotheses in a speech understanding system undergo further processing to yield a *conceptual representation*, which may trigger actions by the non-speech part of the system. For instance, the information contained in the conceptual representation might cause a robot to move forward and pick up an object, or might initiate a search through a database for information requested by the user.

The part of a speech understanding system that translates word sequence hypotheses into a conceptual representation will be called the *linguistic analyzer*. In the recent past, the linguistic analyzer of a typical speech understanding system was built around strict syntactic rules [ErmWL,LowWL]; it was usually called the "parser". 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.

This approach has strong academic as well as practical appeal: one can write elegant papers about how a particular syntactic theory is incorporated in the parser. Unfortunately, many spoken sentences are meaningful but ungrammatical. A linguistic analyzer that relies heavily on syntax will refuse to respond to such sentences, or will generate and respond to an incorrect word sequence hypothesis that happens to be grammatical. Neither outcome is desirable.

A growing number of speech understanding 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. Some current speech understanding systems have a linguistic analyzer that will invoke the robust matcher only if a sentence-level parse fails, while others have a linguistic analyzer consisting entirely of the robust matcher. The robust matcher requires a large set of *semantic rules* to carry out its task;

these tell it how to identify the frame or frames referred to by the current utterance, and how to match slot-fillers to slots.

This thesis describes a robust matcher for speech understanding that incorporates a set of semantic rules automatically learned from training data. A data structure called the *Keyword Classification Tree* (KCT) has been devised for the purpose of learning semantic rules which depend on a small number of keywords in each utterance. These keywords, and the phrases they make up, are **not** specified in advance by the programmer but generated by the KCT-growing algorithm from the entire lexicon on the basis of the training data.

The robust matcher proposed in this thesis is original. Other robust matchers tend to ignore irrelevant words in an utterance, but these do not attempt to minimize the number of keywords that must be seen to generate the correct conceptual representation. The KCT-growing algorithms tend to find close to the smallest possible number of keywords required for semantic rules. Since the robust matcher built out of KCTs is unaffected by recognition errors in non-keywords, it is very tolerant of recognition errors. Researchers at AT&T have also proposed a robust matcher whose rules are learned from data rather than hand-coded [Pie92a, Pie92b, Pie91]. However, their matcher is based on statistical segmentation of the word sequence into concepts, rather than on classification trees; it has trouble dealing with concepts that overlap each other. The KCT-based robust matcher makes extensive use of independent KCTs, each of which looks at the entire word sequence, and therefore deals effectively with overlapping concepts.

1.2 Training and Testing

The training corpus and testbed for the KCT-based robust matcher was the DARPA-sponsored ATIS ("Air Travel Information System") task. ATIS was chosen for pragmatic reasons:

1. DARPA provides a large corpus of recorded ATIS utterances, each accompanied by its typed transcript and by the "translation" of the utterance into SQL judged most appropriate by DARPA. The KCT-based robust matcher requires large amounts of semantically labelled training data. The ATIS data fit this requirement perfectly, if one

takes the SQL translation as the criterion for successful semantic interpretation of an utterance. (However, the robust matcher generates a frame-based conceptual representation that is not in SQL code; an independent module called the "SQL module" generates the code from the conceptual representation, and could be replaced if we chose to use another database query language).

2. Many of North America's leading speech groups are working on the ATIS task. If the KCT-based robust matcher was applied to the ATIS task rather than some self-devised task, my work could more easily be compared to the work of other researchers.

3. The speech group at CRIM (Centre de Recherche Informatique de Montréal) had recently begun to participate in ATIS, and therefore had access to ATIS data. Members of the group kindly let me use the data, and subsequently collaborated with me in building a linguistic analyzer incorporating the KCT-based robust matcher for the November 1992 ATIS benchmarks (described in Chapter 8).

4. I was interested in seeing how semantic rules learned by the KCT-growing algorithms from written sentences differed from rules learned by the same algorithms from word sequence hypotheses output by a recognizer. Certain semantically important words may, for acoustic reasons, be poorly recognised by a particular system; KCTs trained on written sentences will choose some of these words as keywords, while KCTs trained on word sequence hypotheses should choose more reliably recognised words. The CRIM group was willing to provide me with speech recognizer output for the ATIS task.

Despite these arguments for the ATIS task, the work presented in this thesis would be worthless if it were *only* applicable to ATIS. Later in the thesis, I will argue that a robust matcher that learns rules for a particular speech understanding task from training data can be ported quickly to new tasks or new languages, unlike a hand-coded matcher.

1.3 The Probabilistic Approach to Natural Language Processing

The work reported here is part of a larger shift in natural language processing research, from approaches based on linguistic theory to approaches that treat natural language processing as a pattern recognition problem that can be handled probabilistically. Researchers who employ the probabilistic approach borrow some ideas from linguistics, but they avoid implementing linguistic theories in their entirety. They often consider linguistics, especially syntactic theory, as an obstacle to building practical systems that can handle a wide range of input.

The probabilistic approach has achieved its greatest successes and acquired the largest number of adherents in the speech recognition community. Many existing speech recognition systems work better than their predecessors because simple, robust models trained on large amounts of data were substituted for cumbersome systems of hand-coded linguistic rules, at several different levels of speech recognition. This shift in perspective was partly brought about by the ARPA Speech Understanding Project of the 1970's [KlaWL].

Despite the rhetoric employed on both sides of the debate, all probabilistic natural language systems are hybrids that incorporate a considerable amount of *a priori* linguistic expertise along with probabilistic parameters whose values are calculated from training data. A "pure" probabilistic approach that uses no linguistic knowledge at all is impossible. It is always necessary to define basic units and a structure for the probabilistic model, and this can only be done on the basis of linguistic knowledge. For instance, the IBM language models employ words or parts of speech as the basic units. Each choice reflects an *a priori* linguistic judgement. This is obviously true for parts of speech, and also true - though less obviously so - for words. Speakers of Indo-European languages tend to believe that language is segmented into words, and the typographical conventions of these languages reinforce the belief. If the designers of the IBM language models had been speakers of Hungarian or Inuit then a unit that seems more natural to speakers of these languages, such as the morpheme or the phrase, might have been chosen instead. The structure of the IBM models reflects another *a priori* judgement: that the identity of a word can be predicted by the immediately preceding

words.

Thus, the basic units and the structure of a probabilistic model always reflect the linguistic judgements or prejudices of its designers. What distinguishes the probabilistic approach from other approaches is that once the model has been defined, its parameters are calculated from training data. Furthermore, the structure of the model is usually very simple and linguistically "naive". The popularity of this approach seems to be spreading from the speech recognition community to researchers in other branches of natural language processing, such as machine translation and message understanding [Bro92, Bro88, Wei92].

Advocates of the approach, such as Geoffrey Sampson [Gars87 chap. 2] argue that the predominant rule-based approach leads to brittle toy systems that can deal only with a tiny set of made-up examples. Because these systems categorize sentences as either grammatical or ungrammatical, they cannot estimate degrees of acceptability; they reject a high proportion of sentences a human being would judge acceptable, and derive little useful information from such sentences. Adding more rules will not help. "I find it hard to imagine that in practice this revision process could ever be concluded. Like other rules concerning human behaviour, rules of grammar seem made to be broken... If the activity of revising a generative grammar in response to recalcitrant authentic examples were ever to terminate in a perfectly leak-free grammar, that grammar would surely be massively more complicated than any extant grammar, and would thus pose correspondingly massive problems with respect to incorporation into a system of automatic analysis" [Gars87 pg. 20].

By contrast, probabilistic systems deal with relative frequencies of outcomes and make no binary judgments about the grammaticality or ungrammaticality of a sentence. They can handle even extremely ill-formed input. In most cases, they are much simpler in structure than rule-based systems and require less programming time to set up. The main disadvantage of probabilistic models is the need to accumulate large collections of training data and carry out computation-intensive probability calculations. The rule-based approach makes heavy demands on human effort, in the form of linguistic expertise and programming time. If the need for hand-labelling training data can be avoided or minimized, the main demands of the probabilistic approach are on computer memory and processing power. As memory and computation get cheaper, the competitive advantage of the probabilistic approach

increases.

Advocates of the probabilistic approach are often caricatured as arrogant technocrats who believe that the secrets of natural language can be extracted by a purely mechanical process, like the Professor in the Academy of Lagado described in the quotation from *Gulliver's Travels*. I prefer to think that the probabilistic approach reflects an understanding of the fragility and trickiness of language. Language is complex, ever-changing, and difficult to master; we cannot force it to behave deterministically. Instead, we should model its uncertainties, giving systems the ability to learn probabilistic rules that work well in situations resembling those they were trained on.

The KCT-based robust matcher described in this thesis represents an attempt to extend the probabilistic approach to the one level of current speech understanding systems where hand-coded rules still reign supreme: the linguistic analyzer.

1.4 Thesis Outline

Note that some of the chapters listed below are mainly theoretical or describe related work by others; a reader interested in a quick overview of the original work and its practical results may wish to focus on Chapter 6 ("Building Keyword Classification Trees"), Chapter 8 ("CHANEL: A KCT-Based Linguistic Analyzer for ATIS"), Chapter 9 ("Results") and Chapter 10 ("Discussion").

- Chapter 2 - "Speech Recognition and Speech Understanding". Describes the structure of speech recognition systems and recent progress in speech recognition; discusses the role of the linguistic analyzer of a speech understanding system. A major theme of the chapter, which will be illustrated at several different levels of speech recognition, is the triumph of brute-force approaches involving simple models trained on large amounts of data over linguistically sophisticated, hand-coded approaches.

- Chapter 3 - "Speech Understanding Systems for the ATIS Task". Describes the DARPA-sponsored ATIS task and recent suggestions for changes in the definition of ATIS. The bulk of the chapter is a comparison of the linguistic analyzers of various ATIS systems.

- Chapter 4 - "Learning Patterns in Strings". Before designing algorithms that learn semantic rules, one must ask: what kinds of rules are learnable? A review of the literature on grammatical inference, syntactic analysis, and related topics, which places KCTs in context.

- Chapter 5 - "Classification Trees in Speech Processing". This chapter presents the techniques underlying the original work described in Chapter 6. These techniques for growing and using classification trees are illustrated by examples from other levels of speech processing (the work described in this thesis is the only application known to me of classification trees to speech semantics).

- Chapter 6 - "Building Keyword Classification Trees". Summarizes the decisions that must be made by a robust matcher in a speech understanding system, and shows how KCTs can learn rules for making these decisions. Most of the chapter is devoted to a description of the algorithms for growing two kinds of KCTs: the *single-symbol* KCT and the *set-membership* KCT.

- Chapter 7 - "Computational Complexity of the KCT Algorithms". Rigorous serial and parallel time complexity computations for the KCT growing and classification algorithms. The discussion of parallel implementation of these algorithms is of particular interest.

- Chapter 8 - "CHANEL: A KCT-Based Linguistic Analyzer for ATIS". CHANEL is a linguistic analyzer developed at CRIM and tested in the November 1992 ATIS benchmarks. This chapter describes the structure of CHANEL. Details of the conceptual representation language, of the local parsers that handle slot-filling phrases, and of the ATIS training data are given.

- Chapter 9 - "Results". For both the transcript task and the word sequence hypothesis task, comparisons are made between the results obtained with single-symbol KCTs and set-membership KCTs, and between KCTs permitted to ask questions about semantic categories and those that can only use lexical items. It is shown how performance varies with the size of the training corpus. The hypothesis that KCTs trained on recognizer output perform better in a speech understanding

system than transcript-trained KCTs is tested and discussed. Finally, this chapter analyzes the results obtained by CHANEL in the November 1992 ATIS benchmarks.

- Chapter 10 - "Discussion". Discusses the advantages and shortcomings of the KCT-based robust matcher, and makes suggestions for further work.
- Appendix - gives technical details of the KCT-growing algorithms.

Chapter 2

Speech Recognition and Speech Understanding

Speech recognition is a hard problem. There is a large amount of variability in human speech, as illustrated in figures 2.1 and 2.2 (courtesy of the CRIM Speech Recognition Group). Note from figure 2.2 that even the same speaker pronouncing the same word at different times demonstrates considerable variability. Human beings wield a vast amount of knowledge about acoustics, syntax, semantics, and about the pragmatics of the situation in which the speech signal is produced in order to identify spoken words. The nature of this knowledge and the manner in which it is applied are as yet only partially understood. Thus, systems which understand unrestricted natural language will not be built for many years.

Given the difficulty of the problem, it is remarkable that practical speech recognition systems are currently being built. This chapter describes potential applications for such systems, and surveys the twenty years of steady progress in speech recognition that have made them possible.

2.1 Potential Applications

It is likely that man-machine communication by voice will become part of daily life in the developed world within the next twenty years. Many millions of people in North America have already replied "yes" or "no" to a recorded message asking them whether they accepted a collect call; their reply was

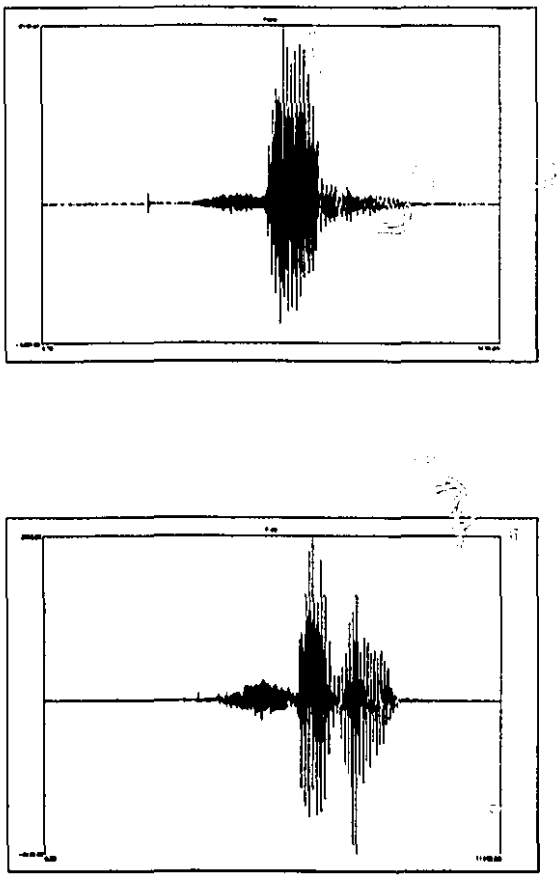
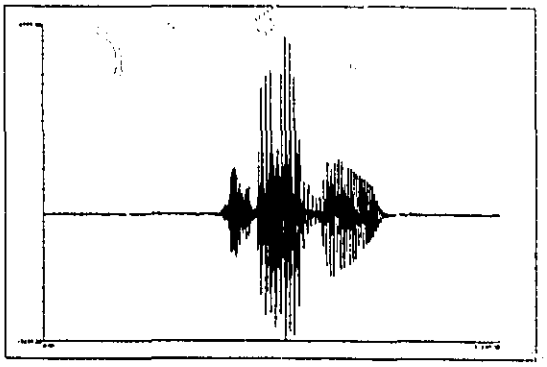


Figure 2.1: Between-Speakers Variation of Pronunciation of "seven"



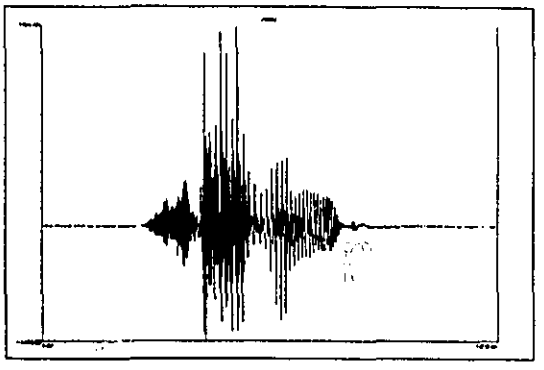


Figure 2.2: Within-Speaker Variation of Pronunciation of "seven"

processed by a speech recognition system developed by Bell Northern Research. It remains to be seen whether speech recognition technology will be confined to a few niche applications like this one, or whether voice will become one of the principal channels of man-machine communication.

Among existing systems, one may conveniently distinguish between *dictation systems* and *speech understanding systems*. The former, marketed by such companies as Dragon Systems of Boston, attempt to transcribe speech accurately. The latter execute spoken commands - for instance, they may attempt to retrieve from a database information asked for by the user. Thus, the work of a dictation system is completed when it has obtained the word sequence that matches the user's utterance, while a speech understanding system must generate a conceptual representation that initiates further action.

Although a dictation system only carries out speech recognition, while a speech understanding system carries out speech recognition and then performs an extra step, dictation systems are no easier to design than speech understanding systems. For instance, dictation systems normally recognize a much larger vocabulary than speech understanding systems. Some data entry systems carry out tasks that lie along the border between dictation and speech understanding. Kurzweil Applied Intelligence of Cambridge currently markets a system designed to take dictation from a doctor examining a patient. The system fills in fields in a chart, and will prompt the doctor at the end of the examination if there are any unfilled fields.

Medium-term applications of speech understanding systems will be limited mainly by the state of the art in knowledge representation and semantics. Speech understanding systems will probably soon handle many routine, high-volume transactions that are carried out in the same way most of the time [WaiWL pg. 1]. Examples are enquiries about schedules and ticket-buying over the telephone. An application like airline flight booking (as in the ATIS scenario) yields a high proportion of simple requests and some more complicated ones requiring human judgment; here, one could envisage the system transferring the complicated requests to a human being. For sensitive applications like bank balance transfers, speech understanding could be combined with **speaker** recognition to enhance security.

Speech understanding systems may also provide a communication channel in command and control situations where the individual's hands and eyes are otherwise occupied, as for surgeons and fighter pilots. Similarly, the

handicapped may benefit from wheelchairs or robots that respond to voice commands, and owners of intelligent houses and intelligent cars may wish to communicate with them verbally. Another obvious application of speech understanding is communication with a personal computer. Apple Computer Inc. is currently working on a voice interface called "Casper" to the Macintosh personal computer, under the guidance of Kai-Fu Lee, a highly respected speech recognition researcher.

In the long term, some researchers envisage the *translating telephone* or even the ultimate *conversational computer* [WaiWL]. The conversational computer would have the ability to understand, think about, and respond to ordinary conversation. I am more sceptical about this possibility than I am about the ones mentioned in previous paragraphs, since I believe it would require a revolution in our understanding of semantics and knowledge representation. In the past, good natural language processing systems were built for semantically limited domains - microworlds - but deep problems were encountered when one attempted to build more general systems. Nevertheless, it is clear that speech understanding systems have a host of potential applications, and an interesting future.

2.2 Dimensions of Difficulty

Let us now return to solid ground and survey the difficulties that degrade speech recognition performance. Existing speech recognition systems can be located in a multidimensional space defined by axes of difficulty. Designers of these systems often deal with unavoidable difficulty in one dimension by accepting a more forgiving definition of the task in another dimension.

The main dimensions of difficulty are [WaiWL]:

- Isolated-word or continuous speech;
- Vocabulary size;
- Task and language constraints;
- Speaker dependent or independent;
- Acoustic confusability of vocabulary items;

- Environmental noise.

Systems that recognize only isolated-word speech require the user to pause for at least 100-250 msec after each word, while continuous speech systems impose no constraint on the user, allowing him to speak rapidly and fluently. Continuous speech may be cut up into words in many different ways: consider "euthanasia" and "youth in Asia", "new display" and "nudist play". The difficulty of recognizing word boundaries makes continuous speech recognition much more difficult than isolated-word speech recognition.

As the size of the vocabulary increases, there are more mutually confusable words, and exhaustive search of the whole vocabulary becomes computationally intractable. With a small vocabulary, one can build a good acoustic model for recognizing each individual word. With a large vocabulary, it becomes difficult to collect enough training data for each word, so that subword models based on phonemes or syllables are employed instead. When such subword models are concatenated to form word models, some word-specific information is lost, reducing recognition accuracy.

The stronger the constraints known to affect the order and choice of words in an utterance, the easier speech recognition becomes. Such constraints are incorporated in a *language model* that helps to reduce the number of reasonable word candidates at a given time. For some tasks, users may be forced to speak according to the rules of an artificial syntax to facilitate recognition.

A speaker dependent system is trained to deal with the utterances of a particular individual. Typically, each new person who will be working with the system takes an hour or so to train it by reading it all the words in the vocabulary (if it is a small-vocabulary system) or a passage containing the most common combinations of phonemes in the language (if it is a large-vocabulary system). A speaker independent system is trained once, before use, and must then be able to handle a wide variety of voices not encountered during training. Provided a speaker dependent system is tested only on the voice it has been trained for, it will perform better than a comparable speaker independent system. To give speaker independent systems accuracy closer to that of speaker dependent systems, *speaker adaptation* methods have been developed; these adjust the parameters of the system's recognition models in the course of an interaction to better model the current speaker, or map the current speaker onto one of a number of speaker clusters and then employ

the model corresponding to that cluster.

Two vocabularies of the same size may differ in acoustic confusability. Thus, the ten digits are easier to recognize than the letters rhyming with 'B'.

Finally, environmental noise often affects performance; a speech understanding system that operates in a factory may be much harder to design than one that operates in a quiet office. Background conversation, slamming doors, sneezes, emotionally stressed users, and a host of other phenomena must be taken into account. Environmental factors may be quite subtle - some speech recognition systems work better with certain microphone types than with others.

These dimensions of difficulty can be traded off against each other. Thus, a dictation system like those marketed by Dragon has an extremely large vocabulary and flexible word order which are "paid for" by requiring each user to train the system to his voice, to pause between words during dictation, and to shield the system from ambient noise. A speech understanding system designed to execute commands in the cockpit of a fighter plane would have extremely high noise tolerance, attained by a small vocabulary and constrained syntax for commands, and possibly also by making the system speaker dependent. A system for making air reservations over the phone must carry out speaker independent continuous speech recognition, and tolerate some background noise; hence, the vocabulary size must be relatively small. While the designer cannot impose a constrained syntax on users of this air reservation system, he might cause the system to ask carefully designed questions that made user utterances more predictable.

In this thesis, the focus will be on speaker independent continuous speech understanding. The ATIS testbed for the KCT algorithms has a modest vocabulary (about 1000 words) and assumes low levels of ambient noise, with the speech transmitted directly to the system microphone rather than over the telephone. There are semantic but not syntactic constraints on user utterances, which must deal with air travel and related subjects. ATIS will be described in more detail in Chapter 3.

2.3 Non-Probabilistic Speech Recognition Systems

This section describes *template-based* and *knowledge-based* speech recognition systems. Although these two system types are still in use for certain specialized applications, the probabilistic type described in the next section has supplanted them for large vocabulary, speaker independent, continuous speech recognition. Many aspects of probabilistic systems are derived from these older types.

2.3.1 Template-Based Speech Recognition

A summary of template-based approaches may be found in [O'S87 pp. 415-459]; further readings on this topic may be found in Chapter 4 of [WL90]. The first two system components shown in figure 2.3 perform feature extraction and are often called the *front end.* The front end eliminates signal variability due to the environment and to special characteristics of the speaker's voice, then converts the signal to acoustic features such as formants, phonemes, or phoneme sets (e.g. "fricative" or "plosive"). Thus, the front end eliminates redundancy and reduces the amount of data to manageable size.

From the sequence of features, the system forms the current pattern. This is then compared with stored templates, and the template that matches the current pattern most closely is chosen. This requires a local *distance measure* for comparing a feature in the pattern with a feature in a template, a global measure for the overall pattern-template distance together with a computationally efficient method for computing it, and a *decision rule* for choosing the final word sequence. In the "active model" of template-based speech recognition, there may be feedback from the component that hypothesizes a pattern to lower-level components.

Consider an isolated word, small vocabulary system. Here, word boundaries will be easy to spot and it makes sense to design the system so that the current pattern and the stored templates are individual words. A global distance obtained by lining up the start of the current pattern with the start of a template and adding up local distances will not tell us much. Instead, we can temporally "stretch" some phonemes and "compress" others in the current word until as many portions of it as possible are lined up with like

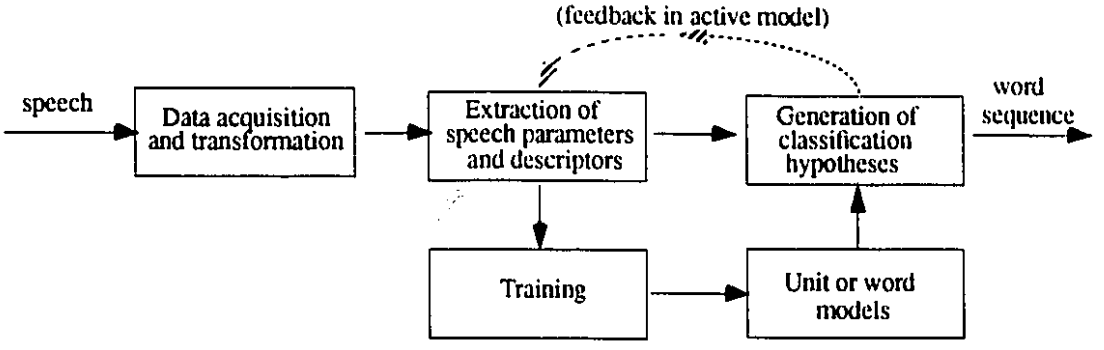


Figure 2.3: Template-Based Speech Recognition

portions of the stored word; a penalty for stretching and compression must be built into the global distance score for each such match. This idea is the basis for *dynamic time warping*, the most popular methodology for pattern matching in template-based systems. A variety of distance measures and decision rules have been devised for the dynamic time warping algorithm.

Template-based systems work well for isolated word speaker dependent recognition of small vocabularies containing short words. As vocabulary size and word length increase, computation time goes up. Continuous speech requires the dynamic time warping algorithm to consider all possible combinations of word starts and stops, and thus also increases computation time. Template-based approaches are even worse at segmenting continuous speech.

Finally, different speakers may use different phonemes in pronouncing the same word, creating a difficulty that dynamic time warping cannot handle with a single word template. As we will see, probabilistic systems can incorporate alternative pronunciations in a single model. Template-based systems are incapable of carrying out this kind of generalization and can only cope with this problem by storing several different pronunciations of the same word, which increases computation time and ignores similarities between the different pronunciations - thus failing to take full advantage of the training data.

2.3.2 Knowledge-Based Speech Recognition

Many researchers from the mid-1970s onward believed it was important to incorporate linguistic rules in speech recognition systems, which is difficult to do with a template-based approach. Although the phrase "knowledge-based speech recognition" is widely accepted as the designation for the work of these researchers, [WaiWL pg. 4], [O'S87 pg. 418], it is misleading. Every speech recognition system, from the earliest template-based system to a recent probabilistic one, is the product of hard-won human knowledge. It would be more accurate to say that this group of systems is characterized by the "expert system" approach.

The best example of this approach was HEARSAY, a system developed at CMU as part of an ARPA-sponsored research effort to achieve speaker independent continuous speech recognition between 1971 and 1976. HEARSAY pioneered the idea of a "blackboard" architecture which allowed multiple knowledge sources to talk to each other. Each knowledge source is an expert

system covering a particular aspect of linguistics, such as acoustic-phonetics, syllabification, prosodics, syntax, or semantics; each functions in parallel with the other knowledge sources. The blackboard contains hypotheses written on it by the knowledge sources; a hypothesis written there by one knowledge source often causes other knowledge sources to add new hypotheses. A description of the system can be found in [ErmWL].

The architecture of HEARSAY permitted it, unlike a template-based system, to benefit from up-to-date linguistic expertise in each area corresponding to a knowledge source. Unfortunately, knowledge sources often contradicted each other, or got stuck waiting for information from each other. Subsequently, a CMU group devised a streamlined, "compiled" version of HEARSAY called HARPY [LowWL]. HARPY discarded many of the knowledge sources employed by HEARSAY and used only phonemic knowledge (specifying one or more acoustic templates for each phoneme), juncture rules (for dealing with phones at word boundaries), lexical knowledge (representing alternative word pronunciations), and syntactic knowledge (specifying permissible word sequences). All these knowledge sources were expressed as graphs, and all except for the phonemic knowledge were hand-coded.

The final, dramatic step in creating HARPY was to compile all these knowledge sources into a single, 15000-state graph. During recognition, a set of paths close to the best found so far was explored in parallel with the best path. This heuristic *beam search* made backtracking unnecessary, thus speeding up search - it was one of HARPY's most important contributions.

HARPY attained the highest level of performance among the systems participating in the five-year ARPA speech understanding project that ended in 1976. Much of its success was due to tight and rather unnatural syntactic constraints which greatly decreased the number of word candidates that had to be considered at a given time. On the other hand, compared with HEARSAY, HARPY demonstrated the advantages of a uniform encoding of different types of knowledge that avoided run-time conflicts between knowledge sources.

Although the "expert system" approach to speech recognition has been superseded for most applications by the probabilistic approach, there are strong arguments to be made for incorporating linguistic constraints in speech recognition systems.

In a 1985 article, Victor Zue listed several linguistic constraints that could improve speech recognition performance [ZueWL]. For instance, the acoustic

front end should take into account what is known about the human auditory system. The ear's temporal window for frequency analysis is non-uniform. Low-frequency sounds such as sonorants are assigned a long integration window, yielding good frequency resolution, while high-frequency sounds such as stop bursts are assigned a short window, yielding good temporal resolution. Other "design decisions" in the human ear lead to superior formant tracking, and thus superior phoneme recognition; they also lead to increased robustness in the presence of environmental noise. Stephanie Seneff initiated and successfully implemented many of these ideas for improving the front end [SenWL].

Zue also advocated the study of phonemes in context. "Speech is generated through the closely coordinated and continuous movements of a set of articulators with different degrees of sluggishness... the acoustic properties of a given phoneme can change as a function of the immediate phonetic environment" [ZueWL, pg. 201]. This phenomenon is called *coarticulation.* Zue suggested that prosody is a valuable clue - unstressed syllables can be represented by broad phonetic categories, with analysis focusing on the stressed syllables that help most in identifying a particular word. Since unstressed syllables are acoustically variable, it is more accurate to model them coarsely. A related point is that the distance measure between the current pattern and a stored template should concentrate on regions which are perceptually salient.

Finally, Zue emphasized the importance of a coherent knowledge representation and control strategy for combining knowledge from different sources. HEARSAY ran into difficulty because of loose coupling between knowledge sources, which led to conflicts and communication problems.

It is on this level of representation and control strategy that the knowledge-based approach and the probabilistic approach differ. At other levels, there is no conflict between the two; many of the good ideas generated by Zue and by other advocates of the knowledge-based approach have been reborn in probabilistic costume. Designers of probabilistic systems employ linguistic expertise in defining the *structure* of probabilistic models. However, they prefer the *parameter values* for these models to be estimated automatically from large amounts of training data, instead of by human experts.

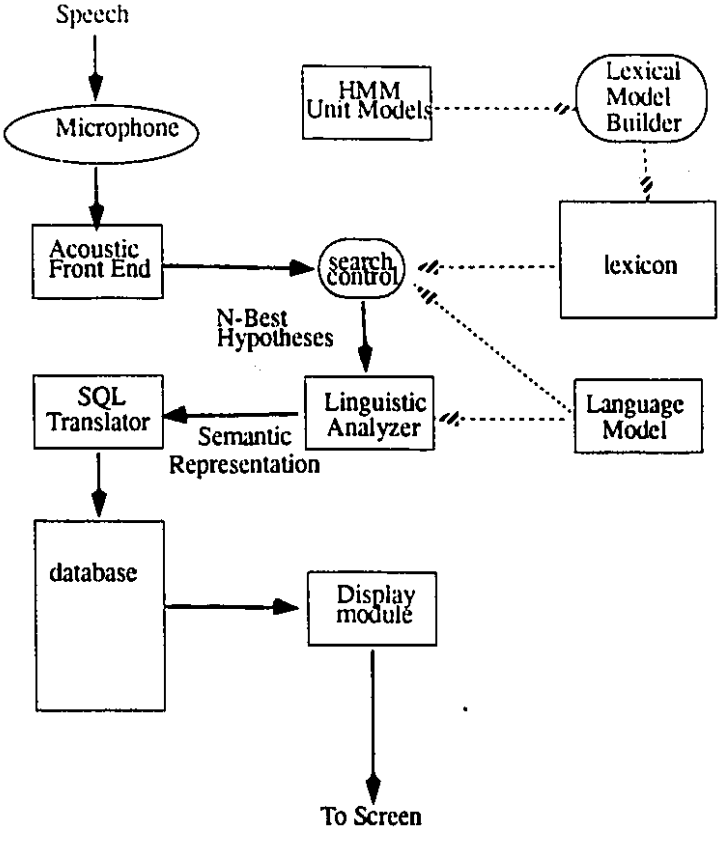


Figure 2.4: Structure of the CRIM Speech Recognition System

2.4 Probabilistic Speech Recognition Systems

Figure 2.4 shows the structure of the CRIM speech understanding system, which is described in [Norm92]. Probabilistic systems for speaker independent, continuous speech recognition all strongly resemble each other, so the diagram would look much the same if another system were chosen as the example.

These systems view speech recognition as a decoding problem. Let $\mathbf{y}$ represent an acoustic observation vector, and $\mathbf{w}$ a sequence of words. The task of a speech recognition system is to find $\mathbf{w}$ such that $P(\mathbf{w}|\mathbf{y})$ is maximal.

By Bayes's rule, we have

$$P(\mathbf{w}|\mathbf{y}) = P(\mathbf{w})P(\mathbf{y}|\mathbf{w})/P(\mathbf{y}).$$

$P(\mathbf{y})$ can be ignored, since it is constant at a given time. Thus, a probabilistic speech recognition system seeks to find $\mathbf{w}$ maximizing $P(\mathbf{w})P(\mathbf{y}|\mathbf{w})$. The calculation of $P(\mathbf{w})$ is the job of the *language model*, while $P(\mathbf{y}|\mathbf{w})$ is calculated by *hidden Markov models* (HMMs) operating on the output of the acoustic front end.

2.4.1 The Acoustic Front End

The front end digitizes the acoustic signal and cuts it into frames, usually at a fixed frame rate. It then extracts a small number of parameters per frame, which reflect aspects of the signal's power spectrum. The frame seen at time i thus generates a vector y_i of spectral parameters. Many systems vector quantize the frame vectors y_i by mapping each onto the nearest entry in a vector quantization *codebook*; some systems use several codebooks. To illustrate the use of classification trees, Chapter 5 of this thesis will describe how they can assist vector quantization. The observation vector $\mathbf{y}$ which is the input to the hidden Markov models is the concatenation of the y_is, or of the codewords to which vector quantization has mapped them.

The front end of a typical probabilistic system incorporates many ideas developed for template-based and knowledge-based systems. The choice of parameters usually reflects some form of auditory modeling, thus building on the work of Seneff and Zue. However, most systems do not attempt to replicate all the stages of processing carried out by the human ear.

An important recent development in the front end is the use of dynamic parameters describing how other parameters are changing - in other words, the use of first and second derivatives. This is a way of letting the front end model longer-term trends that are poorly modeled by hidden Markov models; it can be seen as highlighting one of the major flaws of HMMs [Norm91 pg. 7]. HMMs assume that each frame of the acoustic signal, covering about a centisecond, is statistically independent of the previous one - a completely unrealistic assumption. Thus, the front end can be designed in a way that helps compensate for some of the flaws in the next processing stage.

2.4.2 Hidden Markov Models

HMMs lie at the core of a probabilistic speech recognition system. Once the system designer has chosen the speech unit - possibly the word, possibly a subword unit such as the phoneme - every example of that unit is modeled by means of a finite-state graph and a different output distribution for each state in the graph. Speaker variability is modeled in two ways: by means of the output distribution associated with each state, and by means of probabilities for transitions between states. Each frame of the acoustic signal corresponds to an output from some state. The output distributions for states enable the model to deal with such phenomena as different pronunciations for part of a word, while the transition probabilities enable the model to deal with variations in timing such as skipped, lengthened, or truncated syllables of a word (if the unit is the word).

Figure 2.5 shows an HMM for the word "sauce". This model was formed by concatenating models for "s" and "ao". Given such a model and an observation vector $\mathbf{y}$, it is possible to calculate the probability $P(\text{"}sauce\text{"}|\mathbf{y})$ that an attempt to pronounce the word "sauce" gave rise to $\mathbf{y}$.

The popularity of HMMs is partly due to their surprisingly high level of recognition accuracy, and partly to the tractability of the algorithms associated with HMMs. The accuracy of HMMs is surprising because, as mentioned earlier, each frame of data is assumed to be independent of the previous frame (given the state). The reader interested in the details of the algorithms for training and using HMMs should consult the lucid description given in [RabWL].

HMMs can be classified as *discrete*, *continuous*, or *semi-continuous*; these terms refer to the modeling of the output distributions. Discrete HMMs are easier to implement than continuous HMMs, while the latter offer more degrees of freedom. Semi-continuous HMMs use mixtures of continuous distributions: each state's output is modeled by selecting discrete values specifying the weights to be assigned to each of the continuous densities (which are available to all states). Semi-continuous HMMs seem to combine the advantages of the other two types, and are becoming increasingly popular. [Norm91 pp. 38-43] defines and discusses all three HMM types.

There are some problems with HMMs. Nothing in their structure suggests a speaker is unlikely to alternate from falsetto to bass, or from the accent of Bela Lugosi to that of Nelson Mandela, at each frame boundary. HMMs also

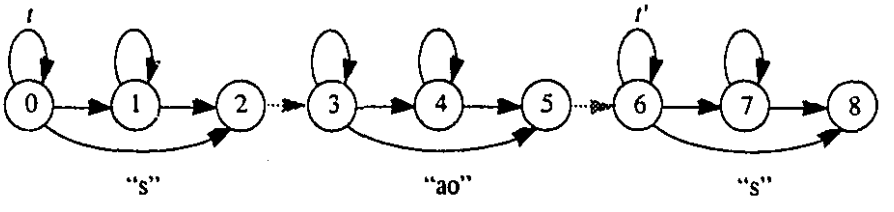


Figure 2.5: HMM for "sauce" [Norm91, pg.22]

yield erroneous predictions for the overall timing of a unit - for instance, an HMM for a word will predict that very fast or very slow pronunciations of the word are much more unlikely than they actually are (the estimated average time will be about right). As a result, there is a tendency to smuggle long-term constraints into other parts of an HMM-based system, such as the front end and the choice of units. This is only a partial solution, and attempts to turn HMMs into better models without losing their advantages are an active area of research.

2.4.3 Choosing the Units

For an application with a small vocabulary, it makes sense to build an HMM for each word in the vocabulary. As vocabulary size grows, it becomes more and more arduous to obtain enough training data for each word. The designer of a large vocabulary system would naturally consider making the phoneme the basic unit of the system, with word models built by concatenating phoneme HMMs.

Unfortunately, the acoustic realization of a phoneme is extremely variable: it depends on the accent, the speaking rate, the phonetic context, stress, and a number of other factors. Many system designers have attained good results by defining units that are more specific than phonemes and more general than words: *phonemes in context*. For instance, triphone modeling involves building an HMM for each phoneme given the previous and following phoneme. Since there are roughly 40 phonemes in English, this could conceivably involve building $40^3 = 64000$ different HMMs. Fortunately, the number of triphones that actually occur is often much smaller. An example is the Resource Management application which involves somewhere between 2000 and 8000 triphones, depending on whether one considers only intra-word or also inter-word contexts [Norm91 pg. 10].

Many variants of this basic idea have been tried. For instance, on the grounds that function words like articles, prepositions, and conjunctions cause a disproportionate number of recognition errors, Kai-Fu Lee introduced models specific to function words. Part of Chapter 5 describes an application of classification trees to derivation of models for phonemes in context. All these context-dependent subword models are an expression within the probabilistic approach of a theme mentioned by Zue in the article cited above [ZueWL].

2.4.4 The Language Model

Recall that the language model specifies the *a priori* probability $P(\mathbf{w})$ of the word sequence $\mathbf{w}$. The knowledge-based systems described earlier tended to define possible word sequences by means of a finite-state grammar, effectively assigning a probability of 0 to almost all possible word sequences.

It was the IBM speech recognition team headed by Dr. Frederick Jelinek that initiated the study of more flexible language models, perhaps because the team was working on a dictation system rather than a speech understanding system. Speech understanding systems tend to make the people building them believe, usually wrongly, that they know in advance how people will phrase their requests to the system. Builders of a dictation system know that the users will employ a wide variety of linguistic constructions that are impossible to predict in advance.

The IBM *trigram model* for dictation predicts the probability of occurrence of a word w_i on the basis of the identity of the two words w_{i-2}, w_{i-1} that preceded it. To do this, one analyses a huge training corpus of documents, which should be similar in topic and vocabulary to the documents which will be dictated to the completed system. The analysis involves counting all three-word sequences. The trigram model estimates the probability $P(w_i|w_{i-2}, w_{i-1})$ that the current word is w_i, given the two preceding words, by the frequency $f(w_i|w_{i-2}, w_{i-1})$ with which w_i immediately followed w_{i-2}, w_{i-1} in the training corpus. Since three-word sequences **not** encountered in the training corpus will inevitably occur and should not be assigned a probability of 0, in practice the trigram frequencies are interpolated with bigram frequencies and unigram frequencies (the frequency of individual words in the corpus).

This kind of language model, often criticized as childishly simplistic by academic linguists, has proven astonishingly successful and durable for practical speech recognition. Many variants exist. For instance, the *triPOS* model "forgets" the exact identity of the preceding two words and remembers only their parts of speech (POS), i.e. whether they were nouns, verbs, adjectives, or whatever. The estimated probability of the current word is the product of its probability within a given POS with the probability of that POS given the two preceding ones. If g_i represents the POS of the ith word, we have :

$$P(w_i|g_{i-2}, g_{i-1}) = P(w_i|g_i)P(g_i|g_{i-2}, g_{i-1}).$$

The trigram model performs best if more than two or three million words of training text are available. If not, other models like the triPOS model or a bigram model estimating $P(w_i|w_{i-1})$ are preferable. It has been shown that a triPOS model can be significantly improved by assigning higher probabilities to words that occur in the course of the recognition task [Kuh90,Kuh92].

The effectiveness of a language model is expressed by the reduction in *perplexity* it permits. When a language model has perplexity S for a given sequence of words $\mathbf{w}$, recognizing $\mathbf{w}$ is approximately as difficult as if, at any time, there were S equiprobable word candidates. The perplexity is defined as $S(\mathbf{w}) = P(\mathbf{w})^{-1/n}$, where n is the number of words in $\mathbf{w}$.

These simple language models, combined with HMMs for acoustic modeling, yield such rapid, robust and accurate lexical search algorithms that they are now employed in many speech understanding systems, as well as in dictation systems. This has occurred despite the original belief of most speech understanding researchers that speech understanding scenarios would yield tighter constraints based on syntactic and semantic models. Chapter 3 describes some speech understanding systems for ATIS in which syntactic and semantic constraints are applied only at the last stage of the search; they come into play after the knowledge contained in a simple language model and the HMMs has been applied to generate a small number of word sequence hypotheses.

2.4.5 Lexical Search and the N-Best Hypotheses

Recall that the task of the speech recognition system is to find $\mathbf{w}$ maximizing $P(\mathbf{w})P(\mathbf{y}|\mathbf{w})$, where $P(\mathbf{w})$ is calculated by the language model, $P(\mathbf{y}|\mathbf{w})$ by concatenated HMMs. How can the search for the optimal $\mathbf{w}$ be carried out efficiently?

Most systems use *Viterbi search* to find the best path through the concatenation of speech unit models. Viterbi search stores a trellis of paths and their scores, where the score is some function (usually logarithmic) of the probability of the path so far. Viterbi search does not yield the total probability of a given word sequence: it discards low-probability paths through the graph of concatenated HMMs. In principle, the probability of a word sequence should be obtained by summing the probabilities of all possible paths through that word sequence, including the most unlikely. Thus, Viterbi search may lose the path with the highest score. Furthermore, the

imperfect nature of the acoustic models implies that even the word sequence with the highest score may not be identical to the word sequence actually uttered by the user. Thus, it is prudent to find other word sequences that are "close" in some sense to the top word sequence found by Viterbi.

In 1990, F. Soong and E.-F. Huang devised an ingenious two-pass algorithm that employs the scores stored in the Viterbi trellis as the basis for a low-cost A^* backward search that generates the N best path hypotheses [Soo90]. A^* search is a strategy that always extends the path with the lowest value of $f(n)$, where n is a node and $f(n) = g(n) + h(n)$. Here $g(n)$ is the **known** cost of the path from the start to n, and $h(n)$ is a heuristic **estimate** of the cost of extending a path from n to the finish. $h(n)$ is called **admissible** if it is guaranteed never to overestimate the cost from n to the finish - in other words, if it is a lower bound. If $h(n)$ is admissible, the strategy of expanding the node with the lowest value of $f(n)$ can never cause us to overlook the lowest-cost path. Even if the wrong path is extended for a while, sooner or later its value of $g(n)$ is guaranteed to exceed the value at an open node on the best path. Note that if $h(n)$ is set to 0, it is admissible but leads to an inefficient breadth-first search; the more closely $h(n)$ approaches from below the true cost of the remaining path to the finish, the more efficient the A^* search.

Soong and Huang perceived that if the Viterbi scores generated in the forward search at each time i are remembered, they yield exact scores $h(n)$ for an A^* search beginning at the end of the utterance and moving backwards in time. In this manner, the N best word sequences can be generated for any reasonable value of N at only a tiny additional computational cost over the amount of processing required to find the best word sequence.

Real-time systems often employ *beam search* or *fastmatch* techniques to speed up Viterbi lexical search [Norm92 pp. 43-44]. In a traditional beam search (as in the HARPY system), paths whose probability falls below a certain threshold are pruned; the threshold is usually the probability P of the best path, minus some δ. The CRIM group is exploring the possibility of a more intelligent beam search that would allow δ to increase in regions of uncertainty, and decrease in regions of greater certainty. Fastmatch is a look-ahead technique which identifies the most promising extensions of a path. Both techniques make search inadmissible, since they make it possible for the best path to be pruned. Provided this happens seldom, it is a price worth paying for real-time performance.

2.4.6 The Linguistic Analyzer

There is a surprising degree of consensus among researchers about the best overall structure and components for a large vocabulary, continuous speech recognition system. Details differ, but the description of components and algorithms given in the preceding subsections could serve as an introduction to most existing speech recognition systems.

This consensus disappears when one turns a speech recognition system into a speech understanding system by adding a linguistic analyzer. The linguistic analyzers of different systems have tended to resemble each other very little. In some "loosely coupled" systems the linguistic analyzer remains aloof during lexical search, but later chooses among the word sequence hypotheses; in "tightly coupled" systems, the linguistic analyzer is intimately linked to lexical search. Some linguistic analyzers have an *ad hoc* flavour, others are firmly rooted in one of a variety of competing syntactic theories. Nevertheless, one of the themes of Chapter 3 is the emergence of a tentative convergence between linguistic analyzers, driven by the practical demands of the ATIS task and occurring despite considerable divergence of opinion at the theoretical level.

Differences between linguistic analyzers have arisen partly because of the variety of tasks for which speech understanding systems may be designed, partly because until recently most effort has been focused on other levels of the system, but mainly because the criteria for judging a speech understanding system are nebulous. A good speech recognition system is one that, other things being equal, recognizes a higher proportion of words than its rivals. What is a good speech understanding system? This question is another underlying theme in Chapter 3.

Chapter 3

Speech Understanding Systems for the ATIS Task

> The criterion of the scientific status of a theory is its falsifiability, or refutability, or testability.

Karl Popper, *Conjectures and Refutations* [Pop69]

3.1 Introduction

Many branches of natural language processing have made little progress towards building practical systems in the last decade. Speech recognition is the exception, even though it does not enjoy the luxury of working with typed input. Why has speech recognition made such rapid strides, while related fields such as machine translation have been relatively stagnant?

Apologists for other branches of natural processing would argue that the problems they must deal with are harder than they look, and that much of the improvement in speech recognizer performance has come from better hardware rather than deeper understanding of the problem. According to the latter argument, improved performance was inevitable - speech recognition systems were pushed forward by favourable technological winds.

I would argue otherwise. Improvements in speech recognizer performance were **not** technologically inevitable. Rather, the reasons for the successes of speech recognition technology are **cultural**. Researchers in this field have always tried to build complete systems that tackle a large, clearly defined

task, rather than toy systems that handle a few examples made up by the designer: what might be called an "engineering" rather than an "academic" approach.

Even more important, they have been willing to compare their systems with those of other researchers by evaluating performance on agreed-on benchmark tasks. These include digit recognition in continuous speech [Car92], the Resource Management task involving read speech with a 1000-word vocabulary [Pri88], and more recently ATIS. Much of the credit for this must go to agencies like ARPA and its successor DARPA, which funded such evaluations and many of the groups taking part in them.

A crucial aspect of the benchmark evaluations is that they involve two similar but disjoint data sets: the training set and the test set. The training set is released to system designers some time before the evaluation, and gives them a chance to fine-tune their systems. The test set is used **only** for testing, and **only** results on this set are considered valid indicators of system performance.

This evaluation methodology may appear obvious, but until recently it has not been applied in other branches of natural language processing. I was once present at a frank and extremely illuminating talk on machine translation given by Alan Melby [Mel88]. Melby noted the tendency of designers of machine translation systems to illustrate the performance of their systems with a single example that has been intensively studied: "Almost any machine translation system can produce brilliant results when the same text is run through it again and again with successive tuning. The power of tuning is well-known and has been given a name in AI research, namely, defining a microworld... In a machine translation system, difficulties arise when a tuned system is applied to a new text" [*ibid*, pg. 412].

At the end of his talk, Melby proposed the establishment of benchmark evaluations for machine translation systems. System designers would be given many details about the passage to be translated in advance - the source language, the target language, the vocabulary and the topic - but not the passage itself. No "tuning" of the system or "post-editing" of the translations would be permitted during the test; every translation would be scored by a jury of professional translators. Melby's proposal was received with hostility by an audience of experts in machine translation, who managed to devise an ingenious collection of reasons for rejecting it. A cynic might conclude that they were worried about the impact of such an evaluation on their funding.

The philosopher Karl Popper maintained [Pop69] that a theory could only be called scientific if it exposed itself to experimental refutation. Thus, a good physicist can devise hundreds of experiments that, if the results turned out a certain way, would prove Einstein's theory of relativity wrong. So far, every such experiment has failed to refute the theory, but it might have been otherwise. On the other hand, it is impossible to imagine an experiment that would convince an ardent Freudian or a believer in astrology that his pet theory is wrong. A scientific theory makes precise predictions and thus runs the risk of being proved wrong. In fact, scientific progress depends to a large extent on the refutation of good theories, which leads to the development of even better ones. A pseudo-scientific theory like astrology can never be proved wrong, and may therefore linger for millennia.

Analogously, progress in natural language processing depends on objective tests of system performance. Without such tests, the best ideas will not win out; instead, victory will go to those ideas whose proponents are the most eloquent, charming or influential. Speech recognition system performance is easy to measure - the percentage of words correct is an obvious metric. It is much harder to devise an objective metric for the performance of a speech understanding system. Thus, as speech recognition performance improved and the attention of researchers shifted towards speech understanding, there was a danger that speech researchers would acquire the bad habits of their colleagues in other branches of natural language processing and abandon common benchmarks.

Fortunately, this has not happened. "There has been a growing appreciation in the speech recognition community of the importance of standards for reporting performance. The availability of standard databases and protocols for evaluation has been an important component in progress in the field and in the sharing of new ideas. Progress toward evaluating spoken language systems, like the technology itself, is beginning to emerge" [Pri90 pg. 91]. In fact, researchers in other branches of natural language processing have begun to develop benchmarks for their own domains [Sun91].

The Air Travel Information Service task domain, known as ATIS, is a benchmark for speech understanding. The next section describes it in detail. Every participating group has criticised one or more aspects of ATIS, and indeed, parts of the ATIS definition are bizarre or inept. There is a concern that participating groups spend too much time optimising their systems to deal with the idiosyncrasies of ATIS, rather than breaking new ground.

Nevertheless, the existence of a standard procedure for comparing different systems, in which many of the most important speech understanding groups take part, is invaluable.

3.2 The Evolution of ATIS

3.2.1 Original Definition of ATIS

An article by P. Price [Pri90] gives a good description of the ATIS benchmark as originally conceived. The main features were described as follows:

- The ultimate goal is evaluation of systems for understanding speaker independent, spontaneous speech with a medium-sized vocabulary (around 1000 words).
- Independence of the training and test sets, as mentioned above. This ensures that the system has learnt some general rules from the training set, and it focuses the attention of the system developer on linguistic phenomena in proportion to their frequency of occurrence.
- Quantitative, automated evaluation methods rather than qualitative, subjective methods. Thus, it is better to devise measures such as the percentage of correct answers, which can be evaluated automatically by comparing system-generated answers with some standard, rather than measures of nebulous qualities like user-friendliness.
- The stress is on "black box" rather than "glass box" evaluation of the two components of a speech understanding system: the speech recognition component and the linguistic analyzer. That is, the overall performance of each component is more important than the performance of its parts. Groups can choose any combination of three evaluations:
  1. a test of the speech recognition component on spoken utterances using percentage of words correct as the metric;
  2. a test of the linguistic analyzer in stand-alone mode on transcripts of utterances using a function of the proportion of correct database queries generated as the metric;

3. a test of the complete speech understanding system on spoken utterances using the database query metric.

- A limited domain which is a plausible application for speech understanding systems.
- A "Wizard of Oz" data collection scenario designed to simulate the conditions under which the speech understanding system will be used. The subject believes he or she is interacting with a speech understanding system. In fact, a concealed transcriber wizard types what the user has said and sends it to the screen display, while a database wizard enters the appropriate database command and causes the answer (columns and rows from the database) to appear on the screen below the transcript.
- Classification of the collected utterances as acceptable or unacceptable. Unacceptable utterances include those that are grossly ill-formed, ambiguous, or unanswerable, and originally included those that depend on previous utterances (context dependent). System designers may train on any or none of these classes of data from the training set; only acceptable utterances are used for testing.
- To obtain reference answers for both the training and test data sets, the database queries typed in by the database wizard during the session with the user were reviewed and modified if necessary. A program to compare the answers generated at the test sites with the reference answer for each utterance was developed.

There were several reasons for choosing air travel as the domain:

- A real, widely-used database, the Official Airline Guide, was available;
- The domain includes a variety of topics, such as schedules and fares, the services available on different flights, information about airplanes, and ground transportation;
- A wide pool of users is familiar with the domain;
- The domain can be scaled as the technology becomes more advanced - that is, one can imagine the same domain supporting more sophisticated man-machine spoken interaction;

- The domain resembles many other possible applications of speech understanding.

3.2.2 Criticisms of ATIS

The first evaluation took place in June 1990, the second in February 1991, and the third in February 1992. Of systems participating in the first evaluation, only the CMU Phoenix system was tested on recorded utterances as well as on transcripts; the linguistic analyzers of all other systems were tested in stand-alone mode as text processors. Since then, the number of groups participating in all three tests - of the speech recognition component alone, of the linguistic analyzer alone, and of the complete speech recognition system - has increased at each evaluation.

Participating groups have criticised several aspects of the original definition of ATIS; see the discussion in [Pol92]. Some of these criticisms have already been dealt with, others may be resolved in the near future. The criticisms are as follows:

- Recall that probabilistic models require large amounts of data for estimation of parameter values. Since researchers in speech recognition began applying probabilistic approaches at most levels of their systems, the complaint "We need more training data!" has become a cliché in the speech recognition community. All the groups participating in the first two ATIS evaluations complained that inter-speaker variability in accent, lexical choice, syntax, and spontaneous speech characteristics such as frequency and type of interjections was far too great to be adequately represented by the 1000 or so utterances in the training set. This problem has now been resolved by MADCOW, a project that gathers and pools data from many participating sites [Hir92]. As of February 1992, 10,000 utterances from 280 speakers at five sites had been collected, with 1,000 utterances set aside for the February 1992 test and the same number for the November 1993 test.

- There is a serious problem with the "wizard" scenario described above: the wizard is much smarter than a machine. He will answer questions a real speech understanding system could not, and thus encourage users to become increasingly bolder and idiomatic in their use of language as they interact with him. Transcripts from the user-wizard interaction

will contain little "error recovery" behaviour, though error recovery will be very important for a real speech understanding system. Clearly, the wizard scenario should only be employed in the first phase of an iterative bootstrapping effort to build speech understanding systems; in the later phases, the systems themselves should be used to collect data. This is exactly what has happened. To obtain the February 1992 MADCOW data, two of the groups employed the complete system to convert utterances into database queries, while the other three groups employed a transcription wizard to type the sequence of words into the linguistic analyzer.

- Some sites complained that there were too many words in the test sentences that were absent from the training sentences. This complaint seems unreasonable; in the real world, different people make different lexical choices.

- ATIS speech is recorded under almost ideal conditions, with no background noise. Thus, speech recognition performance on ATIS data gives no indication of how robust a system would be in a real-life, noisy setting, such as an airport or travel agent's office. There are no indications that this aspect of ATIS will be modified.

- Real-time performance is not one of the ATIS criteria; as far as I know, BBN is the only site that claims to have achieved it. Presumably, it was felt that inclusion of this criterion would unduly reward groups that could afford the most expensive signal-processing hardware.

- There has been a continuing struggle over the definition of correct answers. Since the first evaluation, the release of each set of training data has been accompanied by a document called "Principles of Interpretation". Some examples of rules found in this document: "around" X a.m. means from $(X - 1) : 45$ to $X : 15$; a snack counts as a meal if someone wants to see "flights with meals"; "flights between A and B" means only flights *from* A *to* B. Questions beginning "how many..." may be answered either literally with a number or more pragmatically with a display of data. Similarly, some questions may be answered literally with "yes" or "no", or pragmatically. The document has not eliminated all controversy about correct answers, but it has kept it

within reasonable bounds. Disputes about how much information to provide in response to a question have been resolved since February 1992 by providing minimum and maximum reference answers - if the system generates all the information in the minimum answer and no more than the information in the maximum answer, the response is marked correct.

- What penalty should be assigned to a partial answer, to a wrong answer and to no answer? This has been one of the hottest issues in the ATIS community. At present, the main metric for evaluating the output of the linguistic analyzer is the *weighted error*, defined as $(2 * \%false + \%no\_answer)$. Many groups feel that this excessively penalizes answers that are almost right, to the detriment of user-friendliness. Consider the following question-response pair:

  LIST FLIGHTS FROM BOSTON TO PITTSBURGH LEAVING AT 3 PM

  Here are flights from Boston to Pittsburgh:

  (displays flights from Boston to Pittsburgh)

  This would earn twice the penalty of the response "No answer" because the detail about 3 pm was missed, even though the flights displayed on the screen include those the user wanted to know about. There does not seem to be any consensus on this issue, so the current scoring scheme will probably survive.

- Almost a third of the utterances collected for the first evaluation were questions about the meanings of codes displayed on the screen, such as "What does restriction AP/80 mean?" The original screen displays were cluttered and cryptic; all the user received in response to a question was the appropriate rows and columns from the Official Airline Guide. Since the OAG is designed to be accessed by travel agents rather than untrained members of the public, the users' bewilderment should have been predicted. In more recent releases, the column headings and codes have been made much easier to understand, and the proportion of questions about them has decreased dramatically. Nevertheless, some groups feel that the system responses should be made

far more user-friendly. As described in the next section, MIT has devised a screen display based on an airline ticket for a dialogue version of ATIS; AT&T's ATIS system outputs synthesized speech summarizing the result of the database query.

- Although users are given scenarios before interacting with the system, these are complicated, vague, and often seem to have little to do with the resulting utterances. Questions found in the first release include "What is the payload of an African Swallow?" and "Show me Daphne's itinerary". It was never clear whether the user was a member of the public or a travel agent. The mode of communication was unrealistic: most members of the public would find it convenient to telephone for air travel information, rather than leave their homes and offices to talk to a screen. (The AT&T group is now working on a telephone version of its ATIS system). Finally, since the system cannot pretend to take bookings or sell tickets, an excellent opportunity for purposeful dialogue has been lost; many interactions give the impression of aimless chatter. The MIT dialogue version takes bookings, but is not part of the ATIS benchmark evaluation - unlike the main MIT version. In general, though sites participating in MADCOW have some liberty to define their own scenarios, the resulting utterances still seem somewhat unrelated to them.

- The main cause of the irrelevance of the scenarios is that ATIS benchmark systems are forbidden to engage users in dialogue. One of several negative results of this policy is that users currently try to cram linguistically unnatural amounts of information into a single question, because they know the system cannot ask follow-up questions. It seems absurd to cripple the systems in this manner, but there are good administrative reasons for it. Currently, recorded utterances are sent to the participating sites on CD-ROMs for training and testing; this protocol does not accommodate dialogue. User utterances that refer to previous utterances, such as "show me those flights again except for the ones on United" are allowed, but they are labelled 'D' (for "context-dependent") to distinguish them from the stand-alone 'A' (for "acceptable") utterances. If the ability to carry out dialogue were an ATIS criterion, real-time or near real-time performance would become

obligatory, giving an unfair advantage to groups with the most expensive hardware. Furthermore, some groups prefer the evaluation of non-dialogue systems because it is more "objective" - the performance criteria for a dialogue system are unclear.

- The opposite point of view also exists. According to this point of view, the current "objective" criteria should be supplemented by more "subjective" criteria that attempt to measure the effectiveness and user-friendliness of a speech understanding system. Like the closely related issue of dialogue, this remains unresolved.

3.2.3 The Future of ATIS

Two major, interrelated issues were mentioned above - criteria that measure effectiveness and user satisfaction with a system, and dialogue. These issues are discussed in an interesting paper by members of the MIT group [Pol92]. The MIT researchers have carried out experiments exploring new performance metrics for the linguistic analyzer. The metrics fell into two categories:

1. Metrics for end-to-end evaluation, i.e. for measuring task completion effectiveness. Subjects were asked to discover a certain fact by questioning the system; when they had discovered it, they were told to say "End scenario. The answer is ..." and then give the answer. The metrics included the number of user queries before task completion, the number of successful and unsuccessful ("No answer") queries, the number of times the task as a whole was completed successfully, and the task completion times.

2. Metrics for log file evaluation. Pairs of subject queries and system responses were extracted from log files; seven evaluators then rated the query as clear, unclear, or unintelligible, and the system response as correct, partially correct, incorrect, or "error message". Reassuringly, there was a high degree of consensus among the evaluators.

The MIT researchers then conducted a more ambitious experiment designed to compare two MIT linguistic analyzers, the *full parse* system and the *robust parse* system (to be described in a later section). Subjects took

each of these through four scenarios. All the metrics mentioned above were applied to the interactions between the subjects and the two systems, along with a metric for user satisfaction - subjects were asked which of the two systems they preferred. All these metrics favoured the same system on all scenarios (the robust one) except for one anomalous scenario.

The importance of these experiments is that they suggest realistic methods for evaluating the effectiveness and user-friendliness of a speech understanding system in a problem-solving context. The strong correlation between metrics is reassuring; it suggests that they all measure aspects of the same phenomenon. Adoption of these or similar metrics might remove the main objection to including dialogue systems in ATIS: the difficulty of objectively evaluating such systems. However, one would have to show that they can be used to compare completely different systems, as well as different versions of the same system.

This is precisely the question addressed in [Pri92], which describes an exercise to compare the performance of the MIT and SRI systems. To eliminate the problem of user variability, the design of the exercise was within-subject; that is, each user assesses both systems. Half the users would work with the SRI system first and then the MIT system, while the other half would work with the systems in the opposite order. The main difficulty was the need to have both systems running in the same place at the same time. Hardware, software, and licensing compatibility problems prevented this from happening. As a compromise, the evaluation ultimately pitted the SRI system against a hybrid system made up of the SRI speech recognizer and the MIT linguistic analyzer.

The experimental subjects were asked to play out two scenarios, A and B, on each of the two systems. The metrics for the evaluation were similar to those employed in the MIT experiments:

- User satisfaction, obtained by asking the subject to score his satisfaction with the system on a scale from 1 to 5;
- Correctness of the answers extracted by the subject from the system;
- Total time to complete the scenario;
- Waiting time between the subject's utterance of his first question and the system's response;

- User response time (from appearance of the system's response to the subject's next utterance);
- Recognition word error rate;
- Frequency and type of diagnostic error messages.

The authors of [Pri92] stress the importance of finding metrics for interactive systems that reflect user satisfaction and user efficiency, and the need for a "reality check" on the role of the current ATIS metrics in promoting technological progress. Although within-subject experimental design is the optimum, the MIT-SRI exercise showed that this might pose practical difficulties. The authors therefore suggest log file evaluation or evaluation of taped subject-system interactions by a neutral "jury" of experts as another worthwhile approach.

Many of the arguments for new metrics that might allow dialogue systems to be included in ATIS benchmark tests come from researchers in the MIT group. This may have something to do with the existence of an impressive, ATIS-related dialogue system built by these researchers [Sen91]. This system builds on earlier work on the MIT VOYAGER system [Zue91]. Figure 3.1 shows part of a dialogue between a user trying to make a round-trip flight reservation and the MIT system (taken from [Sen91 pg. 357]). Note that the dialogue is mixed-initiative; the system can ask clarificatory questions, but responds smoothly if the user chooses to incorporate additional answers in his answer. As the MIT researchers point out, the system thus occupies a middle ground between systems that passively respond to user requests without asking questions (as in the ATIS benchmarks), and menu-driven systems that allow the user a very narrow range of choice. The system also has the capability (not shown in the example) of asking the user to repeat a request by printing out "I'm sorry, I had trouble understanding that" or a similar diagnostic.

The system has a mechanism for remembering information mentioned earlier and interpreting anaphoric references like "the third one". To allow this, it keeps a history table containing such objects as flights and dates, represented as frames, as well as the current ticket, previously booked tickets, and previously displayed tables. Another nice feature of the system is that at a certain point in the flight reservation scenario, a facsimile of a partly filled-in airline ticket appears on the screen, as shown in figure 3.2. This enables

Subject: I WANT TO GO TO SAN FRANCISCO
ATIS: Where from?

Subject: BOSTON
ATIS: What date will you be travelling on?

Subject: I'LL BE LEAVING BOSTON NEXT SUNDAY
AND RETURNING THE FOLLOWING TUESDAY
ATIS: These are the flights from Boston to San Francisco
on Sunday January 6:

AIRLINE	*NUMBER*	*FROM*	*TO*	*DEPARTURE*	*ARRIVAL*	*STOPS*
United	*21*	*BOS*	*SFO*	*8:00 am*	*11:31 am*	*0*
United	*343*	*BOS*	*SFO*	*8:40 am*	*1:36 pm*	*1*
AA	*813*	*BOS*	*SFO*	*11:49 am*	*5:58 pm*	*1*
Cont	*1765*	*BOS*	*SFO*	*5:40 pm*	*11:10 pm*	*1*
United	*93*	*BOS*	*SFO*	*5:45 pm*	*9:27 pm*	*0*
TWA	*61*	*BOS*	*SFO*	*6:10 pm*	*9:37 pm*	*0*

Subject: WHERE DOES THE THIRD ONE STOP
ATIS: American flight 813 from Boston to San Francisco
on Sunday January 6 stops in the following places:

AIRLINE	*NUMBER*	*FROM*	*TO*	*DEPARTURE*	*ARRIVAL*	*STOPS*	*AIRPORT*
AA	*813*	*BOS*	*SFO*	*11:49 am*	*5:58 pm*	*1*	*DFW*

Figure 3.1: Simulated Dialogue with MIT Dialogue System [Sen91, pg.357]

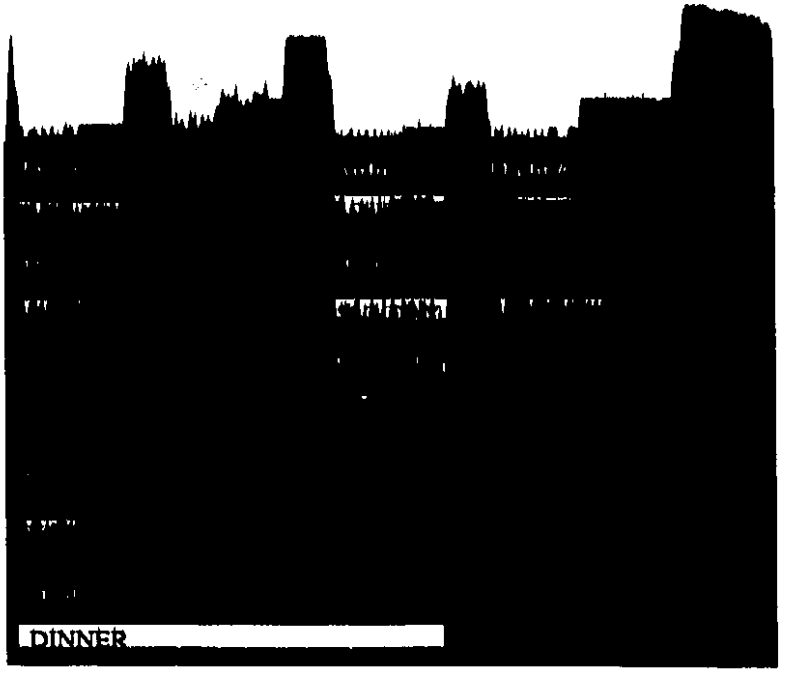


Figure 3.2: Ticket Facsimile shown by MIT Dialogue System [Sen91, pg.357]

the user to see what further information the system needs from him, and whether there have been any misunderstandings that need to be corrected.

In developing the dialogue version of their ATIS system, the MIT researchers concentrated on the goal-oriented task of making flight reservations, and ignored potential dialogues focusing on ground transportation, aircraft capacity, and so on that occur frequently in the ATIS training and test data. They argue that flight reservation is a well-structured goal with well-defined subgoals that is comprehensible to a large pool of untrained users, which forms an ideal testbed for dialogue systems.

If, as the MIT researchers hope, the ATIS benchmarks are redefined to include dialogue systems, the long-term effect on the speech community would

be dramatic. Up to the present, research effort has focused on improving the percentage of words correctly recognized. Although this would remain an important goal (an indispensable one for dictation systems), attention would shift to dialogue as a strategy for recovering from misrecognition. Language models would take dialogue state into account; indeed, the study of discourse phenomena would become central at most levels of speech recognition. Thus, the MIT dialogue system may be a preview of the future of ATIS.

3.3 Linguistic Analyzers for ATIS

The remainder of this chapter describes the linguistic analyzers that had been developed for the ATIS task at the time of the February 1992 DARPA Workshop [DAR92], the latest date for which published material is available.

3.3.1 The SRI System

SRI's Template Matcher [Jack91, App92], is the linguistic analyzer for a speech understanding system closest to the keyword-based one described in this thesis. The input to the Template Matcher is the top hypothesis generated by the speech recognition component, which uses a bigram language model. Interestingly, SRI went into the first ATIS evaluation with a unification grammar-based parser [Moo90]. By the time of the next ATIS evaluation, the SRI group had built the Template Matcher, based on very different principles. It simply tries to fill slots in frame-like templates. For the 1991 evaluation, there were 8 templates: these dealt with flights, fares, ground transportation, meanings of codes and headings, aircraft, cities, airlines and airports. The different templates compete with each other on each utterance; all are scored, and the template with the best score generates the database query (provided its score is greater than a certain "cut-off"). Slots are filled by looking through the utterance for certain phrases and words.

Here is a typical example, taken from [Jack91 pg. 191]. For the utterance

> Show me all the United flights Boston to Dallas nonstop on the third of November leaving after four in the afternoon

the following *flight* template would be generated:

```
[flight, [stops,nonstop],
         [airline,UA],
         [origin,BOSTON],
         [destination,DALLAS],
         [departing_after,[1600]],
         [date,[november,3,current_year]]].
```

Words in the utterance may contribute to selection and filling of a template in various ways:

- They may help to identify the template - the occurrence of the word "downtown" is a good indicator of the *ground transportation* template;
- They may fill a slot - like "Boston" and "Dallas" in the example;
- They may help to indicate what slot a phrase goes in - like "from" or "to" preceding a slot-filling phrase.

However, many words are irrelevant: "please", "show me", "would you", and so on. The Template Matcher simply skips over these.

The score for a template is basically the percentage of words in the utterance that contribute to filling the template. However, certain keywords that are strongly correlated with a particular template will strongly boost the score of that template, if they occur in the utterance. For instance, the occurrence of "how much", "fare", or "price" boosts the score of the *fare* template; the occurrence of "what is", "explain", or "define" boosts the score of the *meaning* template. However, if the template has no slots filled, it is assigned a score of zero; or if the system has tried to assign two or more values to the same slot in the template, it is aborted.

If the best score does not exceed a certain numerical "cut-off", the system responds with "no answer" rather than with the template that yielded the best score. Recall that ATIS scoring penalizes wrong answers rather harshly, so that "no answer" is often a preferable response; to optimize the performance of their system on tests, the SRI researchers set the cut-off to the value that yielded the best results on training data.

For the 1991 evaluation, the SRI researchers called the Template Matcher when the conventional parser failed; for the 1992 evaluation, they scrapped the conventional parser and kept the Template Matcher [App92]. Several

improvements were made in the Template Matcher. The 1991 version could only fill slots with fixed words or phrases - it had no ability to deal with general phrase categories like numbers, dates, and times. In the 1992 version, phrases falling into these categories in the incoming utterance are parsed by special grammars, and are then put into slots by the matcher. Slots are filled by matching regular expressions - for instance, "from" followed by a city or airport name would cause the name to be put into the origin slot of the *flight* template.

For the 1992 version, the number of templates was increased to 20, and the number of slots to 110. Templates now contain illocutionary force markers to note whether the utterance is (for instance) of the "yes-no" or "how much" type. Special mechanisms deal with certain types of false starts and complex conjunctions.

The major difference between the 1991 and 1992 versions of the Template Matcher is that the latter contains a context handling mechanism for class 'D' (context-dependent) queries; the 1991 version was only designed to work with class 'A' (context-independent, acceptable) queries. As in the MIT dialogue system, the context handling mechanism allows slots to be inherited from previous utterances by the same user. Note that if the user's first utterance is wrongly interpreted, his subsequent context-dependent utterances may all be correctly recognized yet misunderstood: this might be called the "getting off on the wrong foot" problem. If systems were allowed to ask users for confirmation from time to time, this problem would be relatively unimportant. Since the ATIS evaluations do not permit dialogue, the SRI researchers devised several ingenious mechanisms for preventing the problem. For instance, when the system gives "no answer" to a query, subsequent context-dependent queries also yield "no answer", until a query that sets a completely new context arrives. This strategy yields the best evaluation results and was consequently adopted for the test, though as the SRI group points out, "it would be a ridiculous way for a system to behave when interacting with a real user" [App92]. During MADCOW data collection, on the other hand, the system generates an answer when possible, on the ground that users prefer slightly incorrect answers to "I don't understand".

The SRI system performed well on both evaluations. In 1992, it placed second in both the natural-language-only test on transcripts of utterances and the spoken language test (in which the output of the SRI recognizer is analyzed by the Template Matcher) [Pal92]. The SRI researchers believe

that about 80% of cases where the wrong template was generated could be remedied within the framework of the Template Matcher - for instance, by adding a new template or a new slot. The rest would require an approach that yielded more information about the structure of the sentence. They conclude that their approach is well-suited to a domain of about the same complexity as ATIS.

3.3.2 The CMU System

Figure 3.3 shows the structure of PHOENIX, the CMU system's caseframe parser [War92, War91, War90]. In many respects, this system is very similar to SRI's Template Matcher. As in the SRI system, the input to PHOENIX is the top hypothesis of the speech recognition component. The system considers different frames in parallel; the score for a frame is the number of input words it accounts for. Unlike the SRI system, which aborts a frame if an attempt is made to overwrite a slot that has already been filled, PHOENIX allows the overwrite to take place - if a slot is filled several times, the last slot-filler prevails.

Scored frames generated by PHOENIX are optionally post-processed by MINDS-II, a "knowledge-based correction module". In addition to dealing with context-dependent utterances, this module uses domain knowledge and syntax to detect erroneous parses, skipped information, and out of domain requests. For instance, if someone asks "what is the shortest flight from Dallas to Fort Worth?" this module informs the user that these two cities share a single airport [You91]. Before deciding between the scored frames provided by PHOENIX, this module looks at word strings **not** accounted for by a given frame to determine if they are important. The module may also make deductions about information missing from a frame and supply it before generating the SQL database query.

The official results are very interesting. For the natural language test, CMU tested the PHOENIX system alone and with the MINDS-II postprocessor. PHOENIX alone obtained the best results of any system participating; PHOENIX plus MINDS-II performed significantly worse and was beaten by several other systems [Pal92]. As far as I can tell from the vague wording of [War92] only PHOENIX was used for the spoken language test, with mediocre results. This is partly due to PHOENIX's policy of guessing at an answer whenever possible, rather than returning "No answer". The CMU

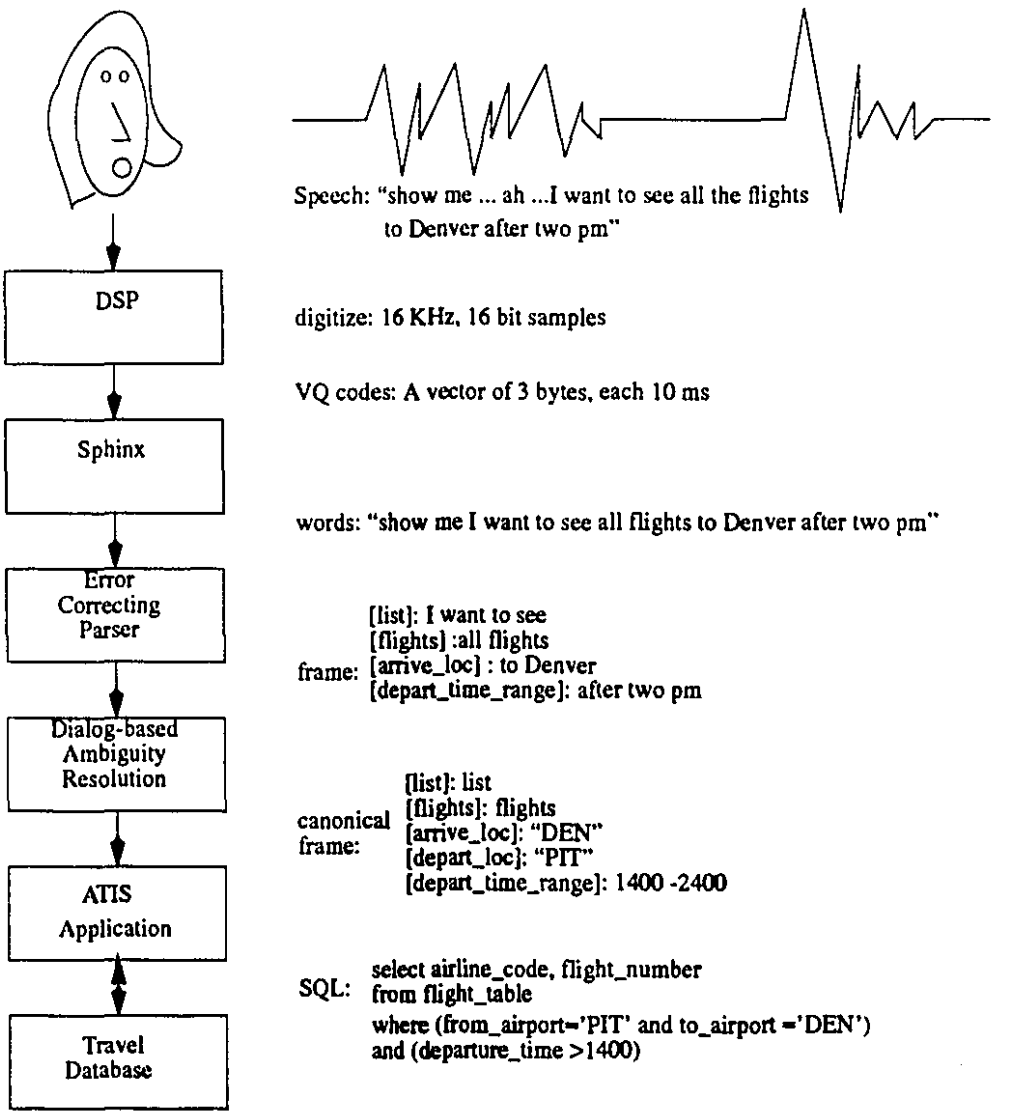


Figure 3.3: Structure of the Phoenix System [War91, pg.103]

group's unofficial spoken language results show PHOENIX plus MINDS-II performing better than PHOENIX alone, but only because MINDS-II returns "No answer" on many utterances for which PHOENIX alone would not have returned "No answer". MINDS-II actually reduces the proportion of correct answers.

Thus, the SRI-style template matcher at CMU performs very well on transcripts (and might perform well on speech if it returned "No answer" more often in doubtful cases); further processing by an expert system may actually impair performance.

3.3.3 The BBN System

BBN's speech recognizer is called HARC. Its speech recognition component is called BYBLOS; its linguistic analyzer is called DELPHI, and is made up of a chart-based unification parser and a robust fallback module [Kub92]. BBN has traditionally taken a conventional syntactic approach to language interpretation; the fallback module is a recent addition that was absent during the February 1991 evaluation [Aus91]. An interesting feature of the chart parser is that it incorporates information about rule firing probabilities, estimated from training data.

The BBN group has a distinctive approach to the design of spoken language systems, characterized by ingenious use of the N best hypotheses. This approach is described in [Ost91, Schw92]. It involves generating the N best hypotheses using a fast, simple algorithm, then repeatedly rescoring these hypotheses by means of more complex, slower algorithms. In this manner, several different knowledge sources can contribute to the final result without complicating the control structure or significantly slowing down derivation of the final result. In the February 1992 tests, BYBLOS first generated an N-best list using discrete HMMs and a bigram language model, reordered the list using cross-word-boundary triphone models and semi-continuous HMMs, then reordered it again using a trigram grammar.

The fallback understanding module within DELPHI is called if the unification chart parser fails [Sta92]. It tries to generate a robust interpretation from parsed fragments left over from the first, failed parse. The fallback module is itself made up of two parts: the Syntactic Combiner and the Frame Combiner. The Syntactic Combiner uses extended grammatical rules that can skip over intervening material in an attempt to generate a complete

parse. If the attempt fails, the Frame Combiner tries to fill slots in frames in a manner similar to that of the systems described above. The main difference is that because it operates on the output of a parser, it can fill slots with complicated phrases such as "the airport closest to Washington DC". The Frame Combiner uses many pragmatic rules obtained through study of training data which could not be defended on abstract grounds. For instance, interpretations which combine flight and ground transportation information are ruled out because they are never observed in the data, even though a query like "Show flights to airports with limousine service" is theoretically possible.

To the surprise of the BBN researchers, the fallback module worked better if only the Frame Combiner - but not the Syntactic Combiner - was included. Both were included in the linguistic analyzer used for the February 1992 benchmarks. In this evaluation, DELPHI did reasonably well on the natural language test and HARC as a whole did better than any other system on the spoken language test. According to [Sta92] both results would have been even better if the version of the system with the Frame Combiner and without the Syntactic Combiner had been used.

The BBN researchers carried out experiments to determine the best way of combining the chart parser with the fallback module, given that the output of the BYBLOS speech recognizer is the N best hypotheses [Kub92]. Note that if only the parser were used, it would go through the hypotheses from most to least probable until it found one that was globally parsable; if there was none, presumably it would return "No answer". If only the fallback module were used, it would always generate some interpretation from the top hypothesis - it has no rejection criterion. Thus, it is not obvious how best to combine the parser and the fallback module. The BBN group obtained the best results by setting N to 5. If the chart parser was unable to obtain a global parse for any of the 5 hypotheses, the fallback component generated an interpretation (presumably from the best hypothesis).

3.3.4 The MIT System

The linguistic analyzer of the MIT system went through the same evolution as BBN's DELPHI linguistic analyzer: originally it consisted only of a syntactic parser, but subsequently a robust matcher was added as backup when the parser fails [Zue92, Sen92]. As in DELPHI, this matcher fills slots in a

frame with parsed phrases found during the failed global parse. Although the MIT system is capable of generating N best hypotheses which can then be reordered by the linguistic analyzer, the version of the system used for testing seems to have employed only the top hypothesis.

Recall that, as described in a previous section, the MIT researchers have built a version of their system (not used in ATIS evaluations) that carries out dialogue. The unusual aspect of MIT's robust matcher is that it exploits features of the history mechanism built to make dialogue possible. During dialogue, this history mechanism allows slots to be inherited from previous utterances. Similarly, the robust matcher "remembers" slots filled earlier in the same utterance. Only one adjustment was needed to make the history mechanism suitable for sentence-internal parsing: overwriting of slots was forbidden (it occurs between utterances in the dialogue version).

Another feature of the MIT system deserves mention. Although a bigram language model first generates 50 N-best hypotheses, these are then reordered by a probabilistic LR language model based on the grammar used by the global syntactic parser. This parser has a finite number of states Q_j. One can employ it to parse a training corpus of sentences, count the number of times the word w_i occurs when the parser is in state Q_j, and thus obtain an estimate of $P(w_i|Q_j)$. Since the probabilistic LR grammar gives an estimate of the probability $P(Q_j|w_0, ..., w_{i-1})$ that the parser is in state Q_j at the time it encounters word w_i, the probabilistic LR language model estimates $P(w_i)$ as

$$P(w_i|w_0, ..., w_{i-1}) = \sum_j P(w_i|Q_j)P(Q_j|w_0, ..., w_{i-1}).$$

3.3.5 The Paramax-Unisys System

The semantic module of this system attempts to instantiate the arguments of case frame structures called "decompositions" [Nort92, Nort91]. These arguments are assigned thematic role labels such as agent, patient, and source. Two types of syntactic constraints apply to role fillers: **categorial** constraints which specify that a role filler must be of a certain grammatical type, and **attachment** constraints which specify that a role filler must be within the phrase headed by the predicate of which it is the argument. In many cases, the attachment constraints turn out to prevent the correct interpretation - for instance, they rule out "What flights do you have to Boston?"

(the permitted form would be "What flights to Boston do you have?") and "I want the $50.00 flight" (costs apply to fares, not flights). These attachment constraints have been relaxed somewhat [Nort91]. However, this system still incorporates global, fairly rigid syntactic constraints.

Recent additions to the system aim at enhancing dialogue. They include non-monotonic reasoning for making tentative inferences which can be withdrawn on the basis of new information supplied by the user, resolution of implicit anaphoric references, and paraphrasing of the user's utterance, enabling the user to check and confirm or correct his request. The performance of this system on the last two tests has been very disappointing: only the AT&T system has performed worse. As we will see, the AT&T system is based on radically different principles from the other systems; of the more conventional systems, the Paramax system is the only one that has **not** incorporated some kind of robust matcher.

3.3.6 The AT&T System

The AT&T system is a bold departure from previous approaches to speech understanding [Pie92a, Pie92b, Pie91]. The emphasis of the AT&T researchers is on a linguistic analyzer that can carry out unsupervised learning, reducing the need for human intervention. This emphasis also inspired the KCT-based linguistic analyzer described in this thesis, although its structure is that of an SRI-style robust matcher which bears no resemblance to the AT&T system.

Figure 3.4 shows the structure of the AT&T system. A hasty reading of this group's papers might give the impression that almost all the work done by the system is performed by rules automatically generated from training data. Actually, the automatically generated rules are confined to the *conceptual decoding* module shown in the figure. The *template generation* module is at least equally important, and is made up of hand-coded rules. The AT&T researchers argue that there are not enough training data to generate rules for both these modules automatically, though this is possible in principle. The same design decision - to use a mixture of hand-coded and learned rules - was made for the KCT-based linguistic analyzer, again because there are not enough training data to learn all rules. In the cited papers, the AT&T researchers describe the *conceptual decoding module* in great detail but fail to provide a thorough description of the rules in the *template generator*.

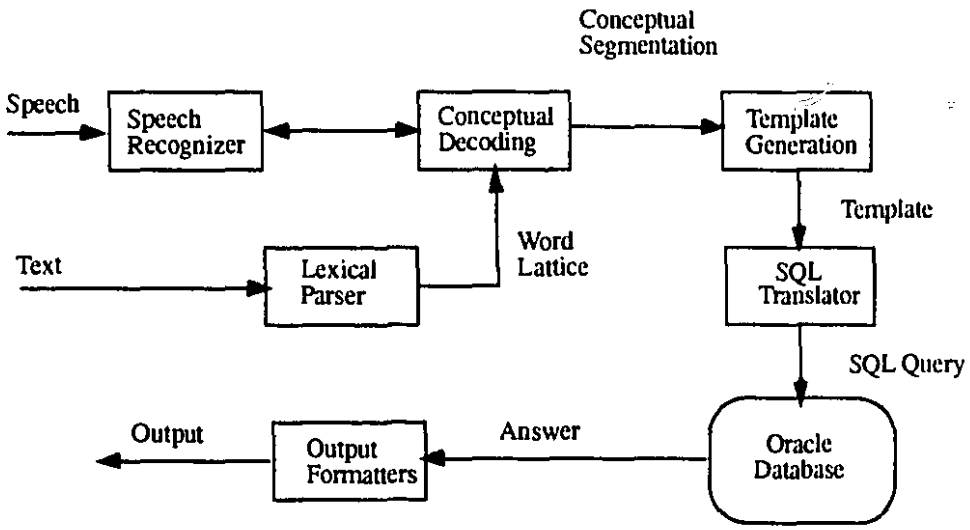


Figure 3.4: Block Diagram of AT&T System [adapted from Pie92b]

The *conceptual decoding* module segments and labels a string of words. (The *lexical parser* shown in the figure simply generates alternate arrangements of compound words and acronyms; for instance, on the substring "B SEVEN FOUR SEVEN" it generates "B747", "B 747", "B7 47", and so on). The labels for the word substrings are drawn from a set of predefined concepts. For instance, here is the labelling of the word sequence "Please list all flights between Baltimore and Atlanta on Tuesdays between 4 pm and 9 pm" [Pie91]:

```
DUMMY:  Please
QUERY:  list all
OBJECT: flights
origin: between Baltimore
destin: and Atlanta
day:    on Tuesdays
time:   between 4 pm and 9 pm.
```

The *template generator* takes labelled word sequences as input, converting them to a form from which SQL code can be generated.

The design of the *conceptual decoding* module is based on HMMs - here lies the originality of AT&T's approach. Each word sequence is considered to be an HMM, with the words being the observations and the concepts being the states. We want to find the sequence of words $\mathbf{W}$ and the sequence of concepts $\mathbf{C}$ maximizing $P(\mathbf{W}, \mathbf{C}|A)$, where A is the acoustic evidence. This is equivalent to maximizing

$$P(A|\mathbf{W}, \mathbf{C})P(\mathbf{W}|\mathbf{C})P(\mathbf{C}).$$

If we make the approximation

$$P(A|\mathbf{W}, \mathbf{C}) = P(A|\mathbf{W})$$

the first of these three terms is taken care of by the speech recognizer's HMMs. The remaining terms are

$$P(\mathbf{W}|\mathbf{C})P(\mathbf{C}) = \prod_{i=2} P(w_i|w_1, ..., w_{i-1}, \mathbf{C})P(w_1). \prod_{i=2} P(c_i|c_1, ..., c_{i-1})P(c_1).$$

The AT&T system approximates $P(w_i|w_1, ..., w_{i-1}, \mathbf{C})$ by $P(w_i|w_{i-1}, c_i)$ and $P(c_i|c_1, ..., c_{i-1})$ by $P(c_i|c_{i-1})$.

By means of the Viterbi training algorithm for HMMs, these probabilities can be estimated from a training corpus of segmented, labeled word sequences. The *conceptual decoding* module uses them as parameter values for a Viterbi segmenting, labeling process on new word sequences. In a recent enhancement of the model, morphologically or semantically related words are grouped together to form word classes on an *ad hoc* basis [Pie92a].

Beginning with a corpus of 532 sentences they segmented and labeled themselves, the AT&T researchers carried out the following iterative procedure:

1. Train the *conceptual decoding* module on segmented, labeled sentences;
2. Use this module and other components of the system on new training sentences to generate answers;
3. Compare the answers to the reference answers;
4. Add segmented, labeled sentences that generated correct answers to the training data and return to step 1.

Sentences that still produce incorrect answers after several cycles must be hand-labeled and added to the training data; however, this procedure clearly reduces the amount of human intervention needed.

For the ATIS spoken language tests, the input to the linguistic analyzer was the top hypothesis of the N-best hypotheses. As noted earlier, results for both the natural language and the spoken language tests were inferior to those of other groups. However, the novelty of AT&T's approach and the way in which human effort is minimized make this system worthy of careful study.

My main criticism of this work is the omission of important details from the group's published papers. It is not clear whether the version of the system used for the spoken language test was trained on hand-labeled transcripts, or on hand-labeled output from the recognizer, though this decision might have a strong effect on performance. A more fundamental problem is that problems with the representation are alluded to but never discussed. The authors mention the problem of adapting the representation to queries with multiple flight identifications, origins, and destinations [Pie92a]. They ignore the possibility of concepts that are split by intervening words. For instance,

in "show me early flights on Friday afternoon for next week" the *day* concept, "Friday ... next week" is split, and so is the *time* concept, "early ... afternoon". This kind of split is common in ATIS data, and poses a grave problem for the very linear approach to parsing favoured by the AT&T group.

3.4 Summary

The preceding survey of linguistic analyzers for ATIS speech understanding systems contains a clear message. Like sailboats with different headings that are blown off-course by a gale, the groups participating in ATIS began with very different ideas about system design but have been compelled by the task to move in the same direction.

Groups like SRI and CMU attained good early results with robust matchers, and therefore kept and improved them. Groups like BBN and MIT that originally scorned robust matchers in favour of parsers based on syntactic theory found themselves obliged to build robust matchers as backups when the parsers fail, which occurs frequently. Apart from AT&T's system, which is radically different from any of the others, the worst-performing system is Paramax's, which relies entirely on syntax. S. Seneff sums up the characteristics of the syntax-driven approach as follows [Sen92]: "While providing strong linguistic constraints to the speech recognition component and a useful structure for further linguistic analysis, such an approach can break down in the presence of unknown words, novel linguistic constructs, recognition errors, and some spontaneous speech events such as false starts".

A robust matcher, on the other hand, seems to work best when it can formulate rules based on word and phrase categories - that is, when a local parser has identified and labeled word groups constituting a category within a word sequence before the robust matcher sees the sequence. Thus, most of the ATIS systems seem to be converging on an approach that employs syntax locally and semantics and pragmatics globally. It would be a mistake to dismiss this as an artifact of the ATIS task. In my opinion, the ATIS researchers have made a genuine linguistic discovery: some types of spoken language are made up of islands of syntactically correct phrases separated by verbal "noise", with weak or non-existent global syntactic constraints.

Each of the systems described above incorporates ingenious ideas which are independent of the system architecture, and may therefore be borrowed

by future systems with a different architecture. For instance, the CMU idea of employing an expert system to post-process the output of the robust matcher has great potential, though it yielded little improvement in the CMU implementation. The idea, associated with BBN, of joining together diverse knowledge sources in series so that each can contribute to the final result via rescoring of the N-best hypotheses is elegant and powerful; as real-time performance becomes more important, it seems likely to become more and more prevalent. Finally, though there are problems with the division of labour and the knowledge representation in the AT&T system, the AT&T group's emphasis on automatic learning of rules from training data deserves to be emulated.

Chapter 4

Learning Patterns in Strings

> From the viewpoint of empirical research, one of the main difficulties in comparing various algorithms which learn from examples is the lack of a formally specified model by which the algorithms may be evaluated. Typically, different learning algorithms and theories are given together with examples of their performance, but without a precise definition of "learnability" it is difficult to characterize the scope of applicability of an algorithm or analyze the success of different approaches and techniques.

M. Kearns, *The Computational Complexity of Machine Learning* [Kea90a]

4.1 Introduction

This thesis proposes a new method for learning from examples the rules that carry out part of a certain task in speech understanding systems. N. Prieto and E. Vidal call this task *formal transduction* and give a good definition of it [PrV91, pg. 789]: "Formal transduction ... (maps) input *acoustic strings* representing uttered sentences into output *semantic strings* representing the actions or semantic messages that are conveyed by these sentences". Theoretical work in both linguistics and formal language theory has focused on syntax rather than semantics, so formal transduction remains mysterious. For instance, though there is ample experimental evidence that semantic content can be successfully conveyed from one person to another in utterances marred by grave syntactic errors, the mechanisms that make this possible

have not attracted the attention of theorists. Thus, the topic of this thesis - learning robust transduction rules from examples - falls into a theoretically uncharted area of research.

Nevertheless, the thesis would be incomplete if it did not try to place Keyword Classification Trees into a formal context. This chapter discusses related work on automatic learning of rules for identifying patterns in strings. Section 2 discusses traditional work on grammatical inference, section 3 discusses a newer body of work concerned with "probably approximately correct" learning, and section 4 discusses KCTs in the light of the preceding sections.

4.2 Grammatical Inference in the Traditional Paradigm

This section summarizes the large body of work on grammatical inference that predates or is unaffected by the theory of "probably approximately correct" (PAC) learning. The main focus will be on the inference of regular grammars, both non-stochastic and stochastic. Non-stochastic regular grammars are either deterministic or else non-deterministic but without numerical probability values; these two types belong together because of the equivalence of deterministic and non-deterministic finite automata (DFAs and NFAs) [Hop79]. Stochastic grammars contain productions that are associated with numerical probability values.

4.2.1 Inference of Regular Non-Stochastic Grammars

Limitations on Inference

Two results proved by Gold (in 1967 and 1978) provide the framework for all recent work in inference of non-stochastic regular grammars. The first proof shows that there are regular languages that cannot be learned "in the limit" from positive samples only. A language L is said to be learned by an algorithm A in the limit from positive samples when, after seeing a finite number of strings from L, A has learned a set of rules which can always decide correctly whether a given string is or is not in L. Since the regular languages are a subset of the context-free languages, which are a subset of the context-

sensitive languages, Gold's 1967 result implies that arbitrary context-free and context-sensitive languages also cannot be learned in the limit from positive samples.

What about inference from positive and negative samples? Many different grammars may generate the same language; furthermore, a finite sample does not uniquely define a language [Fu86a, pp. 344-345]. Thus, unless there are constraints on the form of the regular grammar to be inferred from positive and negative samples, there will often be an infinite number of answers, each of which generates the positive sample and not the negative sample. For instance, one can easily construct in polynomial time a DFA, called the *canonical definite finite-state grammar*, that generates exactly the strings in the positive sample and no other strings whatsoever [*ibid*, pg. 346]. However, this grammar does not satisfy our intuitive notion of "inference": it is too large (the DFA has a number of states proportional to the total number of symbols in the positive sample), and does not generalize beyond the strings actually seen. Most researchers have been interested in finding the DFA of minimum size that accepts the strings in the positive sample and rejects those in the negative sample.

Gold's second proof shows that when both positive and negative finite samples are provided, finding the DFA with the minimum number of states compatible with these data is NP-hard. That is, barring revolutionary developments in complexity theory, we can assume that for practical purposes solving this problem takes exponential time. Because of the equivalence of the set of strings accepted by DFAs and the regular languages, this result also applies to the regular, context-free and context-sensitive languages. D. Angluin states: "This result is generally interpreted as indicating that even a very simple case of inductive inference, inferring DFAs from positive and negative examples, is computationally intractable " [Ang87, pg. 89]. Angluin herself proved an analogous result, showing that the problem of finding the shortest regular expression compatible with a set of positive and negative samples is also NP-hard [Ang78].

Gold's second proof leaves a loophole: perhaps finding the minimum DFA compatible with positive and negative data is an excessively strict requirement, and if we let the DFA be somewhat larger than the minimum size, a polynomial-time algorithm can be found. It might be that by operating on the canonical definite finite-state DFA or in some other way, we can obtain a DFA only a bit bigger than the minimal one. Li and Vazirani [Li88] recently

closed this loophole by proving that even if the inferred DFA is allowed to be bigger than the minimum DFA by a factor $(1 + c)$, for small c, the problem is still NP-complete. Subsequently Pitt and Warmuth [Pit88b] showed that a DFA of size n cannot even be learned in polynomial time by a DFA of size polynomial in n unless NP = RP. Gold's second result is thus waterproof: for all practical purposes, it is impossible to infer reasonable regular languages from the presentation of positive and negative samples.

As Garcia and Vidal point out [Garc90], many of the papers published on grammatical inference pass over these fundamental limitations in silence. Most of them adopt one of two strategies to circumvent the Gold limitations on the inference of non-stochastic regular grammars:

1. The class of languages inferred is some tractable subset of the regular languages;

2. The information provided to the inference algorithm is more generous than described above - for instance, a "teacher" may answer questions posed by the algorithm, and even supply counterexamples to conjectures produced by it.

The next two subsections deal with these two strategies.

Inference of Grammars for Subsets of the Regular Languages

Biermann and Feldmann's k-tails method is a heuristic that requires users to supply a parameter k and positive samples from the language to be inferred [Bie72]. The method begins with a large, often non-deterministic, grammar called the *canonical derivative grammar* obtained from the canonical definite finite-state grammar. States in the canonical derivative grammar that have the same behaviour on strings of length k or less are merged. Thus, if the user picks a low k, such as 1 or 2, the resulting finite automaton will tend to be significantly smaller than the canonical derivative grammar, but may not model the language very well; a high k will often model the language better, but may be almost as large as the canonical derivative grammar. If negative samples are supplied the method can be adapted to generate a variety of finite automata from the positive samples, beginning with the canonical derivative grammar and decreasing in size - the smallest grammar compatible with the negative samples is chosen. This heuristic infers a family of non-deterministic

grammars which depend on the parameter k, and which correspond to finite automata somewhat smaller than the large automaton that generates only the strings in the positive sample. In most cases, the inferred automaton will still be much bigger than the minimal automaton compatible with the data.

Angluin and Smith describe the k-tails method as heuristic because the subset of the regular languages it infers is hard to characterize formally [Ang83]. Angluin has developed a method for learning the pattern languages, a well-defined subset of the regular languages [*ibid*, pg. 261]. A *pattern* is made up of variables and constants - e.g., $3xxy112x$, where x, y, z are variables. The language $L(p)$ generated by a pattern p is the set of strings obtained by substituting constant strings for the variables; for instance, $L(3xxy112x)$ contains 30000511200 and 3114241121. Angluin's inference algorithm finds the smallest finite automaton that generates a pattern language compatible with a given set of positive samples. For the case where the pattern contains only one variable, her algorithm runs in polynomial time.

Shinohara has extended Angluin's work to find polynomial-time algorithms for inferring from positive samples the minimum finite automaton that generates a given *extended regular pattern language*, or a given *non-cross-pattern language* [Shi82]. The extended regular pattern class languages are pattern languages in which all the variables occurring in a pattern are distinct and the null string may be substituted for any variable; the non-cross-pattern languages are pattern languages in which occurrences of a given variable may not be interrupted by occurrences of any other variable. Shinohara's algorithms begin with a general pattern which is then specialized; they have been applied to a practical data entry task.

Another tractable subset of the regular languages is the k-reversible languages [Ang82]. Let a, b, v, w, z be strings. A regular language L is k-reversible iff whenever avw and bvw are in L and v is of length k, then for every z, either both avz and bvz are in L, or neither is. Thus, a zero-reversible language L_Z is such that whenever aw and bw are in L_Z, for every z either az and bz are in L_Z, or neither is. An example of a zero-reversible language is the set of strings with an even number of 0s and an even number of 1.

Angluin has devised an algorithm that infers the smallest k-reversible language containing a positive sample, for fixed k [*ibid*]. Like the k-tails method, the algorithm constructs a large finite automaton from the strings

in the positive sample, then merges states satisfying a criterion of similarity that depends on k. The algorithm runs in $O(kn^3)$ time, where n is the total length of all the strings in the positive sample. If positive and negative samples are given, the algorithm can be run on the positive samples with $k = 0, 1, 2, ...$ until a k-reversible language is found that does not generate any of the strings in the negative sample. This modified algorithm is also polynomial-time: it is $O(m^2n^3)$, where m is the final value of k and n is the total length of the strings in the positive and negative samples. Angluin has proved that $m \leq n$, so the algorithm is $O(n^5)$ [*ibid*, pg. 762]. As required by the Gold limitations on inference of regular languages, the k-reversible languages are a proper subset of the regular languages: for instance, the regular expression $ba^*c + d(aa)^*c$ corresponds to a language that is not k-reversible for any k [*ibid*, pg. 750].

Another line of research arose from a paper published by Richetin and Vernadat, describing the "successor method" for grammatical inference from positive samples [Ric84]. The underlying inference strategy was similar to that employed in the well-known bigram language models employed in speech recognition systems: it relied on analysis of pairwise successions of terminals in the positive sample strings. Subsequently, Garcia, Vidal and Casacuberta showed that the class of grammars inferred by the successor method is the *local languages*, a proper subset of the regular languages [Garc87]. The same paper generalizes the successor method by showing that many non-local regular languages can be obtained from positive samples by applying "morphic operators" to a local language inferred by the successor method. This method supplements the information contained in the positive sample by information contained in the user-specified morphic operators, and thus belongs in the next subsection; in any case, it is too cumbersome to be practical.

A later paper by Garcia and Vidal generalizes the successor method in a different, more practical way [Garc90]. The paper presents a polynomial-time algorithm for inferring "k-testable languages in the strict sense", k-TLSSs. These languages are defined as follows: they begin with a set of initial segments I_k (of length at most k), they end with a set of final segments F_k (of length at most k), and they are forbidden to contain segments in the set T_k (of length exactly k). The set of 2-TLSS languages is exactly equal to the local languages. As k increases, the languages inferred from a given positive sample become increasingly restricted; if k is set to the length of the longest string in the positive sample, the inferred language is identical to

the positive sample. The time required to run the algorithm depends partly on m, the number of permissible substrings of length k (i.e. the number of length k substrings not in T_k): it is $O(kn \log m)$. Furthermore, the number of transitions in the inferred automaton is bounded by $O(m)$.

The k-TLSSs are a subset of Angluin's k-reversible languages, so her algorithm could be used to perform the inference instead. However, if one is sure that a language is best modeled as a k-TLSS, the Garcia-Vidal algorithm is usually faster. Like Angluin, Garcia and Vidal extend their method of inferring a grammar from positive samples to the situation where both positive and negative samples are available by increasing k from a low initial value ($k = 2$) until the grammar generated from the positive samples does not generate any of the strings in the negative sample. I will describe a stochastic extension of the Garcia-Vidal algorithm in the section of this chapter dealing with stochastic grammars.

Inference of Regular Grammars with Additional Information

This section describes methods where the learning protocol allows the inference algorithm to ask a "teacher" for additional information beyond that contained in positive and negative string samples. Here, the question of interest is: how little information must be given by the teacher for the correct regular language to be inferred?

An early paper by B. and K. Knobe describes an algorithm that infers context-free grammars from positive samples [Kno76]. At every stage, the algorithm has a partial grammar made up of production rules that accounts for the strings seen so far. The teacher will tell it whether it is done, or whether the partial grammar generates ungrammatical strings, or tells it the current grammar is incomplete and gives it a new string from the desired language which the inference algorithm can use to generate additional rules for the inferred grammar. The inference algorithm always tries to create new production rules that are as general as possible: they are short, they have a high ratio of nonterminals to terminals, and they will generate recursions.

The Knobes did not attempt to characterize formally the set of context-free grammars that can be inferred by using their approach. It seems probable that this set is a proper subset of the context-free languages, and perhaps does not include the set of regular languages but merely overlaps with it. They did not give a time complexity analysis for the method. Furthermore, the

method depends on the teacher presenting positive samples in appropriate order - it is easy to devise malign orderings of the positive samples which will result in the method never inferring the desired language. This method has received no further attention from researchers. Interestingly, its underlying philosophy is the same as that underlying KCTs: "We think that the principle of working from the most general to the more specific is a principle applicable to any inference problem, and it is definitely the prime reason for the success of our entire approach" [*ibid*, pp. 131-132].

Crespi-Reghizzi *et al* have developed a method for inferring context-free grammars from positive bracketed samples [Cres73, Cres72]. Given a context-free grammar, one can generate a bracketed context-free grammar from it by replacing each production of the form $N \Rightarrow u$, where N stands for a non-terminal and u for a string, by a production of the form $N \Rightarrow [u]$. The strings generated by this grammar will be bracketed in a way that indicates the order of operations. The Crespi-Reghizzi algorithm takes as input a positive sample consisting of a set of bracketed strings of this form, and merges states that satisfy a similarity criterion in a manner reminiscent of the methods described in the previous section. The algorithm generates the smallest, simplest context-free grammar of a certain type, called the "free operator precedence grammars", compatible with the positive sample. I am unaware of any time complexity results for the algorithm, but in practice it seems to run in polynomial time. It requires a substantial amount of work on the part of the teacher, who must supply fully-parsed strings belonging to the language to be inferred.

A recent paper by Angluin contains a proof that any regular language R can be learned in polynomial time by a learning algorithm L from a "minimally adequate teacher" T [Ang87]. A minimally adequate teacher T is one who:

1. answers membership queries: given a string by L, T says whether or not the string is in R;

2. answers conjectures: given a description of a hypothetical regular language H by L (in the form of an encoding of a deterministic finite-state automaton), T either tells L that it has correctly guessed R (H and R are equivalent) or provides a counterexample (a string in $R - H$ or in $H - R$).

Alguin's algorithm yields the minimum DFA generating R in time polynomial in the number of states in that DFA, and in the length of the longest counterexample provided by T.

The requirement that T supply counterexamples may prevent practical use of the algorithm. True, if T itself "knows" the regular language R to be learned by L in the form of a deterministic finite-state automaton, it can provably produce a string in $R - H$ or in $H - R$ in polynomial time [*ibid*, pg. 89] - but if one already owns a finite automaton T that knows the regular language, why use the inference algorithm at all? However, Angluin also shows that a version of the algorithm can carry out "probably approximately correct" learning under very reasonable assumptions, as will be discussed in section 3.2.

4.2.2 Stochastic Grammars

Inference of Stochastic Grammars

A stochastic grammar is one in which every production rule R_i is associated with a probability p_i, and which therefore assigns to every possible string a probability of being produced (perhaps 0). We will only be interested in stochastic grammars for which the total probability over all strings is 1; these are called the stochastic consistent grammars. A stochastic presentation from a stochastic consistent grammar G is an infinite sequence of strings generated according to the probabilistic production rules of G. In 1969, J. Horning showed that stochastic context-free languages can be inferred in the limit from a stochastic presentation with probability 1 [Ang83 pp. 250-1]. Gold showed that the inference of non-stochastic regular grammars requires a negative sample. This is untrue for stochastic regular or context-free grammars that are given a stochastic presentation, because the strings that never appear can be assigned a probability of 0 (with a degree of error that depends on how many strings in the presentation are actually used by the inference algorithm).

Practical algorithms for learning stochastic grammars access a sample made up of a finite set of strings $E = \{(x_1, c_1), (x_2, c_2), ..., (x_n, c_n)\}$, where x_i stands for a string and c_i the number of times it occurs in the sample. They tend to fall into two classes, "search" algorithms and "constructive" algorithms. The latter are called "jumping to conclusions" algorithms by

Angluin and Smith [Ang83, pg. 260]. Search algorithms generate all grammars of a particular class in a systematic manner until a grammar is found that meets an acceptance criterion; they are cautious, being designed not to skip over any acceptable hypothesis, and tend to be time-consuming. Constructive algorithms resemble the algorithms discussed in the last section. They begin with a hypothetical grammar generated directly from the sample E, which is then successively refined to produce new grammars until an acceptance criterion is met; they tend to run faster than the search algorithms, but often produce more complicated, less elegant solutions. Search algorithms usually start with very simple, incorrect grammars that are modified in the direction of greater complexity and greater conformity with the sample E, while constructive algorithms begin with a complicated grammar that generates only the strings in E and is then modified in the direction of greater simplicity.

Horning's 1969 Ph.D. dissertation provided some of the necessary machinery for the search approach, by showing how to construct a Bayesian acceptance criterion for hypothesized stochastic grammars (see [Fu86b, pg. 367]). Suppose we have a finite set of such hypothesized grammars, $\{G_1, G_2, ..., G_r\}$, each with an *a priori* probability $p(G_i)$. Horning derived a formula for calculating from the productions of a grammar G_i the probability $p(E|G_i)$ that G_i will generate the sample E. The optimal G_i is the one for which the Bayesian criterion $p(G_i)p(E|G_i)$, and thus $p(G_i|E)$, is maximized. Horning also devised a stochastic generator of grammars to provide the hypothesized grammars G_i and their associated probabilities.

The main disadvantage of Horning's approach is its computational complexity: the algorithm takes a long time to calculate $p(E|G_i)$, and must look at all the G_is before it can pick the optimal one [Fu86b, pg. 367]. To speed up search algorithms, Maryanski and Booth devised a χ^2 test to measure the goodness of fit of a hypothesized stochastic grammar to the sample E [Mar77]. For each sample string x_j occurring c_j times in E, one can calculate the expected number of occurrences of x_j according to grammar G_i. The χ^2 test compares these expected numbers with the actually observed counts c_j, telling us whether the deviation is about what we would expect, or if G_i should be rejected.

Another piece of machinery is needed before a practical search algorithm can be designed: a method for converting a non-stochastic grammar into a stochastic one, by estimating the probabilities of the production rules from

the sample E. This is necessary because search algorithms usually explore a search space made up of non-stochastic grammars. Once a non-stochastic grammar H_i is reached, it is converted to a stochastic grammar G_i and then tested against the acceptance criterion. A method for estimating the probabilities from E is given in [Fu86b, pp. 364-365]. It only works for unambiguous grammars, in which every possible string can only be generated in one way from the productions.

Maryanski and Booth have implemented a search algorithm that asks the user for a threshold for the χ^2 acceptance criterion [Mar77]. The algorithm searches through an infinite tree of deterministic regular grammars in which the first level corresponds to grammars or partial grammars with one nonterminal symbol, the second level to grammars or partial grammars with two nonterminals, and so on. Each of the candidate grammars examined is converted to stochastic form by the probability estimation algorithm, then evaluated by the χ^2 test. Three outcomes are possible: the grammar may be accepted, it may be completely rejected (along with all its unborn children), or it may be selected for expansion. In the last case, the grammar generates several deterministic regular grammars at the next level of the tree. Heuristic pruning techniques were developed to reduce the number of nodes that must be considered. The search algorithm can be implemented in either depth-first or breadth-first fashion.

Maryanski and Booth have shown that this method will always generate a finite automaton with the minimum number of states that meets the user-supplied χ^2 goodness of fit criterion. Gaines employs a similar algorithm, but prefers to consider goodness of fit and the number of states as separate criteria that should be considered independently [Gai78]. These researchers have not proved that their algorithms run in polynomial time - given the exhaustive nature of the search they perform, they are probably not polynomial-time.

Cook *et al* devised a constructive algorithm that carries out some local search [Coo76]. Like other constructive algorithms, it begins with a complex grammar that is derived from the observed sample, and modifies it in the direction of greater simplicity. In this case, the starting grammar is a stochastic grammar that generates each string in the sample with precisely its observed frequency. A cost measure $M(G, E)$ that is a linear combination of the complexity of the grammar G and its discrepancy from the sample E is defined; the measure assigns lower cost to productions generating homogeneous strings than to strings made up of mixtures of terminals. Several

grammar transformations - substitution, disjunction, and removal of productions - are also defined. These transformations are used to generate new grammars. The algorithm then begins work on the lowest-cost new grammar, and iterates until no further improvement is possible. The algorithm of Cook *et al* is greedy: it finds the locally optimal, lowest-cost stochastic grammar in the neighbourhood of the starting grammar. No complexity results were presented for the algorithm.

The approach of Van der Mude and Walker is similar to that of Cook *et al* [Van78]. The main difference is the use of a quasi-Bayesian acceptance criterion $P(E|G)P(G)$, as suggested by Horning; the criterion is only quasi-Bayesian because $P(G)$ is a heuristic, rather than a genuine probability estimate. The algorithm first derives a stochastic regular grammar with many rules that generates exactly the strings in E, with expected frequencies exactly equal to the observed frequencies. This grammar is called the *tree grammar* G_T and is easily constructed in polynomial time. A set of transformations called "splits" is applied to G_T to produce a new set of candidate grammars which are then evaluated in terms of the quasi-Bayesian criterion. This procedure is iterated to find the optimal grammar in the local neighbourhood of G_T. The time complexity is not given, but the way in which time goes up with size of sample in the trial runs listed in the paper suggests it is exponential.

A previous section discussed Garcia and Vidal's method for learning k-TLSSs from a positive sample. The stochastic extension of this method by the same researchers is a good example of a pure constructive approach [Garc90]. To adapt the method to learning stochastic k-TLSSs, Garcia and Vidal use the k-TLSS method to infer a non-stochastic grammar, then obtain a maximum likelihood estimate of the production probabilities by means of the estimation procedure in [Fu86b, pp. 364-365]. This procedure only works if the inferred grammar is unambiguous.

Since Garcia and Vidal are interested in seeing how well stochastic k-TLSSs can approximate the regular languages, they carried out a series of experiments in which each of ten stochastic regular grammars generated a training set of strings; a test set was generated by pooling other strings generated by the ten grammars [Garc90]. A stochastic k-TLSS grammar was inferred from each of the training sets, so that ten stochastic k-TLSS grammars were generated. To allow for the parsing of strings almost but not quite in a given grammar, a wild-card production with low probability

was added to match otherwise unparsable input symbols. The strings in the test set were then parsed by each of the ten stochastic grammars, with each string assigned to the grammar which assigned it the highest probability of being produced.

Garcia and Vidal examined the effects of varying the size of the training sets and the value of k on the proportion of strings assigned to the correct grammar. When each training set contained 150 strings and k was greater than 2, the error rate was never more than 0.6%. These results show that a language at one level of the formal hierarchy may be successfully approximated for classification purposes by a stochastic language at a lower level. The experiment also illustrates one of the least attractive aspects of the grammatical inference approach to classification: ten different grammars must be grown to perform a ten-fold classification. A single KCT grown on the same data could perform the same ten-fold classification, perhaps equally well.

Calculation of Probabilities in a Stochastic Grammar

The estimation of probabilities for a stochastic grammar whose structure is known is an easier problem than the inference of the grammar from nothing, but is by no means trivial. As mentioned above, Fu gives a method for estimating production probabilities in unambiguous context-free grammars [Fu86b, pp. 364-365]. The estimation techniques for stochastic grammars currently in the most widespread use are those that estimate the parameters of Hidden Markov Models (HMMs), which are described in Chapter 2. These techniques, such as the Baum-Welch method, iteratively search for probability values in an HMM that maximize the probability of the observations, given the HMM [RabWL]. They do not analytically solve this estimation problem, but find values that yield a local maximum.

Though HMMs are finite automata, J. Baker extended an HMM parameter estimation method to stochastic context-free grammars, calling this extension the "Inside-Outside" algorithm [Bak79]. Recently, SCFGs and the Inside-Outside algorithm have been applied to natural language [Lar91, Lar90]; unfortunately, serious difficulties have been encountered. The amount of time required to estimate the SCFG parameters on a reasonable amount of training data is large. Worse, the algorithm often gets trapped in a local maximum that is extremely sub-optimal, unlike the HMM case where the local maximum found is usually quite good.

The approach of Pieraccini *et al* at AT&T, which was described in Chapter 3, is somewhat less ambitious. Recall that these researchers employ HMM-like estimation techniques to obtain parameter values for a finite-state model of natural language. In this respect, their approach resembles the KCT approach: both approaches assume that for practical purposes, natural language can be approximated by a finite-state model.

4.3 PAC Learning and P-Concepts

4.3.1 Introduction

Most of the techniques for inferring stochastic grammars presented in the previous section had a heuristic flavour. Their inventors neither prove that they were capable of learning a clearly-defined set of languages, nor provide performance guarantees for them, in the sense of proving that the learning process requires only a reasonable amount of time. Clearly, precise criteria for evaluating probabilistic learning are needed.

A 1984 article by L. Valiant has acted as a catalyst for recent research into computational learning theory [Val84]. M. Kearns summarizes the features of Valiant's new paradigm as follows [Kea90a]:

1. The requirement for the learning algorithm to identify the target rules exactly is relaxed to allow approximations. The paradigm defines an acceptable approximation to the target rules, and this definition is phrased in terms of probabilities.

2. The demand for computational efficiency is now central - we are interested in rules that can be learned in a reasonable amount of time (which implies a limited sample size), rather than in rules that can only be learned in the limit.

3. The learning algorithm should work for any probability distribution, provided that the samples on which it learns and the domain in which the learned rules will be applied are governed by the same distribution. We will call an algorithm that meets this criterion "distribution-free". Traditional techniques in statistical pattern recognition often assume a particular input distribution (see [Dud73]).

As Kearns points out, the relationship between formal and empirical machine learning research has often been less close than one might expect. "Many of the problems tackled by artificial intelligence ... appear extremely complex and are poorly understood in their biological incarnations, to the point that they are currently beyond mathematical formalization. The research presented here does not pretend to address such problems" [Kea90a, pg. 3]. Nonetheless, since the new paradigm gives a model for machine learning that is both more practical and more precisely specified in formal terms than older models, it constitutes a major step towards the application of formal learning theory to problems in artificial intelligence.

4.3.2 PAC Learning

Valiant begins his definition of PAC learning as follows [Val84]. Consider a domain X made up of descriptions of certain objects. For instance, X might be a list of descriptions of all the objects in a house. The goal of a learning algorithm A is to learn a subset of X; the subset to be learned is called a *concept*. Thus, we might present A with some examples of descriptions of chairs and of non-chairs in X, hoping that it will learn rules capable of deciding whether an arbitrary element of X belongs to the subset "chairs" in X. A must have a formalism for encoding the rules it learns. The set of concepts capable of being expressed in this formalism is the *representation class*. A representation class C need not be capable of describing all the possible concepts (subsets) of X, only those that are of interest. Given the representation class C, however, we want A to be able to learn any specific member c of C.

Each time A is executed, its goal is to learn some concept c in C. The next time it is executed, the concept to be learned may be different, but will still be in C. A has access to two oracles, **POS** and **NEG**, which yield descriptions of sample objects in X; the descriptions provided by **POS** are of items in c, those provided by **NEG** are of items not in c. We will assume that the descriptions provided by **POS** and **NEG** are efficiently encoded; furthermore, they are drawn randomly from probability distributions $D^+(c)$ and $D^-(c)$ respectively. These two probability distributions are arbitrary but fixed: they do not vary over time. As well as being the distributions that provide the examples from which A learns, D^+ and D^- are also the target distributions. That is, we will be satisfied with the rules learned by A if they

work well on data drawn from D^+ and D^-; we do not require A to learn rules that also work well on other distributions.

The learning performed by A may be approximate: A may output a hypothesized concept h that is only an approximation to c, provided that h satifies certain criteria. h, like c, is always a subset of X. We will assume that the description of h output by A is polynomially evaluatable. That is, given h and a data point x, it can be decided in time polynomial in the length of the descriptions of h and x whether x is in h or not. h may be drawn from a representation class H that is different from C, the representation class for c.

Finally, consider the two ways in which h may wrongly classify a new data item i: it may wrongly exclude i even though it is a member of c, or it may wrongly include i even though it is not in c. Let $e^+(h)$ be the probability that an item randomly chosen from D^+ is not in h, i.e. is wrongly classified by h as not being an example of concept c, and let $e^-(h)$ be the probability that an item randomly chosen from D^- is wrongly "accepted" by h. We can now describe *probably approximately correct learning* [Kea90a, pg. 11].

Definition:

- Consider two representation classes C and H over X. An algorithm A with access to oracles **POS**, **NEG** is a learning algorithm for C if, for any c in C and for any values of δ, ϵ between 0 and 1, A outputs a representation h in H such that with probability $1 - \delta$, $e^+(h) < \epsilon$ and $e^-(h) < \epsilon$. C is called the "target class" and H the "hypothesis class". A is a "polynomial learning algorithm" for C if C and H are polynomially evaluatable and A runs in time polynomial in $1/\delta$, $1/\epsilon$, and $|c|$ - we say that A carries out "probably approximately correct learning", or "PAC learning".

A simple example will illustrate this (taken from [Kea90a, pp. 14-17]). Let the formalism for both C and H be conjunctions of Boolean variables (also called "monomials"), where the set of variables describes various properties of animals. Some variables describe physical properties: *is_large*, *has_claws*, *has_mane*, *has_four_legs*, *has_wings*, and so on. Some variables describe behaviour: *can_fly*, *can_walk_on_two_legs*, *can_speak*, *hibernates*, and so on. Some variables describe the animal's habitat: *is_wild*, *lives_in_circus*, and so on. Other variables deal with scientific classifications: *is_mammal* and so on. There may be many other types of variables.

Now, suppose that the concept c to be learned is "lion". c might be the following monomial:

$c = is\_mammal\ and\ is\_large\ and\ has\_claws\ and\ (not\ hibernates)$.

As long as c successfully identifies the subset of lions within X, it doesn't matter whether it is a valid description of lions in general. Consider D^+, the probability distribution for positive examples of c. D^+ yields descriptions of particular lions, each of which specifies the values of all variables. By definition, each of these descriptions must have the variables *is_mammal*, *is_large*, and *has_claws* set to $TRUE$ and *hibernates* set to $FALSE$ with probability 1. However, the variable *has_mane* may be $TRUE$ about half the time (when the lion is male) and $FALSE$ the rest of the time; *has_wings* will always be false. There may be dependencies between some of the variables: for instance, *can_walk_on_two_legs* may have low average probability of being $TRUE$, but much higher probability of being $TRUE$ when *lives_in_circus* is $TRUE$. Similarly, D^- is the probability distribution for examples of non-lions in X, and may have arbitrary dependencies between variables.

We require a learning algorithm A for finding a monomial h that is good at distinguishing lions from non-lions. A will learn h from examples generated from D^+ and D^-. Although C and H are identical in this example, since both the concept c and the hypothesis h are monomials, h need not be the same as c. For instance, suppose the domain X only includes two species of large animals: lions and elephants. In that case, A might output the following h: *has_claws*. This h would perform well, correctly identifying examples from D^+ as lions and examples from D^- as non-lions, even though it is different from c.

Valiant devised a polynomial learning algorithm A that learns monomials over n variables [Val84 article]. The algorithm only needs examples drawn from **POS**, not **NEG**, so it is called "positive-only". Though it only uses examples from **POS**, the definition of PAC learning given above requires it to classify correctly most new examples generated by both **POS** and **NEG**, which it does. The algorithm is subject to two types of error, one associated with ϵ, one with δ. The ϵ type of error occurs when a variable with low but non-zero probability of being false in D^+ is not deleted from h. This type of error, by definition, only misclassifies new examples that have low probability in D^+. The δ type of error is more serious. It occurs when the learning examples drawn from D^+ are very unrepresentative of D^+, so that the rate of misclassification on new examples is higher than ϵ. For instance,

if the m positive examples drawn during learning were all trained circus lions (even though D^+ assigns a moderately low probability to this type of lion), the final version of h might include the variable *can_walk_on_two_legs*. This h would perform poorly on new data.

In addition to defining PAC learning and giving a polynomial algorithm for learning monomials, Valiant's original article extended the algorithm to two other classes of Boolean formulae [*ibid*]. Define a "k-clause" as a disjunction of at most k Boolean variables, e.g. $C_i = v_1$ *or* ... *or* v_n. Then the representation class kCNF consists of conjunctions of k-clauses, e.g. C_2 *and* C_5 *and* C_6. Similarly, define a "k-term" as a conjunction of at most k Boolean variables, e.g. $T_i = v_1$ *and* ... *and* v_n.. The representation class kDNF consists of disjunctions of k-terms, e.g. T_1 *or* T_4. Valiant adapted the algorithm for learning monomials to obtain polynomial algorithms for learning kCNFs and kDNFs. For each of these algorithms, the concept class C is equal to the hypothesis class H.

After the publication of Valiant's 1984 paper, D. Angluin studied a problem closely related to PAC-learning a regular set from examples [Ang87]. Recall from section 2.1.3 that in the same paper, Angluin showed that a learning algorithm L exists for learning any regular language R in polynomial time from a minimally adequate teacher T. Such a teacher answers membership queries and answers conjectures. That is, L can give T strings which T must declare as members or non-members of R, and T must respond to a hypothetical regular grammar H output by L by either confirming that H is identical to R or by providing a counterexample - a string in the symmetric difference of R and H.

Angluin also considered the case where L has access to an oracle supplying sample strings randomly chosen from an unknown distribution D, each labeled as belonging to R or not. In this case, T need only carry out the first part of its job: answering membership queries. The oracle can be used as a source of counterexamples for modifying L's current hypothesis H - some of the strings from the oracle belong to R but not H, and vice versa.

Angluin defines a hypothesis H that ϵ-approximates R as follows: H ϵ-approximates R if D assigns probability less than ϵ to strings in the symmetric difference of R and H. Let n be the number of states in the minimum DFA for R, and m the maximum length of a string output by the oracle. Angluin showed that in time polynomial in $1/\epsilon$, $1/\delta$, n and m, her learning algorithm L yields with probability $1-\delta$ a hypothesis H that ϵ-approxim[illegible]

R. Note that although Angluin's algorithm *almost* meets the PAC-learning criteria, it still requires a teacher T to answer membership queries. Thus, it is not a PAC-learning algorithm.

Subsequently, Pitt and Valiant studied two classes, k-term-DNF and k-clause-CNF that are properly contained within kCNF and kDNF, respectively [Pit88a]. They showed that learning a concept c in k-term-DNF by h in k-term-DNF is NP-hard for $k > 1$, yet learning the same c by h in kCNF can be done by the Valiant algorithm for kCNF. Similarly, learning a concept in k-clause-CNF by a hypothesis in the same class is NP-hard, yet can be done in polynomial time if the hypothesis class is kDNF. Thus, there are cases where the learning problem is too hard if we impose the constraint $C = H$, but tractable if we let H be a more powerful class than C.

From this, it follows that there are two types of hardness result: "representation-based" and "representation-independent". Both concern learning of a fixed representation class C of concepts. A representation-independent hardness result would prove that C is hard to learn by *any* polynomially evaluatable representation class H of hypotheses, while a representation-based hardness result merely shows that C is hard to learn for *some* particular H. Clearly, Pitt and Valiant's proof that it is hard to learn k-term-DNF by k-term-DNF was representation-based.

Representation-independent hardness results are much more powerful than representation-based ones, and correspondingly harder to obtain. Kearns gives some interesting representation-independent hardness results based on cryptographic assumptions [Kea90a, Chap. 7]. He shows that given these assumptions, general Boolean formulae over n variables of length polynomial in n cannot be learned in polynomial time.

This concludes the introduction to PAC learning. In previous sections, we have been concerned with the learning of regular languages, i.e. of finite automata. Gold showed that learning a DFA with the minimal number of states from positive and negative samples was NP-hard (see section 2.1.1 above). However, PAC-learnability provides a much more tolerant criterion for learning than was considered previously. Is polynomial-time PAC-learning of finite automata possible?

In 1989, Pitt and Warmuth proved that it is NP-hard to PAC-learn a DFA of size n by an NFA whose size is bounded above by any polynomial in n [Pit89]. This was a representation-based hardness result which still left open the possibility that DFAs can be PAC-learned by some representation

class H other than regular sets. Next, Pitt and Warmuth proved that the existence of a polynomial-time algorithm for PAC-learning DFAs by *any* representation class H implies the existence of a polynomial-time algorithm for PAC-learning arbitrary Boolean formulae [Pit90]. However, as described above, Kearns showed that given reasonable cryptographic assumptions, one cannot PAC-learn Boolean formulae in polynomial time. This constitutes a representation-independent proof that DFAs cannot be PAC-learned in polynomial time.

The next section considers another paradigm for probabilistic learning closely related to PAC learning, but even more tolerant of error. This paradigm will be used subsequently to prove a result about KCTs.

4.3.3 Learning P-Concepts

PAC learning involves a tolerant and realistic definition of acceptability for the hypothesis yielded by a learning algorithm. However, the PAC-learning assumptions about the nature of the examples are open to question. Is it realistic to assume that one can obtain sources **POS** and **NEG** of positive and negative examples that are 100% reliable?

If one relaxes this assumption, one obtains new models for learning. For instance, we may assume a "white noise" model in which we still use the oracles **POS** and **NEG**, except that the examples yielded by each have a small, fixed probability of being mislabelled before they reach the learning algorithm [Ang88]. That is, if examples from **POS** are normally labelled "+" and those from **NEG** normally labelled "-", a certain proportion chosen at random will get the wrong label. In chapter 5 of his book, M. Kearns describes a more pessimistic model in which a malicious adversary is allowed to choose and mislabel a proportion E_{MAL} of the examples, and does this in the way most likely to cause the learning algorithm to output a bad hypothesis [Kea90a].

These two models still assume that in principle, a particular example belongs to exactly one of the two categories. However, we may wish to explore domains where classification is uncertain or probabilistic in principle - where there is not necessarily a single right classification. The theory of learning "probabilistic concepts" or "p-concepts" deals with this kind of situation, and is being developed mainly by M. Kearns and R. Schapire. This theory is very new; the only references I know of are Schapire's Ph.D. thesis [Scha91] and

an extended abstract by Kearns and Schapire [Kea90b], though M. Kearns kindly sent me the full version of the latter.

Kearns and Schapire give three examples to clarify p-concepts:

1. A meteorologist measures a small number of relevant parameters, such as the temperature, barometric pressure, and wind speed and direction; he makes a prediction of the form "chances for rain tommorrow are 70%". The next day it either rains or does not rain.

2. A statistician wants to predict which students will or will not be admitted to a particular college on the basis of their high school average and SAT scores. Some students have such good or such bad marks that the statistician can make a prediction with almost complete certainty; the fate of borderline students depends on the mood of the admissions officer at a particular moment (about which the statistician has no information). Each student either is or is not admitted.

3. A physicist wants to know the orientation of spin for particles in a magnetic field of known strength and direction. Each particle has spin either up or down.

In each of these three cases, as with PAC-learning, the examples have an unambiguous label - each is either positive or negative. For instance, it either rains or does not rain on a particular day. The difference is that two identical examples may receive opposite labels, and this is not due to some error in the labelling process but part of the phenomenon itself. In some cases, as for examples 1 and 2, we may lack additional information that would enable us to correctly distinguish positive and negative examples 100% of the time. In other cases, as in example 3, quantum theory tells us that 100% accuracy of classification is impossible in principle. Both types of phenomenon are covered by the theory of p-concepts. The theory also deals with the "fuzzy logic" type of concept, where two different people may disagree about the boundaries of a concept like "tallness".

In all these situations, we have a domain X from which we draw an example x according to distribution D. Once drawn, x is labelled "+" with probability $c(x)$ and "-" with probability $1 - c(x)$. The learning algorithm A has access to labelled examples of the form $< x, + >$ or $< x, - >$, from which we want it to generate in polynomial time an approximation h to the

function $c(x)$. We will call both c and h p-concepts: that is, they are real-valued functions mapping X onto the interval $[0, 1]$. An obvious algorithm A would collect several examples for each possible value x, and obtain an estimate $h(x)$ as an average frequency $f(+|x)$. This will work if x only takes on a few discrete values. In most cases, however, we will seldom get *exactly* the same metereological measurements or the same combination of marks in a row. Instead, algorithms for obtaining an approximate $h(x)$ for a $c(x)$ usually rely on some kind of structure in $c(x)$ - for instance, days with similar measurements may have a similar probability of rain.

Definition:

- A p-concept h is an (ϵ, γ)-good model of probability for distribution $c(x)$ if $Pr[|h(x) - c(x)| > \gamma] \leq \epsilon$, where the probability is taken over distribution D. Then by analogy with PAC-learning, we say that A learns a class of p-concepts C if for any $c(x)$ in C, given access to an oracle that draws examples from D and then labels them according to $c(x)$, A outputs with probability $1 - \delta$ a p-concept h that is an (ϵ, γ)-good model for distribution $c(x)$. If A can produce such an h in time polynomial in $1/\epsilon$, $1/\gamma$, and $1/\delta$ we say that A is a polynomial-time learning algorithm for p-concepts.

Note that once we have learned a reasonably good approximation $h(x)$ to $c(x)$, we can easily turn it into a decision rule. The best decision rule is to label examples x for which $h(x) > 0.5$ with "+", those for which $h(x) < 0.5$ with "-", and the rest arbitrarily (unless one type of misclassification is more costly than another).

Kearns and Schapire have devised polynomial-time learning algorithms for a number of p-concept classes C. For our purposes, the most important of these is the algorithm that learns probabilistic decision lists. The next section shows that the existence of this algorithm casts light on what is learned by KCTs.

Suppose we have a set of n Boolean parameters, each of which is either $TRUE$ or $FALSE$. We can construct a set $f_1, f_2, ..., f_s$ of Boolean-valued functions over these parameters. If our domain is the admissions task described above, f_1 might be the question "does the student have straight A's?", f_2 might be the question "does the student have a SAT math score of more than 650, or an English SAT score of more than 700?", and so on. Now, we might have an admissions procedure which resulted in probabilities

of the following form: if f_1 is $TRUE$ the student is admitted with probability r_1; otherwise, if f_2 is $TRUE$ the student is admitted with probability r_2; otherwise... and so on until f_s, which is the constant function $TRUE$, gives a probability r_s for admission. This is a probabilistic decision list. A probabilistic decision list with decreasing probabilities is the special case where each $r_i \geq r_{i+1}$.

We will assume that we have already drawn up a list of the relevant Boolean-valued functions or questions $\{q_i\}$ from which the $\{f_i\}$ will be selected. The learning task is to find out what order they should be in in the decision list, and to estimate the probabilities r_i. First, we draw a large number m of examples, each labelled "+" or "-"; m is polynomial in $1/\epsilon, 1/\gamma, 1/\delta$. Now, consider a particular question q_i. Among the m examples, some will yield a value $TRUE$ to q_i; of these, some will be labelled "+" and some "-". Of those that yield the value $FALSE$ to q_i, some will be labelled "+" and some "-". We can calculate from these counts a conditional frequency $f(+|q_i)$ for each q_i.

Now, the function f_1 has the property that if it is $TRUE$, the probability of getting a "+" is maximal (it is equal to r_1). Therefore, the question q_i that yields the highest conditional frequency $f(+|q_i)$ is likely to be f_1; furthermore, if $f_1 = q_i$ then $f(+|q_i)$ is a good estimate for r_1. The learning algorithm sets f_1 to this q_i and uses the corresponding conditional frequency as an estimate of r_1. Then, it discards all the examples for which f_1 is $TRUE$ and calculates the conditional frequency $f(+|q_i)$ for the q_is other than the one just chosen as f_1. The q_i yielding the maximal conditional frequency is chosen as f_2, this conditional frequency is taken as the estimate for r_2, and so on until the algorithm runs out of questions. Kearns and Schapire prove that this learning algorithm is polynomial in $1/\epsilon$, $1/\gamma$, $1/\delta$, in the total length of the question list $\{q_i\}$, and in the maximum time required to evaluate any question q_i [Kea90b pg. 385, Scha91 pp. 117-122]. The proof is complex, and will not be given here.

4.4 Keyword Classification Trees

4.4.1 The Theory and Practice of Machine Learning

Ideally, this thesis would supply rigorous answers to two questions:

1. What do KCTs learn?

2. What computational resources do they require to carry out learning?

Chapter 7 answers the second question; the current state of machine learning theory makes it impossible to answer the first one fully.

As Kearns and others point out, there is a large gap between the theory and practice of machine learning. The complex learning algorithms actually used in AI do not lend themselves to formal analysis. Thus, theoreticians have tended to employ simplistic models that leave out important aspects of practical machine learning, and designers of AI algorithms have often abdicated their responsibility to analyze formally what they are doing. The admirable work of Valiant, Kearns, and their colleagues has provided us with new models of learning that are simultaneously more rigorous and more computationally realistic than earlier models. Perhaps one day, AI practitioners will routinely give and prove probabilistic performance guarantees for their learning algorithms. Meanwhile, research on PAC learning and p-concepts has narrowed the gap between theory and practice, but not bridged it.

This section provides some clues to the question of what KCTs learn. First, it is shown that each class in a single-symbol or set-membership KCT corresponds to a regular set. Next, p-concept theory is used to prove a result for a data structure closely resembling KCTs. Finally, the relationship of KCTs to the learning algorithms described earlier is discussed.

4.4.2 Each Class of a KCT is a Regular Set

We will prove by induction that every node in a single-symbol tree represents a regular language. A regular language is one describable by a regular expression. Hopcroft and Ullman define the regular expressions over an alphabet Σ and the sets they accept as follows [Hop79 pg. 28]:

1. ϕ is a regular expression and denotes the empty set.

2. ϵ is a regular expression and denotes the set $\{\epsilon\}$.

3. For each a in Σ, a is a regular expression and denotes the set $\{a\}$.

4. If r and s are regular expressions denoting the languages R and S, respectively, then $(r \cup s)$, (rs), and (r^*) are regular expressions that denote the sets $R \bigcup S$, RS, and R^*, respectively.

We will also need some results from Hopcroft and Ullman pertaining to the closure properties of regular sets [Hop79 pg. 59]:

1. The regular sets are closed under union $\bigcup$.
2. The regular sets are closed under complementation - that is, if L is a regular set and $L \subseteq \Sigma^*$, then $L^c = \Sigma^* - L$ is a regular set.
3. The regular sets are closed under intersection $\bigcap$.

We wish to show that every node in a single-symbol KCT accepts a regular set. Now, our algorithm for growing a single-symbol KCT chooses a question for every interior node in the KCT, generated from the "known structure" for the node. Questions are made up of the symbol + and the symbols in Σ. For instance, if the known structure for N were $< a + c+ >$, the chosen question might be "is the form of the string $< azc+ >$?"

Our first step is to show that KCT questions involve regular expressions, even though they may include the non-standard symbol +. The KCT-growing algorithms use the symbol + to denote all strings other than the empty string ϵ; + thus denotes the complement of the regular set $\{\epsilon\}$, and therefore (by closure property 3) denotes a regular set. Thus + by itself is a regular expression. By the definition of the regular expressions, any single symbol **a** in Σ is itself a regular expression, and any concatenation (rs) of two regular expressions is itself a regular expression. It follows by induction on the length of any string w that is made up of the symbol + and symbols in Σ that w is a regular expression. Thus, questions in single-symbol KCTs are regular expressions.

For a node N, let $S(N)$ denote the set of strings arriving at N; for a question Q involving a regular expression, let $S'(Q)$ denote the set of strings that would yield "YES" to the question (i.e. that match the regular expression in Q). Consider an interior node N, its chosen question Q, and its children YES and NO. YES accepts strings in $S(N) \bigcap S'(Q)$; NO accepts strings in $S(N) \bigcap [S'(Q)]^c$. Since a regular set is accepted at the root, and the sets accepted by YES and NO children of a set N are obtained from the set accepted by N via intersection and complementation with regular sets, it follows by induction from the closure properties of these operations that every node in a single-symbol KCT accepts a regular set. In particular, each of the leaf nodes in a KCT accepts a regular set.

Now, a given class in an KCT may be associated with more than one leaf node - i.e., more than one leaf node may have the label associated with the particular class. A class corresponds to a union over leaf nodes. Since regular sets are closed under union, this implies that each class in a single-symbol KCT is represented by a regular set. To extend this proof to set-membership KCTs, one must prove that the expressions in the nodes of a set-membership KCT are regular, which is trivial.

Of course, this does not imply that the KCT-growing algorithms are capable of learning any given regular language from examples. As described above, Gold proved that learning a minimal DFA from positive and negative samples was NP-hard; Pitt and Warmuth proved that PAC-learning a DFA of size n by an NFA whose size is bounded above by any polynomial in n is also NP-hard. Since the KCT-growing algorithms are polynomial-time (see Chapter 7) it follows that there will be regular languages that cannot be exactly learned, nor PAC-learned, by KCTs. KCT-growing must therefore be regarded as a heuristic method that always learns **some** regular language approximation to the language that produced the training examples. The next section considers the question: when is the KCT grown from training data a good approximation?

4.4.3 Classification Properties of KCTs

This section examines the ability of KCTs to separate classes of strings. The discussion that follows considers only the case where we wish to classify a string s as belonging to L_1, the language generated by stochastic grammar G_1, or to L_2, the language generated by stochastic grammar G_2; however, the results could easily be extended to a larger number of classes.

KCTs and p-Concepts

A given string s may have non-zero probability of being generated by G_1 and non-zero probability of being generated by G_2. The p-concept approach is relevant here. Let $c(s)$ be the probability that s belongs to L_1, i.e. $c(s) = P(G_1|s)$. Similarly, the probability that s belongs to L_2 is $P(G_2|s) = 1 - c(s)$. Since $c(s)$ is a p-concept, a decision rule can easily be derived from it once it has been learned; in most cases, we will decide that s belongs to L_1 if $c(s) > 0.5$.

Each leaf node of a KCT represents a decision rule of the form "strings that have arrived at this node will be labelled X". However, KCTs can be converted into a data structure that estimates probabilities. First, grow the KCT on one set of labelled data; since we are dealing with the two-class problem, each training string will be labelled either 1 or 2. Next, use a new set of labelled strings to estimate $c(s)$ for each leaf node. For a particular leaf node N, the estimate $h(N)$ is simply the proportion of strings ending up in that node that is labelled 1. This modified KCT clearly approximates p-concepts $c(s)$, with each leaf node N providing the estimate $h(N)$ for the strings that end up in it.

What classes of p-concepts $c(s)$ defined on strings can be learned by KCTs? The theory of p-concepts cannot be applied directly to KCTs, which are obtained via the iterative expansion-pruning algorithm in a way that would be difficult to analyze formally. However, we can obtain an interesting result using a modified decision list closely resembling single-symbol KCTs.

Two questions are of interest:

1. Are there stochastic grammars G_1, G_2 such that no KCT can reliably decide whether a given string belongs to L_1 or L_2?

2. Given two stochastic grammars G_1, G_2 such that a KCT that can reliably determine membership in L_1 or L_2 exists, will this KCT actually be learned in a reasonable amount of time from examples?

These questions are closely related. The first question can be answered by producing an example of stochastic grammars G_1, G_2 such that a KCT which reliably determine membership in L_1 and L_2 is impossible to grow. Suppose that G_1 yields strings of the form $< aa(bc)^n aa >$, and G_2 yields strings of the form $< aa(cb)^n aa >$, with identical probability distributions on values of n. In general, it will be impossible to get the KCT-growing algorithms started - no question of the type "is the form of the string $< X >$?", "is the form of the string $< X+ >$?", "is the form of the string $< +X >$?", or "is the form of the string $< +X+ >$?" (whether the single-symbol or set-membership protocol is followed) yields a significant drop in impurity at the root.

We could deal with practical situations in which stalemates of this sort often arose by developing a new KCT protocol in which questions involving n-grams are permitted. In a protocol that allowed bigrams, a question such as "is the form $< +bc+ >$?" would be permitted at the root and this would

handle the example just given. However, for any protocol allowing questions about n-grams (value of n fixed), it is easy to produce a counterexample which requires a higher value of n to handle it. Thus, there are definitely cases of stochastic grammars G_1 and G_2 for which membership cannot be determined by KCTs.

The second question we wish to answer concerns situations where there is a possible KCT that would correctly model $c(s) = P(G_1|s)$. Will such a KCT actually be learned in a reasonable amount of time from sample strings generated by G_1 and G_2? Recall that Kearns and Schapire proved that a decision list with decreasing probabilities is learnable. That is, an algorithm exists for learning a list of the form $(f_1, r_1), (f_2, r_2), ..., (f_s, r_s)$ where an example has probability r_1 of being in class 1 if f_1 is $TRUE$, otherwise probability r_2 of being in class 1 if f_2 is $TRUE$, otherwise... - and so on (where the r_i are in decreasing order). It is assumed that the f_i are drawn from a list of potential questions $\{q_i\}$; the only restriction on these questions is that they must be Boolean. The algorithm is guaranteed to output with probability $1-\delta$ a hypothesized decision list h that is an (ϵ, γ)-good model for the probability $c(x)$ given by the **real** decision list that an example x belongs to class 1. Furthermore, the running time of the algorithm is polynomial in $1/\epsilon$, $1/\gamma$, $1/\delta$, in the total length of the question list $\{q_i\}$, and in the maximum time required to evaluate any question q_i.

Suppose we have access to labelled strings from stochastic grammars G_1 and G_2. Let $\{q_i\}$ be the questions considered at the root of a single-symbol KCT. That is, we allow questions of the type "is the form of the string $< X >$?", "is the form of the string $< X+ >$?", "is the form of the string $< +X >$?", and "is the form of the string $< +X+ >$?", where X is a variable allowed to take on the value of any symbol in the alphabet Σ. There are $4 * |\Sigma|$ such questions. If we run the Kearns-Schapire decision list algorithm using this list of questions, we obtain a probabilistic decision list that with probability $1 - \delta$ gives us an (ϵ, γ)-good model for the probability $c(s)$ that a string s was produced by G_1, *in terms of the questions asked.* The list will be obtained in time polynomial in $1/\epsilon$, $1/\gamma$, $1/\delta$, and alphabet size $|\Sigma|$. If we regard ϵ, γ, and δ as fixed, the time to find the list is polynomial in $|\Sigma|$.

Figure 4.1 shows how the Kearns-Schapire algorithm can be iterated to produce a bloated kind of KCT. Since the lexicon is $V = \{b, c, m\}$, at the

root

$$\{q_i\} = \{< b >?, < c >?, < m >?, < b+ >?, < c+ >?, < m+ >?, < +b+ >, < +c >?, < +m >?, < +b+ >?, < +c+ >?, < +m+ >?\}.$$

The algorithm then arranges these questions so that the probabilities $r_1, ..., r_{13}$ shown for iteration 1 are in decreasing order. The chain of questions generated in this iteration will act as a backbone from which future questions grow; it contains $O(|\Sigma|)$ questions.

During iteration 2, single-symbol questions from the known structure $< +b+ >$ for the YES child of the root are generated; there will be $O(|\Sigma|)$ of these. Using a new set of examples generated by G_1 and G_2, we run the Kearns-Schapire algorithm with the new questions filling the role of $\{q_i\}$. Again, we are guaranteed to find a probabilistic decision list that is a good estimate of the corresponding conditional probabilities in time polynomial in $|\Sigma|$.

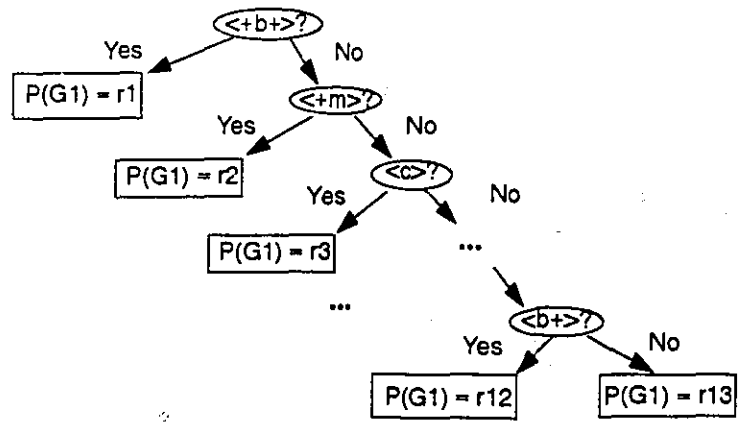
We now run the Kearns-Schapire algorithm on the YES child of the second question in the backbone, $< +m >$? Again, we get a probabilistic decision list in time polynomial in $|\Sigma|$. We continue until every one of the $O(|\Sigma|)$ YES children in the backbone has a probabilistic decision list attached to it. The whole process takes $O(|\Sigma|^2)$ time, since an $O(|\Sigma|)$ algorithm is being run $O(|\Sigma|)$ times.

Thus, the final structure will consider every possible question involving exactly two symbols in a way that yields, with probability $1-\delta$, an (ϵ, γ)-good model of the corresponding conditional probabilities (these are stored in the leaves). The structure takes $O(|\Sigma|^2)$ time to build. In general, if we wanted to consider all questions involving k symbols, we would require $O(|\Sigma|^k)$ time.

We have now shown that the class of p-concepts given by $P(G_1|K(s))$, where $K(s)$ is the known structure for a string s, is learnable in polynomial time by the iterated version of the Kearns-Schapire algorithm just given. The running time is polynomial in the size of the alphabet for fixed k (the maximum number of symbols in a known structure). This does not prove that the KCT-growing algorithms we actually use produce an (ϵ, γ)-good model with probability $1 - \delta$, but it makes such a claim plausible.

V = {b,c,m} => Question List = {<b>?, ... , <c+>?, ... , <+m+>?}

Iteration 1:



Iteration 2:

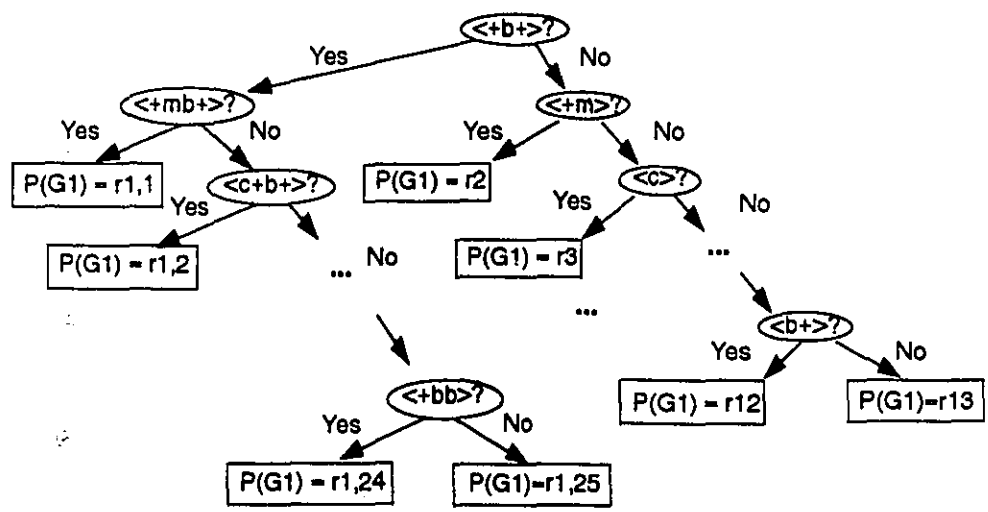


Figure 4.1: First 2 Iterations of Kearns-Schapire Alg. on KCT-Style Question List

When Do KCTs Perform Well?

Formal analysis normally considers the worst case: how badly a given method might perform. Later chapters of this thesis present evidence that KCTs perform quite well on some natural language tasks. This fact may tell us something about natural language. Reversing the usual question, one therefore asks: in what situations would one expect KCTs to perform well?

The performance of KCTs in deciding whether a string s belongs to L_1 or L_2 has little to do with the position of these languages in the Chomsky hierarchy. Rather, it depends on the way in which the corresponding grammars G_1 and G_2 use the symbols in the alphabet. If G_1 and G_2 use these symbols with different frequencies, KCTs find L_1 and L_2 easy to distinguish; if symbols are used with similar frequencies, KCTs do not easily distinguish L_1 and L_2.

For a trivial example, suppose that L_1 is the language denoted by the expression $a^n z b^n$, where $n \geq 1$, and L_2 is the language denoted by $a^n b^n$, where $n \geq 1$ (both with some consistent probability assignment to strings). The KCT consisting of the single question "is the form $< +z+ >$?" is capable of deciding membership in L_1 or L_2 with 100% accuracy. Neither L_1 nor L_2 is regular [Hop79 pp. 61-62]. Thus, one cannot conclude from the regularity of the sets accepted by KCTs that they only perform well in deciding membership in regular languages.

Figure 4.2 presents a more interesting example. Here, G_1 and G_2 differ only in the probability values for the rules associated with the symbols x and z. The set of strings produced by L_1 and L_2 is the same, but some of them differ in their probability of occurrence. If the *a priori* values $P(G_1)$ and $P(G_2)$ are identical, one can show by the Bayes theorem that $c(aab) = P(G_1|aab) = 0.5$. On the other hand, the string $aaxb$ is more likely to be produced by G_1 than by G_2. By the Bayes theorem, $c(aaxb) = P(G_1|aaxb) = 0.67$.

For this example, the only information about a string that makes either L_1 or L_2 more likely than the alternative is the fact that it contains x or z. Any further details about the string are irrelevant. The hypothetical KCT shown yields the best possible estimate $h(s) = c(s)$ for the probability that a string s was produced by G_1: no other data structure or grammatical inference method can do better than this. Assuming that a KCT containing the questions shown here is obtained from training data, the probabilities

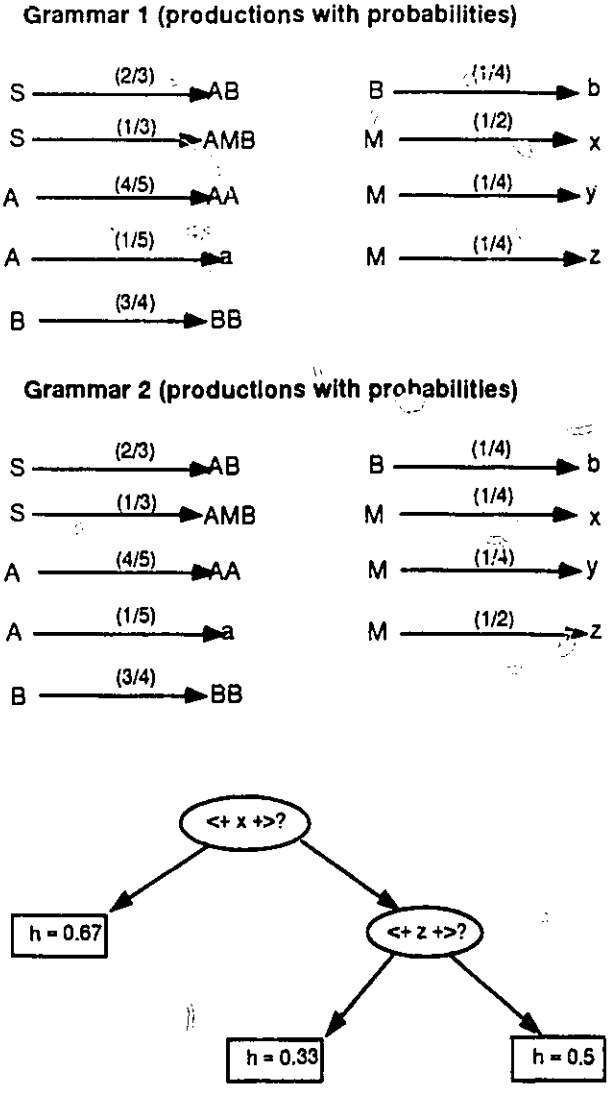


Figure 4.2: p-Concept KCT for 2 Stochastic Grammars, where $P(G_1) = P(G_2)$

$h(s)$ at the leaves can be estimated from additional training data, and will rapidly converge to $c(s)$.

4.4.4 Discussion

In a survey of pattern recognition, K.S. Fu stated: "Many mathematical methods have been proposed for solving pattern recognition problems. They can be grouped into two major approaches, the decision-theoretic or statistical approach and the structural or syntactic approach" [Fu86c pg. 398]. These two approaches differ in their goals, as well as in the techniques they employ. The decision-theoretic approach aims at making a decision about new patterns on the basis of statistical classification rules learned from patterns seen earlier. The syntactic approach tries to describe the nature of the rules that generated a set of patterns, and may use this information to make a classification decision about new patterns. The syntactic approach is thus more ambitious than the decision-theoretic approach: in addition to classifying the data, it attempts to explain the process that gave rise to them.

Without exception, all the papers I have read on machine learning of string patterns assume that the syntactic approach is the only possible one. Nobody seems to have posed the following question: can strings be classified without full syntactic analysis? In other words, what method should one employ if one's only interest in a set of strings is decision-theoretic rather than syntactic? KCTs fill a hitherto empty niche. They are a means of achieving for string data the modest goal of decision theory - that of classifying the data correctly - rather than the more ambitious syntactic goal of characterising the processes that generated the data.

This accounts for some of the apparent idiosyncrasies of KCTs:

- KCTs permit gaps - arbitrary substrings of length one or more that can be made up of any symbols from the alphabet. In fact, operations performed on gaps lie at the heart of the KCT-growing algorithms. No grammatical inference technique I know of permits gaps, since the aim of such techniques is to account for all the symbols in a given string.

- The performance of KCTs depends on the extent to which, in a given domain, the class of a string correlates with the identity of the terminals it contains, i.e. with string composition. Natural languages have a huge

number of possible terminals (words); only a fraction of the lexicon can appear in a given sentence. If one were interested in classifying sentences by topic, an unordered list of the words occurring in the sentence would be quite helpful. Many natural language classification problems thus seem ideally suited to KCTs. By contrast, a domain in which different string classes tend to have the same terminals arranged differently might be better-suited to another technique.

- Most work in grammatical inference considers only two classes of strings: those generated by a grammar G and those not generated by G. When more than two classes are considered, a separate grammar is inferred for each. Thus, a new string is parsed first by G_1, then by G_2, G_3 and so on until a grammar G_i is found such that the parse is successful. If the grammars are stochastic, the successful parse is the one which yields the highest probability of generating the string. A single KCT can separate strings belonging to several different classes in a small number of steps.

- Although the leaf nodes of a KCT accept regular languages, KCTs can reliably tell apart strings produced by some pairs of grammars G_1, G_2 higher up in the Chomsky hierarchy - if certain conditions are placed on production probabilities in those grammars. Broadly speaking, good performance is possible when at least one of the two grammars generates strings containing regular patterns **not** produced by the other grammar as substrings.

- Growing a KCT on a set of training strings is usually much faster than inferring a grammar from the same data, as is using a KCT to classify a new string. In return for giving up the goal of explaining how the data arose and concentrating on classification, KCTs obtain speed advantages.

KCTs are not the only possible decision-theoretic learning method for string classification rules. As others arise - based, for instance, on the theory of Bayesian networks [Char91] - it will be important to develop theoretical criteria for assessing them. The PAC learning paradigm (including p-concepts), with its emphasis on computability and on probabilistic performance guarantees, promises to supply such criteria. I have devoted considerable space

to this relatively new, incomplete paradigm because I am convinced that it will be of great importance in the future. In the long term, work in machine learning in general and string classification in particular will grow out of closer collaboration between theory and practice.

In the medium term, the validity of KCTs must be established empirically. Like bigrams and trigrams for language modeling, KCTs seem to yield simple, rapidly-trainable models that approximate linguistic reality by ignoring some aspects of it. Chapter 5 presents binary decision trees, of which KCTs are a specialised form; Chapter 6 presents KCTs themselves.

Chapter 5

Classification Trees in Speech Processing

5.1 What is a Binary Classification Tree?

A binary classification tree is a binary tree each of whose internal nodes consists of a yes-no question, a YES subtree, and a NO subtree, and each of whose leaf nodes is labeled with a category. To classify a data item, apply the question at the root to it and enter either the YES or the NO subtree, depending on the answer to the question; recurse until the data item arrives at a leaf node, whose category is assigned to the item.

Thus, using a binary classification tree is analogous to playing the game "Twenty Questions" - except that in pattern-recognition applications there is no rule limiting the depth of the tree to twenty. The most difficult part of this game is the formulation of a good tree of binary questions; once this tree has been devised in the course of several games, it can be used over and over again with little mental effort. Similarly, the trickiest part of binary tree classification as a statistical technique is the generation of the tree from training data; once it has been grown, classifying new data is trivial. The structure of the completed tree is easy for people to understand and often yields a deeper understanding of the hidden processes generating the data, which is not always true of competing statistical techniques.

In the last few years efficient algorithms for growing binary classification trees from training data have been devised. Many of these are contained in a

book sometimes called the "Bible" or "Bhagavad Gita" of classification tree methodology: "Classification and Regression Trees" by Breiman, Friedman *et al* [Bre84]. Some important recent contributions to the methodology are [Cho91, Gel91, Nad91]. To employ these tree-growing algorithms, one must supply three elements [Bre84]:

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

Note that the set of *possible* questions one supplies to the tree-growing algorithm is normally much larger than the set of questions ultimately assigned to nodes of the tree. Any question that yields a "YES" or "NO" answer when applied to the type of data one is studying is permissible in principle. The beauty of the classification tree approach is that if one thinks a certain aspect of the data *might* conceivably be relevant to classification, one can supply a question reflecting this aspect to the tree-growing algorithm and allow it to make the final decision. The algorithm discards most of the possible questions and determines the placement in the tree of the remaining questions on the basis of training data.

The classification tree method can be applied to a wide variety of problems: [Bre84] discusses a tree that identifies heart attack patients who risk having a second, fatal attack within the next month, a tree that determines the class of a ship from its radar range profile, and a tree that detects the presence of bromine from mass spectrometer measurements. Figure 5.1 [*ibid* pg. 2] shows the tree that identifies high-risk heart attack patients: G means high risk, F means low risk. People who had suffered a mild heart attack and were admitted to a certain medical centre underwent careful examination involving measurement of 19 variables: blood pressure, age, and so on. If they died within the next 30 days they were classified as G, otherwise they were classified as F. The resulting corpus of data was used to grow the tree in the figure.

Note that only three of the 19 variables (minimum systolic blood pressure, age, and presence or absence of sinus tachycardia) turned out to be relevant for classification. Sophisticated statistical regression techniques applied to

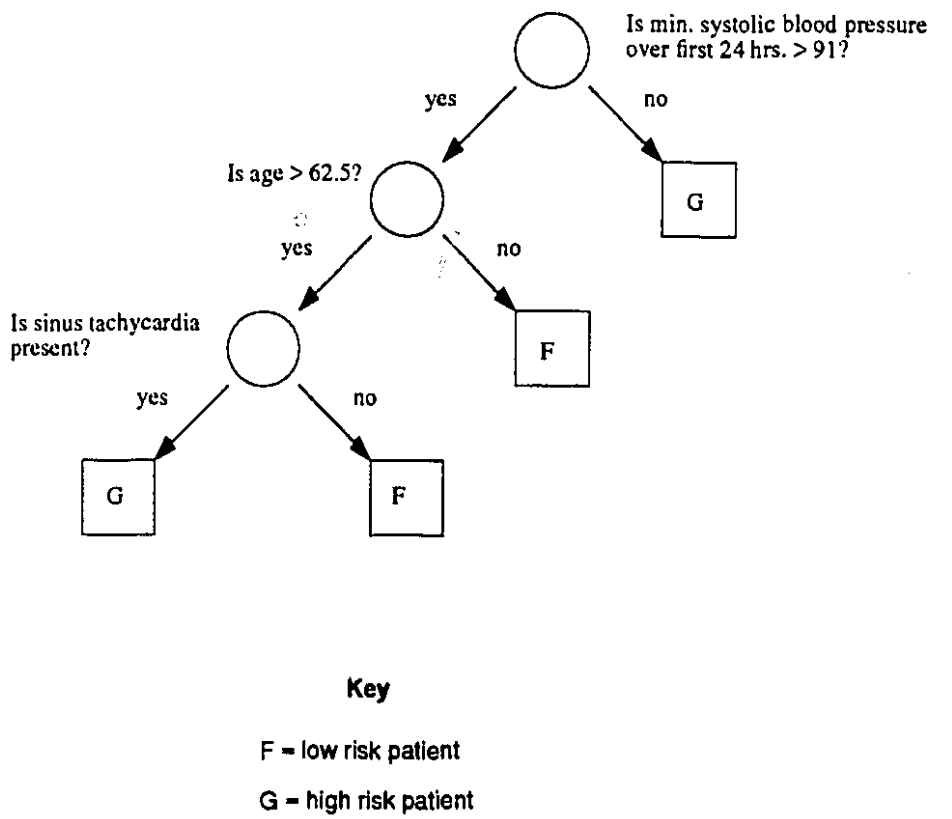


Figure 5.1: Patient Classification Example [Bre84, pg.2]

the same data yielded complex formulae involving most of the 19 variables, but these turned out to be less accurate than the tree in classifying new data. Classification trees often have a competitive advantage over other statistical approaches when the data are highly complex. They should be considered when the data have the following characteristics [Bre84 pg. 7]:

- High dimensionality;
- A mixture of data types;
- Nonstandard data structure;
- Nonhomogeneity - i.e., different relationships hold between variables in different parts of the measurement space.

This chapter discusses general algorithms for growing classification trees, and applications of these trees at other levels of speech processing. All algorithms and formulae are given for the case where the cost of misclassifying a class i item for a class j item is the same for all i and j, but they are easy to modify for non-uniform misclassification costs.

In subsequent chapters, I will show how to grow specialized classification trees that aid speech understanding by dealing with word sequence hypotheses generated by the speech recognition component. Note that most of the characteristics listed above apply to word sequence hypotheses:

- Their dimensionality varies and can be quite high;
- They consist of a mixture of nouns, verbs, and other information-carrying parts of speech with meaningless interjections and repetitions;
- They are highly non-homogeneous in the sense that one sentence may require elaborate computation based on all its words, while another may require very little processing.

5.2 Splitting Rules and Stopping Rules

As mentioned above, one must supply three elements in order to grow a binary classification tree:

1. The set of possible questions;

2. A rule for selecting the best question at a node, or deciding that it should be a leaf node;

3. A method for pruning trees to prevent over-training.

The set of possible questions depends on the application. This section describes the Gini criterion for selecting a question and a stopping rule that decides when a node is a leaf node; the next section describes pruning techniques.

Consider the items in the training data that end up at a newly generated node. Since we know the category of each item, we can define a measure of the "impurity" of the node with the following properties:

- The impurity is always non-negative;
- A node containing equal proportions of all possible categories has maximum impurity;
- A node containing only one of the possible categories has impurity of 0 (the minimum possible impurity).

There are several functions satisfying these conditions; all depend only on the counts of each category within a node. Breiman, Friedman, *et al* [Bre84] considered several possible impurity measures. They found that the misclassification rate of the tree is quite insensitive to the function chosen, as long as it belongs to a set of functions with reasonable properties. From this set, they ultimately chose the Gini criterion. The Gini criterion is a measure of impurity that always lies between 0 and 1. If T is a certain node and $f(j|T)$ is the proportion of items in the node that belong to category j, then the Gini impurity $i(T)$ of the node is defined as

$$i(T) = \sum_{j \neq k} f(j|T) f(k|T) = 1 - \sum_{j} f^2(j|T).$$

For instance, suppose there are three possible categories, A, B, and C. Consider a given node T at which 10 items in the training data arrive. Of these 10 items, 5 belong to category A, 3 to category B, and 2 to category

C. Then $f(A|T)$ is 0.5, $f(B|T)$ is 0.3, and $f(C|T)$ is 0.2. From these values and the equation, we deduce that the Gini impurity $i(T)$ is 0.62.

The best question for a node is considered to be the question which brings about the greatest drop in impurity in going from the parent node to its children. In other words, if $i(T)$ is the impurity of node T, 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 ΔI. The process of picking a question for a node, generating the YES and NO children of the node, and dividing up the strings at the parent node between the two children is called *splitting* the parent node.

Consider the node T above, which had 5 category A items, 3 category B items, and 2 category C items. Suppose one of questions Q_1 and Q_2 must be chosen, where the YES child of Q_1 contains 4 A items and 1 C item, and its NO child contains 1 A item, 3 B items, and 1 C item, while the YES child of Q_2 contains 3 A and 3 B items and its NO child 2 A and 2 C items. Then the impurity of Q_1's YES child is 0.32 and the impurity of its NO child is 0.56, so

$$\Delta I(Q_1) = 0.62 - (0.5)(0.32) - (0.5)(0.56) = 0.18.$$

The impurity of both the YES and the NO child of Q_2 is 0.5, so

$$\Delta I(Q_2) = 0.62 - (0.6)(0.5) - (0.4)(0.5) = 0.12.$$

Q_1 brought about the greatest drop in impurity, and will therefore be preferred to Q_2.

Clearly, the children of a given node will be less impure than their parent. If this process is carried out recursively to generate grandchildren, great-grandchildren, and so on of the original root node, some of the descendants of the root may have impurity of 0. These nodes will be designated leaf nodes, and labelled with the name of the single category they contain. Other leaf nodes will be designated as such because there is no way of reducing the impurity any further, or because the maximal ΔI is too small, or for

some other reason. These leaf nodes will be labelled with the name of the most common category they contain; ties are broken arbitrarily. The criteria used to determine when a node should be prevented from splitting further by declaring it a leaf node are called the "stopping rules".

Breiman *et al* initially experimented with a variety of stopping rules to obtain the best-sized tree. For instance, they considered different numerical values for a threshold b such that a node is considered to be a leaf node if the maximal ΔI (over the set of questions) is less than b. None of these stopping rules achieved the desired goal of obtaining a tree with strong predictive power. The reason is that in many cases, a node for which the best question gives an unimpressive ΔI has children which can achieve high values of ΔI; a stopping rule that turns the original node into a leaf prevents these valuable child nodes from being born.

These researchers ultimately adopted a tree-growing strategy with two stages: first grow a tree that is much too large using a simple stopping rule, then prune the tree upwards from the leaves using an independent data set. The simple stopping rule they adopted was to keep splitting nodes until for each terminal node, either there are fewer than N items (N close to 1), or the maximal value of ΔI is 0. The growing-pruning strategy yielded better results than the most sophisticated stopping rules they tried.

Before we consider pruning, an important point about the choice of question at each node must be made. This choice is **not** necessarily optimal for the tree as a whole. Ideally, we would choose the question that ultimately led to the tree with the lowest misclassification rate. This would require consideration of all possible subtrees of the current node and is computationally impractical. Use of the Gini criterion is a greedy heuristic that gives good global results most of the time. If the set of possible questions is small, one can employ a look-ahead version of the algorithm in which the criterion is the impurity of the grandchildren or greatgrandchildren of the current node, rather than the impurity of the children. Once the number of layers of look-ahead is greater than two or three, this again becomes computationally impractical.

5.3 Pruning Techniques

A tree grown with a tolerant splitting rule like the one utilized by Breiman *et al* will not perform well on new data - it will be "overtrained". Overtraining occurs when a predictor is so well-fitted to its training data that its ability to predict new data is handicapped. For instance, if we have 15 data points (x, y) we could fit them perfectly with a polynomial in powers of x up to x^{14}. However, it is unlikely that this polynomial will accurately predict the value of y for a new x. We would be better off using the 15 points to estimate accurately the parameters of a cubic, or of some other lower-order polynomial. Similarly, too large a classification tree will have a higher true rate of misclassification than some smaller trees, despite better performance on the training data. Picking the best-sized tree is a compromise between a tree that is too small, ignoring useful information in the training data, and a tree that is too large and therefore overtrained.

Breiman *et al*'s **CART Cross-Validation** approach to this problem is complicated and computationally expensive; their description of it is a confusing part of what is otherwise a beautifully clear book [Bre84 pp. 59-92]. Recently, Gelfand *et al* [Gel91] proposed a new **iterative expansion-pruning** approach that is elegant, computationally cheap, simple to implement, and guaranteed to perform as well as or better than the Breiman *et al* approach. The Keyword Classification Trees (KCTs) described later in this thesis were all grown by means of the iterative Gelfand *et al* approach.

5.3.1 The CART Cross-Validation Approach to Pruning

Once the initial, too-large tree has been grown, we are no longer concerned with Gini impurity. To prune the tree, Breiman *et al* calculate the *resubstitution misclassification rate* $R(n)$ for each node n. First, give each internal node, as well as each leaf node, the label of the most common data type that passed through that node.

Figure 5.2 illustrates this. The tree shown here was grown on 100 data items, with equal numbers falling into classes A, B, C, D, E. At the root, the classes are tied; the tie was broken arbitrarily in favour of the label A. When the question at the root was chosen, 52 data items yielded a "yes" answer and went into the left child, 48 yielded a "no" answer and went into the right

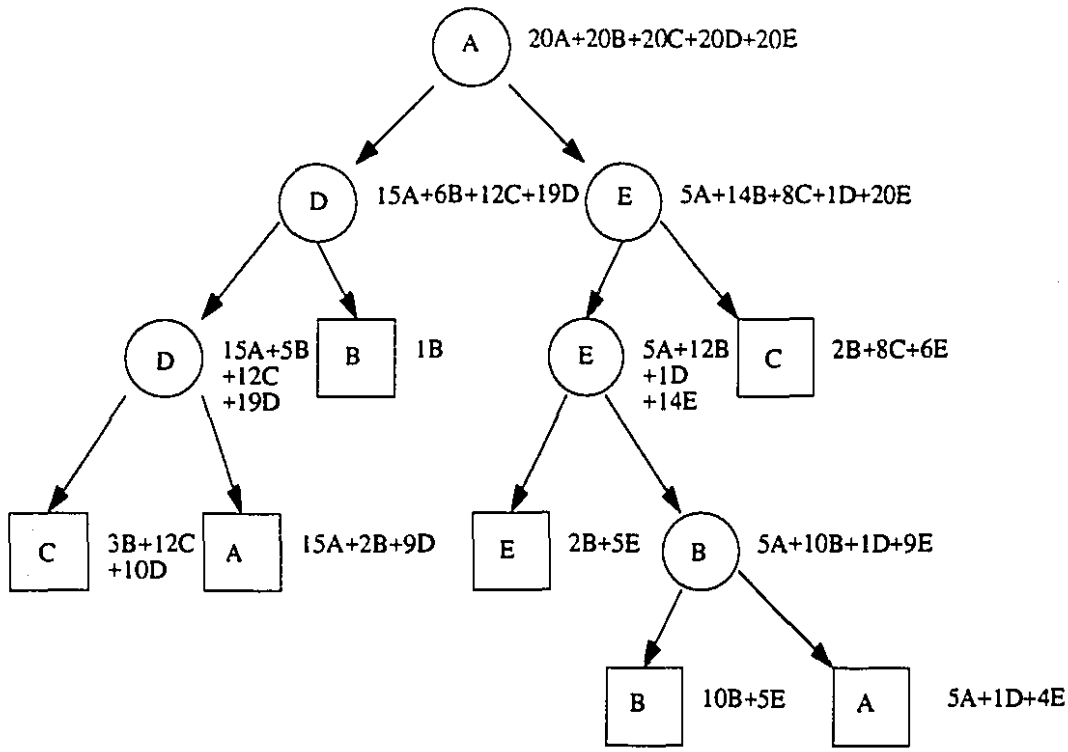


Figure 5.2: Classification Tree (Training Data Items shown at each Node)

child. Among the 52 items in the left child of the root, the largest group was labeled D (19 of these) so the label of this node is D; similarly, the presence of 20 E items in the right child of the root causes this node to be labeled E. Each data item ends up at one of the leaf nodes; of course, the total count of items shown in the leaf nodes is 100. I made up this example - a tree grown on 100 items using the Gini criterion would normally be much bigger, with only a few items per leaf node (unless the set of possible questions was very restricted).

$R(n)$ is defined as the ratio of the number of items misclassified by the label at node n to the **total** number of data items used to grow the tree. For instance, the leftmost leaf node (with label C) contains 13 items that do not

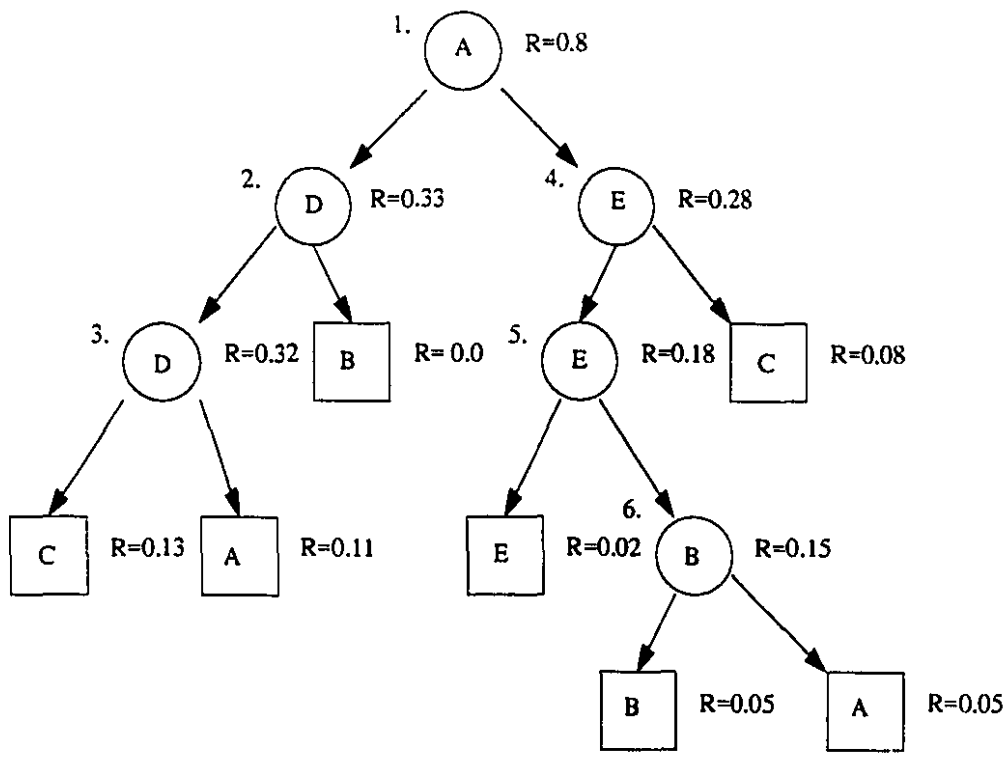


Figure 5.3: Same Tree, $R(n)$ shown for each Node n

belong to class C (3 B and 10 D); divide this by 100 (**not** 25, the number of items in this node) to get an $R(n)$ of 0.13 for this node. Figure 5.3.1 shows $R(n)$ for all the nodes in the tree.

In a tree grown on real data when the stopping rule was "keep splitting until all leaf nodes are as pure as possible", one would expect to see most leaf nodes with an $R(n)$ of 0.0. The sum of the leaf node $R(n)$s gives the misclassification rate of the whole tree on the data it is trained on, which will be close to 0.0. As explained above, this is an overoptimistic estimate of the misclassification rate on new data.

To obtain a good pruned tree from the original tree, Breiman *et al* first obtain a family of pruned trees. For an internal node n, let $R(n_{sub})$ be the

resubstitution misclassification rate of the subtree at n, defined as the sum of $R(i)$ over descendants of n that are leaf nodes. For instance, the node numbered 2 in figure 5.3.1 has three leaf node descendants, labeled C, A, and B respectively; thus

$$R(2_{sub}) = 0.13 + 0.11 + 0.0 = 0.24.$$

Also, for each internal node n, let $|sub(n)|$ denote the number of leaf node descendants it has; for the same internal node 2, this is 3. Finally, for each internal node n, define

$$g_1(n) = [R(n) - R(n_{sub})]/[|sub(n)| - 1].$$

Figure 5.4 shows the g_1 values for the internal nodes in the tree. Note that, for instance, $g_1(2)$ is $(0.33-0.24)/(3-1)$. Breiman *et al* call the internal node n with the lowest value of g_1 the "weakest link". In the example, this happens to be node 4. The first member of the family of pruned subtrees is obtained by turning the weakest link into a leaf node and pruning its descendants, as shown in figure 5.5.

What is the rationale for this procedure? The numerator of $g_1(n)$ tells us how much more misclassification we get if we prune the subtree of n and just classify data by the label at n. Obviously, if this numerator is small the subtree of n gives only a small improvement over n alone. If the denominator is large, there are a lot of leaf nodes in the subtree of n. Thus, a small value for $g_1(n)$ implies that if we turn n into a leaf node, we can get rid of a lot of nodes without greatly increasing misclassification of the training data.

To obtain the other members of the family of pruned subtrees, one must recalculate the $g_1(n)$s in the new tree, turn the internal node with the lowest value of $g_1(n)$ in the new tree into a leaf node, and iterate until there is nothing left but the root. Compared to the computational cost of growing the original tree, these steps take little computation.

We now have a family of pruned subtrees of the original tree. If we have plenty of data, the original tree should be grown on most of the data. To pick the best member of the family of pruned subtrees, test each on the remaining data and pick the one with the lowest misclassification rate. (If we grew and tested the trees on the same data, the first, largest pruned subtree would always be chosen). Unfortunately, this procedure is a waste of data.

To use the available data more efficiently, Breiman *et al* devised a complicated cross-validation scheme:

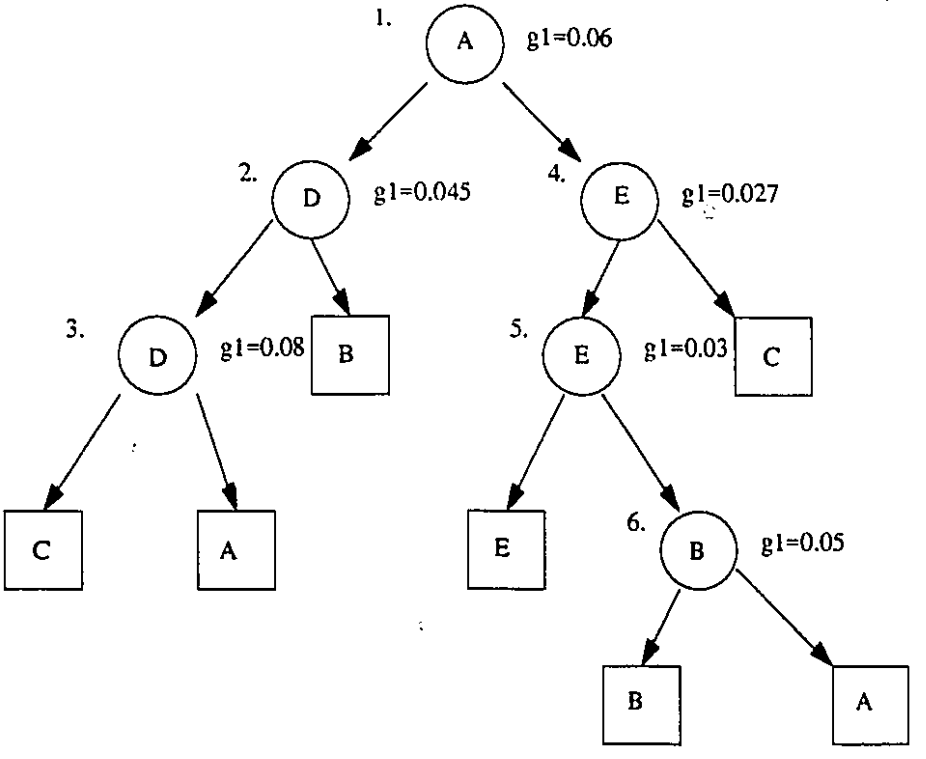


Figure 5.4: Same Tree, $g_1(n)$ shown for each Node n

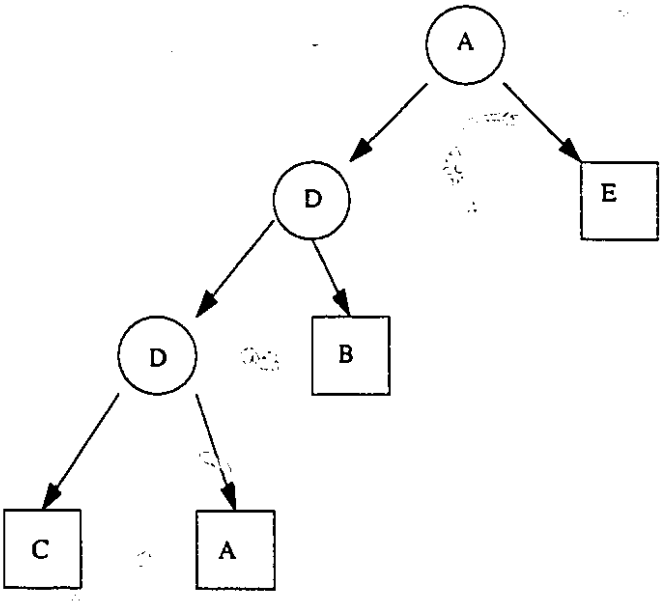


Figure 5.5: Same Tree after Pruning of Subtree with lowest g_1

1. Divide the data into V disjoint subsets.

2. Employ $V - 1$ of these subsets to grow a too-large tree and the corresponding family of pruned subtrees; find the best pruned subtree from this family by testing on the subset that was **not** used for growing the tree.

3. Repeat this V times, each time reserving a different subset for finding the best subtree.

4. One now has V best pruned subtrees. From these, one can obtain an estimate of "how big" the optimal subtree should be (actually, a certain tree complexity parameter is estimated).

5. Now grow a too-large tree on **all** the data, obtain the family of pruned subtrees, and pick the one that is "the right size" (i.e. has close to the estimated value of the complexity parameter).

Fortunately, just as I was beginning to implement this cumbersome and computationally expensive procedure, I encountered an article describing the Gelfand *et al* approach [Gel91].

5.3.2 The Iterative Expansion-Pruning Approach

Gelfand *et al*'s iterative algorithm is easy to understand and to implement. In my description of it, I will (unlike Gelfand *et al*) call the process by which a tree acquires new nodes "expansion" (rather than "growth"). The removal of nodes will still be called "pruning", and the process of obtaining the final tree via cycles of expansion and pruning will be called "growth". The Gelfand *et al* algorithm will therefore be referred to as the "expansion-pruning" algorithm.

Compared to the cross-validation procedure described above, the expansion-pruning algorithm has several advantages. In the cross-validation procedure, a pruned subtree is selected from a parametric family of pruned subtrees which represents only a small subset of all possible subtrees of the original tree. There is no guarantee that the parametric family of pruned subtrees contains even a single member capable of yielding good performance on new data, because the parametric family was obtained using only information contained in the training data. In the pruning step of the expansion-pruning

algorithm, a somewhat better estimate of the misclassification rate is minimized over **all** possible pruned subtrees, not just a subset of them. This algorithm thus tends to yield a final tree with a lower misclassification rate on new data. Furthermore, the cross-validation procedure requires growing many trees that are later discarded. The expansion-pruning algorithm constructs the bulk of the final tree in the first few iterations and does comparatively little work thereafter, so it is computationally cheaper than cross-validation.

The expansion phase of this algorithm works exactly as in the Breiman *et al* algorithm, using the Gini criterion (or some similar criterion) to pick questions and a simple stopping rule that generates overtrained trees. Labels are again assigned to all nodes in the expansion phase, as described above. Since the pruning phase that follows will cut off leaf nodes (and often internal nodes) without doing any labelling of its own, internal nodes *must* be labelled during the expansion phase. Otherwise, we would obtain new leaf nodes with no labels at the end of the next pruning phase. To expand a pruned tree, one retains the labels on the leaf nodes but selects new binary questions for them so as to produce YES and NO children of minimal impurity. These children are split recursively, in the usual way, until pure leaf nodes are obtained.

The most important aspect of the expansion-pruning algorithm is that the training data are split into two disjoint sets of approximately equal size, which I will call F and S (for "first" and "second"). After being expanded on one set, the tree will be pruned on the other, then expanded on the set it was just pruned on; this process is iterated until two successive pruned trees are of the same size. If we let T_0 stand for the original empty tree, the iterations can be described as follows:

- Expand T_0 on F to get T_1;
- Prune T_1 on S to get T_2;
- Expand T_2 on S to get T_3;
- Prune T_3 on F to get T_4;
- Expand T_4 on F to get T_5;
- Prune T_5 on S to get T_6;

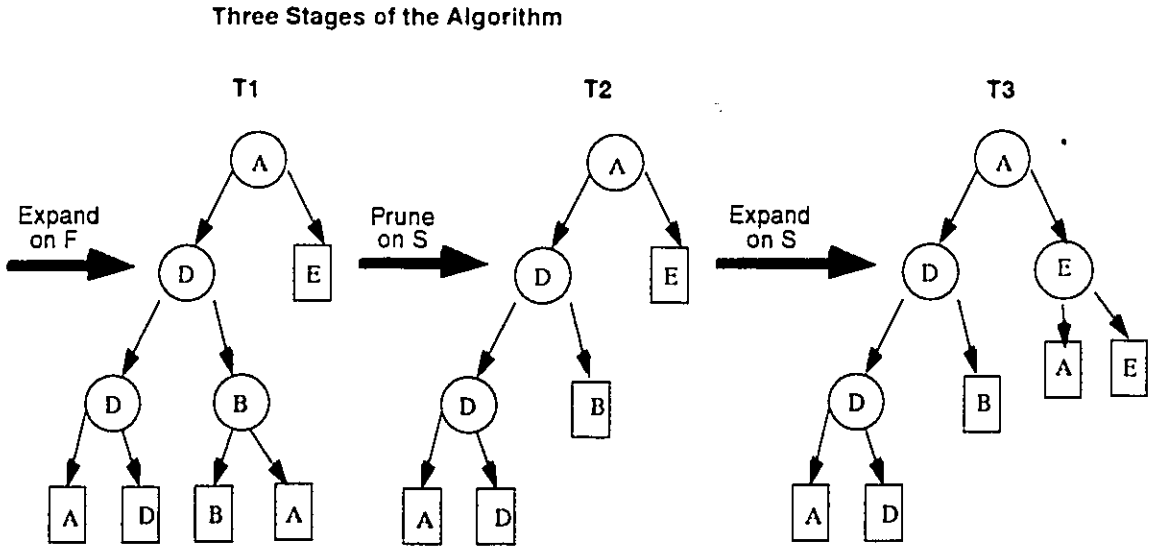
and so on. Eventually two successive pruned trees - for instance, T_6 and T_8 - will have the same number of nodes, and we set the final tree to be the second one. Gelfand *et al* have proved that this must happen, and that when it does the two successive pruned trees must be identical (i.e. in the example, T_6 and T_8 will be the same tree). They also state informally that the number of iterations is small in practice, and so far my results bear them out: I have never needed to go beyond growing T_6 (which was the same as T_4). The pruning phase is always much faster than the expansion phase. Figure 5.6 shows part of this cycle.

The pruning algorithm takes as input a set of data and a tree. First, all data items are shunted to the leaf nodes by the binary questions in the tree; then the algorithm works backwards from the leaves to the root. For each parent of two leaf nodes, consider what number of items in the two children will be misclassified if we assign to all of them the label of the parent. Compare that with the number of items that will be misclassified if we assign to items in the YES child the label of the YES child, and to items in the NO child the label of the NO child. It is quite possible that fewer items are misclassified if we label all of them with the parent's label. If so, cut off the two children; if not, keep them.

This idea can be extended. For an internal node n, we want to compare the number of items arriving at the node that will be misclassified if we give them the label at n with the number that will be misclassified by the subtree whose root is n. If the label at n is a better way of assigning the class than the subtree rooted at n, the subtree should be pruned. Working recursively from the leaves to the root until no more nodes can be pruned, the iterative expansion-pruning algorithm carries out this idea.

Therefore, certain YES-NO pairs of child nodes that are initially retained because they yield less misclassification than their immediate parent may be removed later. This occurs when a more distant ancestor of these nodes (a grandparent or greatgrandparent) yields even less misclassification than any possible pruned subtree descending from it. All descendants of this ancestor will be pruned.

To implement the algorithm, define a parameter $\hat{R}(n)$ that will be stored at each node n in the classification tree. $\hat{R}(n)$ is the proportion of the **total** number of data items that is misclassified by the label at node n. This is **not** equivalent to $R(n)$ in the cross-validation procedure, since it is estimated on a data set other than the set on which the last tree expansion took place



Basic Idea of Algorithm

- Split training data into two data sets, F and S.

- Expand T1 on F, letting nodes be split until leaf nodes are "pure" (only 1 class of item in each leaf node). T1 is now overtrained.

- Prune T1 on S to get T2.

- Expand T2 on S to get T3 (with pure leaf nodes).

- Prune T3 on F to get T4.

- Iterate until 2 successive even-numbered Ts are the same size.

Advantages

- Unlike Breiman et al pruning, minimizes misclassification over all pruned subtrees;

- Much faster than Breiman et al algorithm;

- Easy to implement;

- Efficient use of training data.

Figure 5.6: The Expansion-Pruning Algorithm

and is defined at leaf nodes as well as internal nodes. As with $R(n)$, it is important to remember that $\hat{R}(n)$ is **not** the proportion of the number of items in node n that are misclassified by the label of n. If the data set used for pruning has 100 items in it, 5 of these reach a node n, and 2 of these 5 are misclassified by the label of n, then $\hat{R}(n)$ is 0.02 rather than 0.4. I stress this point because I made this error when first implementing the algorithm!

Another parameter $\hat{S}(n)$ is also stored at each node. $\hat{S}(n)$ represents the proportion of the total number of data items that is misclassified by the best possible pruned subtree obtained from the original subtree rooted at n. Our data structure for the tree will have storage at each node for $\hat{R}(n)$, for $\hat{S}(n)$, and for a 'done' bit (the 'done' bit is not really necessary but makes the algorithm easier to understand).

Here is the pruning algorithm, which takes as input a classification tree with all nodes labelled and a set of data:

1. Read in the tree; set the 'done' bits of the leaf nodes to 1 and all other bits to 0.

2. Use the tree to classify the data, and record the value of $\hat{R}(n)$ for all nodes n, both internal and leaf. The data can be discarded once this has been done.

3. For each leaf node n, set $\hat{S}(n)$ to $\hat{R}(n)$.

4. Carry out the following step recursively (you will be working from the leaves to the root), stopping when all nodes in the tree have the 'done' bit set to 1. The step is carried out on those nodes of the tree that have a 'done' bit set to 0, but whose children both have a 'done' bit of 1. Denote such a node by n, and its children by YES and NO. Compare $\hat{R}(n)$ with $\hat{S}(YES) + \hat{S}(NO)$. If $\hat{R}(n)$ is smaller than or equal to this sum, dispose of the YES and NO descendants and set $\hat{S}(n)$ to $\hat{R}(n)$; otherwise, leave the children where they are and set $\hat{S}(n)$ to $\hat{S}(YES) + \hat{S}(NO)$. Now set the 'done' bit for n to 1.

Gelfand *et al* have proved that this fast, simple algorithm finds the optimal pruned subtree of the original tree.

This completes the description of the iterative expansion-pruning algorithm. When I first read and implemented the algorithm, there was one

question that puzzled me: why do more than two iterations ever occur? It seemed to me that after expanding the empty tree T_0 on data set F to get T_1, then pruning T_1 on S to get T_2, then expanding T_2 on S to get T_3, then pruning T_3 on F to get T_4, one should always have T_4 identical to T_2. This may happen, but frequently it does not - further iterations are often necessary. I will give the reasoning that led me to this false conclusion, and then show what is wrong with the reasoning. In the process, the reader may gain a deeper understanding of the algorithm.

My reasoning was as follows. Consider an internal node n that is in T_1 and is then pruned (without its parent being pruned), so that it is not in T_2. When n was originally put into T_1, a large number of possible yes-no questions were considered to fill this node: call them Q_A, Q_B, Q_C, and so on. Ultimately all but one of these questions was rejected during this expansion phase: let us call the one that was chosen Q_A. Now, suppose that though n was pruned from T_2, during the next expansion phase on S another internal node grows where n used to be. Obviously, the question contained here in T_3 cannot be Q_A, since Q_A did a worse job of classifying the data in S than its parent did (that's why it was pruned). We know that the same set of questions as before were considered for this position (since the parent is the same) - Q_A, Q_B, and so on - but this time another question was chosen. Let's call the question occupying node n in T_3 Q_B. Now, my reasoning was that when we prune T_3 on F, we know that Q_B was already rejected once before on this same data, during the expansion phase that led to the choice of Q_A. Therefore, node n, which contains Q_B, must be pruned from T_3.

This is not a watertight case for saying that T_2 and T_4 must be identical, since it only concerns one type of node. However, it was enough to make me feel uneasy about the algorithm. The error in the argument lies in the confusion between rejecting a possible question during the expansion phase and rejecting it during the pruning phase. During expansion, the question that most reduces impurity is selected at each node. Many of the questions that are not chosen may be capable of reducing misclassification (they may even be better at this than the question chosen). Thus, in the previous paragraph, when Q_A was originally chosen to fill n over Q_B and the others, Q_B may still be an improvement over its ancestors (in terms of classifying the data in F). When the pruning algorithm later considers the effect of removing Q_B, it may therefore decide to keep Q_B because it reduces misclassification of S, even though it is not the best possible question for S in that position

from the point of view of reducing impurity (Q_A fits that description). In other words, the pruning algorithm only rejects a question if it does not reduce misclassification, whereas the expansion algorithm rejects all but the question that reduces impurity the most. Therefore, a question that was rejected earlier by the expansion algorithm on a set of data may reappear on the other set and be retained by the pruning algorithm on the original set.

Thus, as one might expect, the final tree is a compromise between the trees that would be locally optimal for the two data sets F and S. Each question in the final tree will be locally optimal for one data set, and be acceptable for the other data set - in the sense that it does not increase misclassification.

5.4 Set-Membership Questions

When one seeks to apply classification tree methodology to a problem, one frequently has an obvious set of possible "primitive" yes-no questions which one might like to combine to get "compound" questions. For instance, suppose that each of the data items consists of a vector of observations, each taking on one value in a discrete set of integer values; thus, each item is a vector $\mathbf{X} =< x_1, ..., x_m >$. Typical primitive questions would then be "does $x_3 = 5$?" and "does $x_6 = 1$?"

In such a situation, one might be content with letting the tree-growing algorithm pick good primitive questions for each node. However, it may be that data items with $x_3 = 2$, $x_3 = 5$ and $x_3 = 6$ tend to be similar; if so, a question that groups them together makes more economical use of the training data and will therefore yield a better tree. Hence, one might wish to consider compound questions like "is x_3 in the set $\{2, 5, 6\}$?"

In the case of word sequences, different words may have similar meanings. In the ATIS application, the sentences "Show me taxi fares", "show me limousine fares", and "show me bus fares" should have exactly the same effect on the speech understanding system (the display of a ground transportation table), whereas the sentence "show me flight fares" should have a completely different effect. I was therefore interested in looking at algorithms capable of asking questions about a set of words in a given position - for instance, "does the sentence contain a word in the set $\{taxi, limousine, bus\}$?"

The problem with asking questions about sets of values of a variable is

that the number of possible questions goes up exponentially with the number of values the variable can assume. I.e., if x_i can take N discrete values, the number of possible "set-membership" questions regarding x_i is exponential in N. In [Bre84], Breiman *et al* give an exact, linear-time algorithm for finding the optimal set-membership question in the case where there are only two classes. For more than two classes, no such algorithm has been found. Therefore, if there are more than two classes one must employ a heuristic.

Two such heuristics for finding a good set-membership question in polynomial time are found in [Cho91] and [Nad91]. The first reference describes an iterative, K-means-like clustering algorithm; the second exploits the Breiman *et al* two-class algorithm (by grouping, and by iteratively reversing the roles of variable values and classes). The first algorithm was inapplicable to the problem of classifying word sequences because it assumed that the outcomes being combined in a set-membership question are mutually exclusive. It would have had trouble generating a question like "does the sentence contain a word in the set $\{taxi, limousine, bus\}$?" because two or even three of the words in the set may occur in the **same** sentence. The second algorithm assumes a small number of possible values, corresponding to a small number of possible words - with a vocabulary of about 1000 words, it would have taken far too long to run.

However, I considered these two heuristics carefully, and even came up with modified versions that might have been applied to the word sequence problem I was considering, though they would have demanded considerable computation time. Ultimately, I decided on a simpler heuristic for generating set-membership questions that made more sense for this problem; it is described in the next chapter.

5.5 Applications of Classification Trees in Speech Processing

Keyword Classification Trees are an adaptation of classification trees to the requirements of the speech understanding task. Both the nature of the questions in KCTs and the application of classification trees to speech understanding are original. However, standard classification trees are employed at other levels of the speech hierarchy. The examples that follow will serve both

to illustrate the practical application of classification trees and to underline the unique aspects of KCTs.

5.5.1 Vector Quantization

Vector quantization is a technique that maps a stream of high rate data into a stream of relatively lower-rate digital data [GraWL]. The goal is to achieve a maximum amount of data compression and a minimum of distortion. The technique has been applied to many types of coding, especially image coding and speech coding. As was mentioned in Chapter 2, vector quantization is often applied in speech recognition systems to provide the input to HMMs.

Although the details of particular applications of vector quantization vary greatly, the basic idea is easy to grasp. Consider an input vector

$$\mathbf{X} = < x_1, ..., x_n >$$

that one wishes to transmit. Suppose that there exists a *vector quantization codebook* containing $\mathbf{M}$ n-dimensional vectors $\mathbf{C_i}$ of the same type as $\mathbf{X}$, and that there also exists a distance measure $d(\mathbf{X}, \mathbf{C_i})$ between such n-dimensional vectors. To transmit $\mathbf{X}$, find the closest vector $\mathbf{C_i}$ in the codebook to it, then send the index i of that vector across the channel instead of the vector itself. At the other end, a decoder which has access to the same codebook will be able to reconstitute the codebook vector chosen. If the indices of the $\mathbf{M}$ vectors in the codebook are the numbers $1, ..., M$ in binary, it is obvious that the message will require approximately $\log_2 \mathbf{M}$ bits.

The technique has the advantage that by increasing or decreasing the number M of vectors in the codebook, one can smoothly trade off fidelity of transmission against data compression. The more non-uniform the distribution of vectors in the observation space, the greater the potential for data compression - if the choice of codebook vectors is carried out cleverly. Thus, codebook design is crucial to the success of the technique.

A common procedure for designing the codebook is iterative and locally optimal [O'S87 pp. 315-316]. It works as follows:

1. Start with an initial codebook and a set of observation vectors. Calculate the average distance (often called the average "distortion") between an observation vector and the codebook vector that will represent it. If the average distance is small enough, stop.

2. If not, replace each codebook vector with the average of all observation vectors that mapped onto it, i.e. with their centroid.

3. Using the new codebook, return to step 1.

The choice of initial codebook has a strong effect on the performance of the final codebook. A simple approach is to pick $\mathbf{M}$ vectors at random from the training set.

Note that in general, the coding of observation vector $\mathbf{X}$ requires distance comparisons with $\mathbf{M}$ vectors $\mathbf{C_i}$. This is called "full codebook" search. We can speed up coding at the cost of increased distortion by employing "tree search" instead. Figure 5.7 shows a binary tree codebook. Here, an observation vector is first compared to vectors y_0 and y_1. If it is closer to y_0, it is then compared to y_{00} and y_{01}, and so on until it reaches one of the leaves. The coding of the vector is the bit string label for the leaf at which it ends up. Note that this procedure is not guaranteed to send an observation to the leaf node it is closest to, though it cannot get to a leaf node that is very distant from the optimal one. On the other hand, it only takes about $\log_2 \mathbf{M}$ comparisons.

There are several ways of obtaining tree-structured codebooks. One way is to begin with the $\mathbf{M}$ (a power of two) leaf vectors, which are clustered into pairs of adjacent vectors. The vectors at the next level are the centroids of these pairs. The process continues until there are two centroid vectors at a level, y_1 and y_2.

Alternatively, one may build a tree-structured codebook by calculating the centroid of all training vectors, taking this vector $\mathbf{V}$ as the root of the tree. One then splits the root by perturbing $\mathbf{V}$ slightly to obtain two distinct nearby vectors $\mathbf{V_0}$ and $\mathbf{V_1}$. About half the training vectors will be closer to $\mathbf{V_0}$, the rest closer to $\mathbf{V_1}$: let y_0 be the centroid of the first group, y_1 the centroid of the second group. Assign each training vector to the y_i it is closest to, and split each y_i by perturbing it and then finding two centroids for the vectors assigned to the y_i. These two new centroids will be y_{i0} and y_{i1}. Continue the process recursively until the codebook is large enough.

Makhoul *et al* [Mak85] were the first to introduce an unbalanced tree: instead of growing the tree a layer at a time, they always split the node (at any depth) that contributes most to overall distortion. To implement a fixed rate code, they simply number the leaf nodes in some convenient order (e.g.

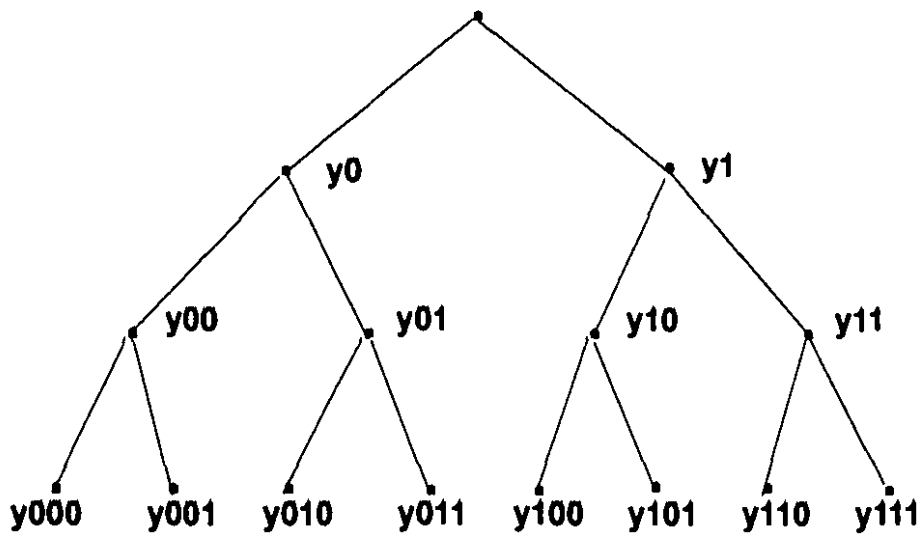


Figure 5.7: Binary Tree Codebook [O'S87, pg.317]

preorder). To maximize transmission efficiency, one should grow the tree until the number of leaves is a power of two.

The application of Breiman *et al* classification tree methodology to unbalanced tree-structured codebook design is now obvious. Two recent papers [Och91, Ris91] describe experiments along these lines. As the impurity measure, they use a function of distortion. In [Ris91] cross-validation pruning is carried out; the criterion λ for pruning is the ratio of potential increase in distortion to decrease in number of bits. Also in [Ris91], the authors made the interesting experimental observation that two-step lookahead during the growing phase yields performance close to that obtained by pruning. Another interesting recent paper on this topic is [Kia91].

5.5.2 Context-Dependent Phone Modeling

Ideally, one would construct HMMs for each word in the vocabulary. As the size of the vocabulary increases, it becomes more and more unreasonable to expect sufficient repetitions of each word in the training data. Hence, **subword modeling** is required for large-vocabulary speech recognition: the unit for which HMMs are trained is smaller than the word. An excellent introduction to this topic is [LeeWL].

Phoneme models seem an obvious choice; it would be easy to get enough data to train an HMM for each of the 50-odd English phonemes. Unfortunately, the acoustic realization of a phoneme - the phone - depends on surrounding phones, because our articulators cannot move instantaneously from one position to another. For instance, the 'l' in "lamp" is quite different from the 'l' in "pull" (in Welsh these are two different phonemes). Context-independent phone models average out these differences, yielding HMMs that perform poorly.

A popular compromise is to model **triphones**. That is, for a given phone, one builds an HMM for each combination of left and right context in which it can occur. Though there are roughly 50 English phonemes, the number of HMMs required is much less than 50^3 because many combinations are impossible. The actual number depends on whether only inter-word or also intra-word contexts are modeled.

Hon and Lee [Hon91] propose an interesting classification-tree-based algorithm for clustering together triphones, if there are insufficient data for training all triphone models. Figure 5.8 shows the resulting tree for the

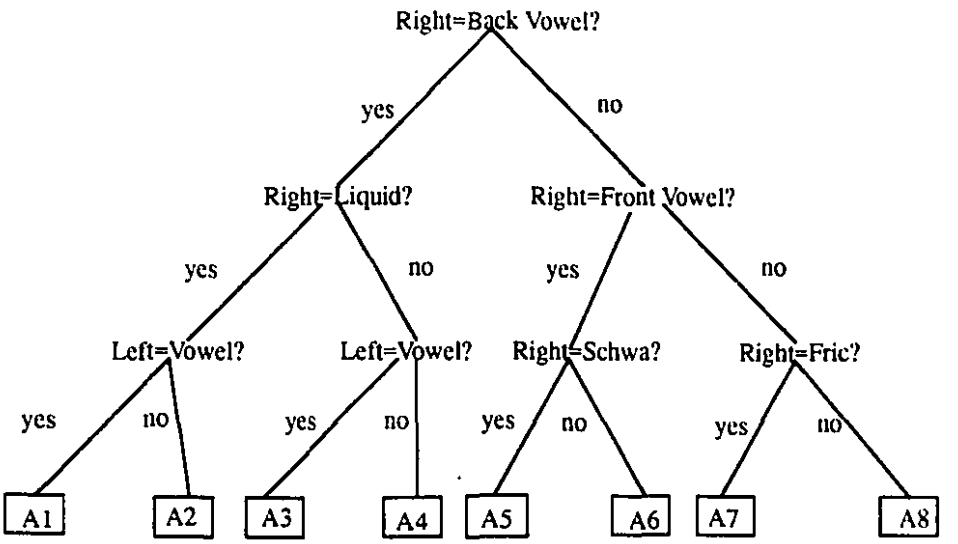


Figure 5.8: Decision Tree for Allophones of 'k' [Hon91, pg.260]

phone 'k' ("left" means the preceding phone, "right" the next phone). First, HMMs were trained for all possible triphones with 'k' in the middle. Once the tree had been generated from these, there were 8 HMMs for 'k': one for each of the leaf nodes shown. During recognition, whenever the probability that a phone is 'k' must be evaluated, the phone is shunted to the appropriate leaf in the tree and analyzed by the corresponding HMM. In an experiment described in [*ibid*], this approach improved performance from 6.5% error rate with the original large set of triphone HMMs to 5.4% error rate with the new, smaller set of phone HMMs.

To grow the tree, consider a root node containing all triphone HMMs and a set of possible questions about the context like those shown in the

figure. Each question will split the triphones into two sets a and b. There is a fast procedure for determining an HMM for each of a and b from the individual triphone models. Using these two new HMMs, calculate the amount of entropy reduction for each possible question and pick the question that maximizes the overall drop in entropy (entropy is the impurity criterion). Recurse until some stopping rule is triggered, then prune the tree on new data.

In an earlier section, I discussed the generation of "set-membership" questions from more primitive questions. The authors of [Hon91] employ an intriguing algorithm to generate what they call "compound" questions from "simple" questions. To pick the best compound question for a given node n, they grow a complete tree made up only of simple questions from n. Subsequently, they consider all possible Boolean combinations of the simple questions in the tree whose root is n and pick the best combination for n, discarding the tree of simple questions. This process is illustrated for a single node A in figure 5.9; provided the size of the simple tree grown at each node is very small, it is computationally feasible. Unfortunately, these authors never define or list the simple questions.

A group of researchers at IBM have taken an even bolder step: instead of working with triphones, they use a new set of context-dependent subword models **defined** by classification trees [Bah91]. Presumably, their reasoning was that there is nothing magic about the context provided by the immediately preceding and immediately following phone only - surely the current phone may sometimes be affected by even earlier and even later phones?

Consider the sequence $P_{-K}, ..., P_0, ..., P_K$, where P_0 is the current phone. The IBM researchers collected training data annotated in this manner (K was set to 5) and grew classification trees on it, one for each phone P_0. The set of possible questions was obtained by asking about one particular P_i at a time. Each question could ask whether P_i was a particular phone, or whether it belonged to a certain phonologically meaningful subset of all phones. For instance, the questions "is P_{-2} the phone 'a'?" and "is P_5 a voiced stop?" were among the questions considered. Questions asking about more than one P_i at a time were not allowed.

Training HMMs for each possible split during growth of the tree in order to calculate an HMM-based impurity criterion would have been computationally expensive. Therefore, a cheaper criterion based on the similarity of the outputs of the vector quantizers associated with the phones was em-

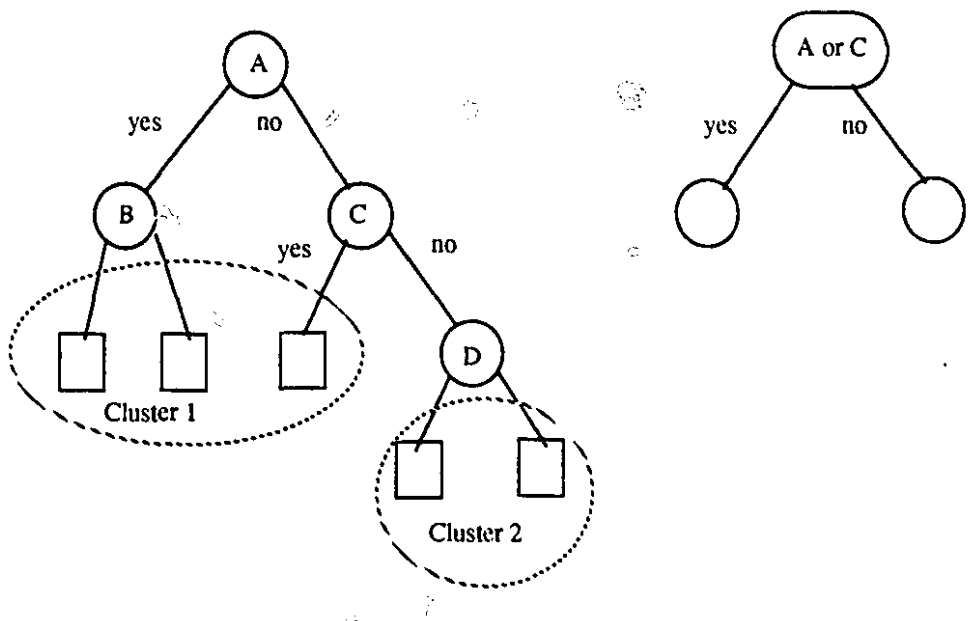


Figure 5.9: Clustering Simple Questions to form a Compound Question [Hon91, pg.260]

ployed instead. Once the tree has stopped growing, an HMM is grown at each leaf; the leaves are called "allophones". Some of the IBM experiments used a stopping criterion that prevented the tree getting very large, but did not prune the tree; others involved an oversized tree that was subsequently pruned.

Experiments with test data showed improvement over triphone models (which are obtained when $K = 1$). The best results were obtained for $K = 5$, i.e. when some of the questions in the tree go back as far as 5 phones in the past or go forwards as far as 5 phones in the future. The average error rate was 6.8% for $K = 1$ (triphone) and 5.9% for $K = 5$. In a slightly different set of experiments, the best average number of allophone models per phone was found to be about 45; of course, some phones may require more models than this and others fewer.

5.5.3 Language Modeling

Recall that if $\mathbf{y}$ represents an acoustic observation vector, and $\mathbf{w}$ a sequence of words, the task of a speech recognition system is to find $\mathbf{w}$ such that $P(\mathbf{w}|\mathbf{y})$ is maximal. By Bayes's rule, we have

$$P(\mathbf{w}|\mathbf{y}) = P(\mathbf{w})P(\mathbf{y}|\mathbf{w})/P(\mathbf{y}).$$

$P(\mathbf{y})$ can be ignored, since it is constant at a given time. Thus, the system seeks to find $\mathbf{w}$ maximizing $P(\mathbf{w})P(\mathbf{y}|\mathbf{w})$; the calculation of $P(\mathbf{w})$ is the job of the *language model.*

$P(\mathbf{w})$ is easily calculated if for each word w_i in the sequence, we know the probability of w_i given all preceding words: $P(w_i|w_0, ..., w_{i-1})$. Among the most popular formulae for estimating this probability are the *bigram* estimate $f(w_i|w_{i-1})$ and the *trigram* estimate $f(w_i|w_{i-2}, w_{i-1})$ where the frequencies $f()$ are obtained from a large training corpus. In theory, one might also wish to consider 4-gram, 5-gram, and arbitrary N-gram models - in practice, N-gram models with $N > 3$ require such huge training corpora that they are inapplicable.

In [BahWLb] researchers at IBM argue that the bigram and trigram classes embody naive definitions of equivalence classes. These models ignore all words prior to the two most recent. In reality, some word sequences ending with the same pair of words may behave quite differently, while other

word sequences ending with a different pair may be functionally equivalent. To remedy this, they propose a classification-tree-based model that allows $P(w_i)$ to be conditioned on much less recent words. The new tree-based model is no more expensive to apply during recognition than the trigram model, but does require much more computation to develop.

The IBM work on the tree-based model is reminiscent of the IBM work on context-dependent phone modeling; however, the context here extends into the past only, and not into the future. The experiments carried out at IBM attempt to predict the 21st word in a sequence, given the preceding 20 words. Let w_{20} denote the most recent word, w_{19} the second most recent word, and so on. As in the phone modeling work, potential questions concerning the w_js ($j < 21$) may be simple or complex. Simple questions can ask whether a given w_j is one of the words in a vocabulary, or whether it belongs to a set of predefined classes supplied to the algorithm by the researchers. Classes may overlap. For instance, since the class of nouns and the class of months are defined, the simple questions "is w_{17} a noun?", "is w_{17} a month?" and "is w_{17} the word 'July'?" will all be considered.

The impurity criterion for choosing questions is entropy. To obtain a compound question at each node in the growing tree from the set of simple questions, the IBM researchers employ a peculiar data structure they call a *pylon*. This is shown in Figure 5.10. Suppose we are at a given node n of the growing tree. First, pick the simple question yielding the greatest reduction in entropy; this splits the data into two groups. Now pick a simple question that will split the YES group, with members of this group that yielded "no" being added back into the NO group, in a way that yields the maximum drop in entropy over all data in the node. Next, split the NO group, adding the "yes" children back into the YES group. Continue until no further drop in entropy can be obtained; the simple questions chosen may involve different w_js. This process generates a subset of the Boolean expressions over the simple questions.

A huge training corpus of about 29 million words was employed to grow and prune a tree of compound questions, which was not allowed to have more than 10,000 leaves. The vocabulary size was 5000 words. This tree was tested on about 1 million words of data. The perplexity of the tree-based model was not much smaller than that obtained with the trigram model: the former was 90.7, the latter 94.9. However, the tree-based model was less likely to assign a very low probability to certain words in the training text. Since this

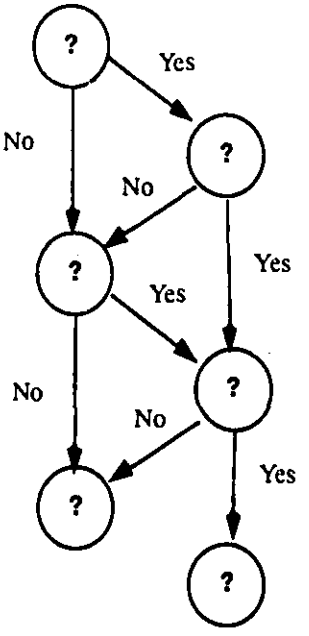


Figure 5.10: Example of a Pylon [BahWLb, pg.508]

is a situation likely to result in recognition errors, the drop in perplexity may underestimate the improvement in recognition performance to be expected from use of the tree-based model. Furthermore, an interpolation of the tree-based model and the trigram model yielded much better perplexity than either alone: 82.5.

The paper also contains some interesting statistics on the pylon depths and how often each w_j formed part of a question. Roughly speaking, about half the questions in the final tree were simple questions (pylon depth is 1), half of the rest contained two simple questions, and so on. Contrary to what one might expect, w_{20} - the most recent word - was **not** the word most often asked about; w_{19} - the second most recent word - was. After w_{19}, the most popular word for questions was w_{18}; only then came w_{20}. Next came w_{17}, and after that usage dropped off fairly rapidly. These results, along with the perplexity results, suggest that the tree-based language model incorporates quite different information from the trigram model, and hence forms a useful complement to it.

Chapter 6

Building Keyword Classification Trees

6.1 Introduction

In Chapter 3, we looked at the linguistic analyzer of several speech understanding systems. We saw that most of these systems are evolving in the same direction: though they once relied exclusively on syntactic parsing, they now rely on **robust matching** to handle some or all utterances at the global, sentence level. The robust matcher tries to fill slots in a frame without attempting a sentence-level parse, and skips over words or phrases that do not help it to fill a slot or to decide on the identity of the current frame. Syntactic parsing is required locally, to identify the slot-filling phrases and convert them to standard representation.

The work done by the linguistic analyzer in translating an utterance to a frame-like conceptual representation can thus be divided into three parts:

1. Parsing slot-filling phrases - this is the task of local syntactic parsers.
2. Deciding on the identity of the frame - this is part of the task of the robust matcher.
3. Assigning each slot-filling phrase to the appropriate slot - this is the rest of the robust matcher's task.

The robust matcher requires a large set of rules that tell it how to identify the frame or frames referred to by the current utterance, and how to match slot-fillers to slots. In all the systems I know of that employ a robust matcher, these semantic rules are handcoded by the system designers. The AT&T system does learn rules for segmenting a sentence into concepts from training data; however, it does not contain a robust matcher in the sense described above.

The assumption underlying the work described in this thesis is that effective semantic rules can be learned from training data, and that these rules can afford to ignore many of the words in an utterance. Keyword Classification Trees (KCTs) were devised for the purpose of learning such semantic rules; they are the building blocks of the linguistic analyzer described in Chapter 8.

KCTs have the following properties:

- They are a tool which can be applied whenever one wishes to classify strings or substrings. To apply them to a particular problem involving strings, one must formulate it as a classification problem.

- To train a KCT, one must supply it with a corpus of classified strings or substrings.

- The rules found in the nodes of a KCT refer to the symbols in the strings. These may be words, parts of speech (POSs), or higher-level syntactic structures like noun phrases (NPs) or prepositional phrases (PPs). We will assume that all possible symbols are listed in a lexicon. For convenience, though we will be considering strings made up of a mixture of words, NPs, and PPs, we will refer to the symbols referred to in the nodes of a KCT as its "keywords". The keywords are selected from the lexicon by the KCT-growing algorithm, rather than by the programmer, and normally constitute a small fraction of the lexicon.

- The questions in the nodes of a KCT concern regular expressions made up of keywords and gaps, where a gap is defined as an unspecified substring of length at least one. If an incoming string matches the expression in a node, it is sent to the node's YES child; if not, it is sent to the NO child. As classification proceeds, gaps are progressively filled in.

- There are two kinds of KCT, *single-symbol* KCTs and *set-membership* KCTS. The questions in a single-symbol KCT contain regular expressions made up of individual symbols; set-membership KCTs contain regular expressions made up of sets of individual symbols. The distinction between the two question types somewhat resembles that made by IBM researchers between "simple" and "compound" questions, in the work described at the end of Chapter 5.

As noted above, KCTs can be employed either to classify strings, or to classify substrings. In the first case, each data item $\mathbf{W_i}$ is a sequence of words:

$$\mathbf{W_i} = < w_1, ..., w_n > .$$

The training corpus consists of a large number of such $\mathbf{W_i}$s, each with a known class $c(\mathbf{W_i})$. The KCT grown on this corpus assigns a class $\hat{c}(\mathbf{W_j})$ to new word sequences $\mathbf{W_j}$. This case will be discussed extensively in the sections that follow; it often arises when one wishes to derive semantic rules for selecting the correct frame for an utterance.

In the second case, we wish to classify only some parts of a string: inside each $\mathbf{W_j}$, we have $N(j)$ substrings $S_1, ..., S_{N(j)}$ that are to be assigned classes $\hat{c}(S_1), ..., \hat{c}(S_{N(j)})$. Here, the training corpus consists of word sequences whose substrings are labelled. An example from ATIS will illustrate this case, which often arises when one wishes to derive rules for assigning slot-filling phrases of a given type to the correct slot of the same type.

Suppose that the local parsing step replaces all city names with the generic symbol "CIT". For the ATIS task, it is important to label city names that occur in the word sequence hypothesis as "ORI" (origin of a flight), "DEST" (destination of a flight), "STOP" (stopover location), or "SCRAP" (irrelevant). The original word sequence hypothesis might be: "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".

If this sentence is used for training, it should be labelled as follows: "show me flights from CIT⇐SCRAP no sorry from CIT⇐ORI to CIT⇐DEST stopping over in CIT⇐STOP". If the sentence is a new one presented to a KCT, we wish the KCT to label each "CIT" in exactly this way. Fortunately, the

algorithms used to grow KCTs that classify entire strings require only minor modification to grow KCTs that classify parts of strings. This will be demonstrated at the end of the chapter.

The properties of the KCT-based robust matcher are partly determined by the properties of KCTs. In particular:

- The work done by the robust matcher on word sequence hypotheses is broken up into subproblems, in such a way that many of these subproblems can be defined as classification problems handled by KCTs. The only part of the ATIS task not handled by KCTs is an initial parse which identifies slot-filling phrases such as city names, dates, times, flight numbers and so on (recall that the AT&T group also came to the conclusion that this type of specialised local parsing was best handled by hand-written rules, because of the paucity of training data). Thus the robust matcher is basically a collection of KCTs, each handling a different subproblem, together with the mechanisms for invoking them in the correct order.

- Since KCTs ask about only a small fraction of symbols in the lexicon, the KCT-based robust matcher ignores many of the words in an utterance. Other robust matchers also ignore some words, but the KCT-growing algorithms find close to the smallest possible number of keywords required for semantic rules. Recognition errors in non-keywords do not hinder generation of the correct conceptual representation - consequently, the KCT-based matcher is very tolerant of recognition errors.

A full description of the KCT-based semantic matcher for the ATIS task will be found in Chapter 8; the current chapter is concerned with the KCTs themselves.

6.2 Single-Symbol KCTs

6.2.1 The Basic Algorithm

Recall that to grow classification trees, one must supply three elements:

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.

The original aspect of KCTs is the way in which the set of possible questions is generated. To choose a question from this set, we use the Gini criterion as described in Chapter 5, section 2; to prevent over-trained trees, we use the iterative expansion-pruning algorithm described in Chapter 5, section 3.2.

Each node of the growing single-symbol KCT is associated with a regular expression called the *known structure* consisting of *symbols* and *gaps*; the set of possible questions is generated by manipulating each of the gaps. A gap is an unknown sequence of symbols, of length at least one - it is symbolized by +. The known structure for the root of the KCT is always $< + >$, where $<$ and $>$ stand for the beginning and end of the string. respectively. This implies that strings entering the root must always have length at least one (empty strings are not permitted).

Consider the four regular expressions generated from a particular gap + in the known structure and a given item w_i in the lexicon as follows:

1. The regular expression obtained by replacing this + in the known structure by w_i;

2. The regular expression obtained by replacing this + in the known structure by w_i+;

3. The regular expression obtained by replacing this + in the known structure by $+w_i$;

4. The regular expression obtained by replacing this + in the known structure 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?" For instance, the expression $< w_3 >$ matches only the sequence consisting of the single symbol w_3, the expression $< w_3+ >$ matches sequences of length at least two beginning with w_3, the expression $< +w_3 >$ matches sequences of length at

least two ending with w_3, and the expression $< +w_3+ >$ matches sequences containing a w_3 that is neither the first nor the last symbol. If there are L symbols in the lexicon, we generate $4 * L$ questions by allowing w_i to be any of them. In addition to these questions, we consider all reasonable questions about the total length of the string, of the form "is length $< n$?"; these have turned out to be of little practical importance, as they are hardly ever chosen. From all these questions, the KCT-growing algorithm selects the one which achieves the best split of the labelled sequences in the training data, according to the minimal-impurity Gini criterion.

As the tree grows, known structures get longer. The known structure for the YES child of a node is identical to the expression found in the question of its parent, while the known structure for the NO child is identical to the known structure of its parent. For instance, if the question "does the sequence match $< +w_8+ >$?" is selected to fill the root (which has known structure $< + >$) the known structure for the root's YES child is $< +w_8+ >$, and the known structure for the root's NO child is $< + >$. New questions are generated by applying the four gap operations to each + individually. For instance, if the known structure is $< +w_8+ >$, questions involving the expressions $< +w_i + w_8+ >$ and $< +w_8 + w_j+ >$ will be generated, but not questions involving $< +w_i + w_8 + w_j+ >$ (which would require operating on both gaps in the known structure simultaneously).

Because we are using the expansion-pruning algorithm, we employ tolerant stopping rules that encourage growth of a large tree. A node is declared to be a leaf node when no further split is possible. This may happen for three reasons:

1. All training items in the node belong to the same category - this includes the case where only one training item ends up in the node; **OR**
2. There are no gaps + in the known structure for the node, so no questions can be asked; **OR**
3. There are questions that can be asked, but none of them give a split that reduces the Gini impurity.

For a detailed description of the implementation of this basic algorithm for growing single-symbol KCTs, see the Appendix. The next subsection gives an examples of a KCT grown from real data.

6.2.2 Preliminary Experiments with Single-Symbol KCTs

Preliminary experiments were carried out on the November 1990 release of ATIS data, before work began on the KCT-based robust matcher trained on 1992 ATIS data (as described in Chapter 8). The preliminary experiments were of two kinds:

1. Experiments involving transcripts of utterances;
2. Experiments involving the most probable word sequence hypotheses generated by the December 1991 version of the CRIM speech recognition system from the 1990 ATIS recorded speech. The CRIM system yields the N-best word sequence hypotheses, but only the top hypothesis for each utterance was employed for the experiments.

With these data, we studied two problems:

1. Determining the general topic of a request for information;
2. For questions involving fares or flights that only mention two city names, determine which is the origin and which the destination.

Unfortunately, the second problem proved unsatisfactory: not because it was too hard, but because it was too easy! It happens that in the November 1990 data, the origin of a flight is almost always mentioned before the destination. Although only a clumsy version of the algorithm for growing KCTs that label substrings had been developed at this time, the KCT grown for this problem nevertheless learned this simple rule. It labelled the first city name ORI, the second DEST.

The first problem was much more interesting. SQL queries have three parts:

1. a SELECT clause that specifies what attributes to retrieve;
2. a FROM clause that indicates which tables contain these attributes;
3. a WHERE clause that specifies conditions on the rows.

For the 1990 ATIS data, it was convenient to assign queries to one of six topics or frames, based on the tables mentioned in the FROM clause (changes in the data and in the evaluation have subsequently diminished the usefulness of this six-fold classification). The topics were numbered as follows: **1. AIRCRAFT 2. FARE 3. FLIGHT 4. FLIGHT_FARE 5. GROUND 6. MEANING.**

The task of a KCT was to determine which of these six frames a sentence from the transcript data or the speech recognizer belonged to. Two KCTs capable of carrying out this task were grown: one trained on transcripts labelled with the correct topic, the other trained on the top word sequence hypotheses output by the CRIM recognizer (also labelled with the correct topic). The true labels for both were determined by glancing at the corresponding SQL commands provided by DARPA. Both were tested on new data of the same type - i.e., the transcript-grown KCT was tested on transcripts, and the hypothesis-grown KCT on hypotheses.

Figure 6.1 shows the upper part of the KCT grown on transcripts. The KCT grown on hypotheses from the recognizer was rather similar, except near the leaves. This was to be expected; recognition errors are more likely to affect the choice of question in nodes that receive a small number of training data items. Note that keywords may occur next to each other, as in the question involving the expression

$$< +transportation\ codes >,$$

or apart. None of the questions in this KCT involve categories, such as city names, dates, or flight numbers. This is because the training sentences were not preparsed. If they had been, expressions like $+CIT+$ might have appeared in the nodes of the KCT.

Perhaps the most important aspect of the single-symbol KCT grown to identify the topic is not evident from figure 6.1. There were 41 nodes in this tree; of these, 28 occurred in the subtree indicated with a triangle. This points to a problem experienced with most single-symbol KCTs: at any level of such KCTs, the NO subtrees tend to contain more nodes than their YES siblings.

Again, this might have been predicted. There are many ways of saying the same thing with different words in English. Thus, if we ask whether a particular keyword is present in sentences of a given type, there will always

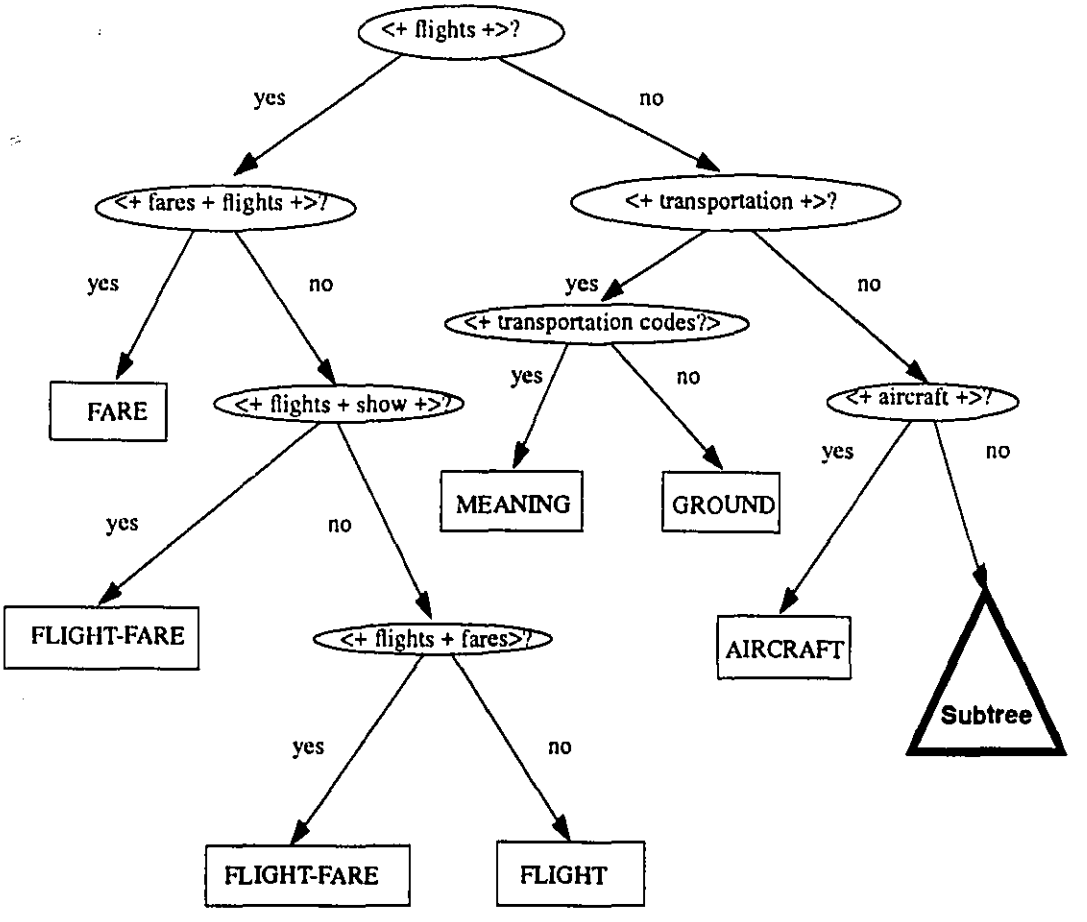


Figure 6.1: Single-Symbol KCT Grown on ATIS 1990 Transcripts

be many sentences for which the answer is "no". Most questions in a single-symbol KCT will therefore shunt more training data items into their NO child than into their YES child, giving the NO subtree more chance to grow before it runs out of data items. This asymmetrical quality implies that single-symbol KCTs do not use the training data efficiently, since word sequences that belong together are forced apart. Set-membership KCTs represent an attempt to overcome this problem.

6.3 Set-Membership KCTs

Above, we mentioned the possibility of preparsing sentences in the training corpus for a single-symbol KCT, thus allowing expressions involving categories like "CIT" and "DATE" to appear in questions. These categories are defined by the system designer. A more radical approach is to allow the KCT-growing algorithms to generate questions about sets of words at a given position, in effect allowing system-defined categories. This is the idea behind set-membership KCTs. Figure 6.2 shows a set-membership KCT grown on the same data as the single-symbol KCT shown in figure 6.1.

Note that the questions involve many of the same keywords as appeared in the figure 6.1 single-symbol KCT. The set of words $\{flights, flight, nonstops, how\}$ in the root node is of particular interest, because in the context of ATIS the first three words act almost as synonyms. For instance, "show me a flight to Boston", "show me flights to Boston", and "show me nonstops to Boston" all belong in the **FLIGHT** category. Experiments with set-membership KCTs often generate sets of words that are quasi-synonyms. So far, the most interesting example I have seen is the set $\{what, explain, describe, define\}$, in a question that was generated by the system but rejected in favour of a question at another position in the known structure that was slightly better at lowering Gini impurity.

Many of the questions in the nodes of a set-membership KCT will insert a new set into the gaps + in the known structure, similar to what happened to individual symbols in the single-symbol KCT. As figure 6.2 shows, this is not the only possible type of question. Consider the node whose known structure is

$$< +\{for, ticket, aircraft, cost\} + \{flights, flight, nonstops, how\} + >;$$

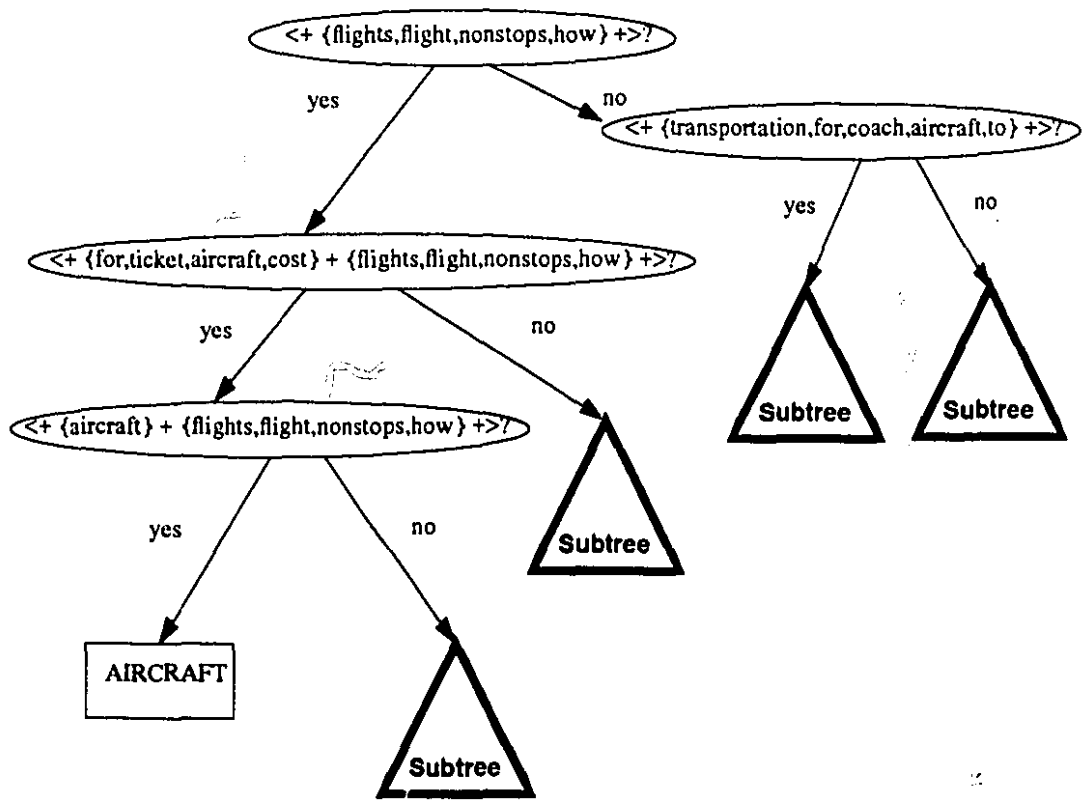


Figure 6.2: Set-Membership KCT Grown on ATIS 1990 Transcripts

the question chosen for this node asks whether sequences match the expression

$$< +\{aircraft\} + \{flights, flight, nonstops, how\} + > .$$

Thus, the question in a node of a set-membership KCT may partition a set in the known structure.

Set-membership KCTs are somewhat trickier than figure 6.2 indicates. Consider the sentence: "please tell me how much the ticket for a flight to Boston costs". Clearly, this matches the pattern shown at the root of the KCT, since the middle of the sentence contains the words "how", and "flights", both in the set $\{flights, flight, nonstops, how\}$. The sentence passes to the root's YES child. Here, it is unclear whether the sentence matches the pattern

$$< +\{for, ticket, aircraft, cost\} + \{flights, flight, nonstops, how\} + > .$$

The word "ticket" in the first set is left of the word "how" in the second set, but right of the word "flight" in the second set.

The solution to this problem is that we must clearly define the left and right boundaries of a set, in order for subsequent questions to be well-defined. If we do so, we may even ask questions that split a set into a substring in the known structure. For instance, the known structure

$$< +\{flights, flight, nonstops, how\} + >$$

might yield a question about the expression

$$< +\{how\} + \{flights, flight, nonstops\} + > .$$

The details of defining set boundaries and questions that depend on them are not difficult, but complicated and tedious; they are given in the Appendix.

As with single-symbol KCTs, questions in the set-membership KCT are generated at gaps + in the known structure. The most important component of the algorithm for growing set-membership KCTs is the function that generates sets of words at a given +. If S stands for a set of symbols, the questions considered at each + are as follows:

1. The regular expression obtained by replacing this + in the known structure by S;

2. The regular expression obtained by replacing this + in the known structure by $S+$;

3. The regular expression obtained by replacing this + in the known structure by $+S$;

4. The regular expression obtained by replacing this + in the known structure by $+S+$.

A variety of methods for generating S was considered, as mentioned in Chapter 5. Ultimately, the simplest possible method was adopted. Consider the problem of choosing a sports team from a large group of temperamental athletes, some of whom hate each other. One heuristic would be to pick the best athlete, then the best remaining athlete who gets on with the first one, then the best remaining athlete who gets on with the two already chosen, and so on. The heuristic is not infallible: if an athlete chosen early on vetoes most of the best remaining athletes, one might obtain a better team by leaving him or her out. However, the heuristic does ensure that the team chosen commands the services of the best athlete. The method for generating S is based on this heuristic.

For each + in the known structure, obtain four initial set-membership questions as follows:

1. Find the best question obtained by replacing this + in the known structure by S, where S contains a single item w_i from the lexicon - i.e., $S = \{w_i\}$;

2. Find the best question obtained by replacing this + in the known structure by $S+$, where S contains a single item w_i from the lexicon;

3. Find the best question obtained by replacing this + in the known structure by $+S$, where S contains a single item w_i from the lexicon;

4. Find the best question obtained by replacing this + in the known structure by $+S+$, where S contains a single item w_i from the lexicon.

Thus, the best single-symbol question for each of the four gap operations is obtained, and used to initialize four different sets S. Next, the algorithm goes through the lexicon to find the symbol that will most diminish Gini impurity

when it is added to each set S, and adds it to the set. The algorithm now cycles through the lexicon again to add to the set the item that further reduces impurity the most. Eventually, the addition of further symbols to a set will leave the impurity the same or actually increase it; at this point the set S for one of the set-membership questions has been found. When all sets are complete, the set-membership question which most reduces the node impurity is chosen.

Figure 6.3 illustrates this method. Here the vocabulary V consists of animal names, one for each letter of the alphabet, and the known structure for the node is $< +horse+ >$. Thus eight initial single-symbol questions will be generated, four for each of the two "+" in the known structure. The diagram shows the expansion of two of the eight initial questions. Note that the best set-membership question may be generated from a single-symbol question that was not the best of the single-symbol questions.

This heuristic for question generation is not guaranteed to yield the optimal set-membership question - but neither are any of the more complicated heuristics described in Chapter 5. It has the merit of being fast, easy to implement, and is guaranteed to bring about at least as big an impurity reduction as the best single-symbol question, because the best single-symbol question is always one of the set-membership questions considered.

Initial experiments showed that one modification to the heuristic was necessary. If each set S is allowed to grow until no more words that reduce impurity when added to S can be found, some words may be added to S on the basis of a single example. In a sense, the set S becomes "overtrained" to the training data. This situation was resolved by a "stopping rule" that specified that a set is only allowed to incorporate new words that reduce impurity and do so by moving at least N training data items to a more appropriate child node, where N is a number greater than 1 (usually set to 2 or 3).

6.4 Classifying Substrings

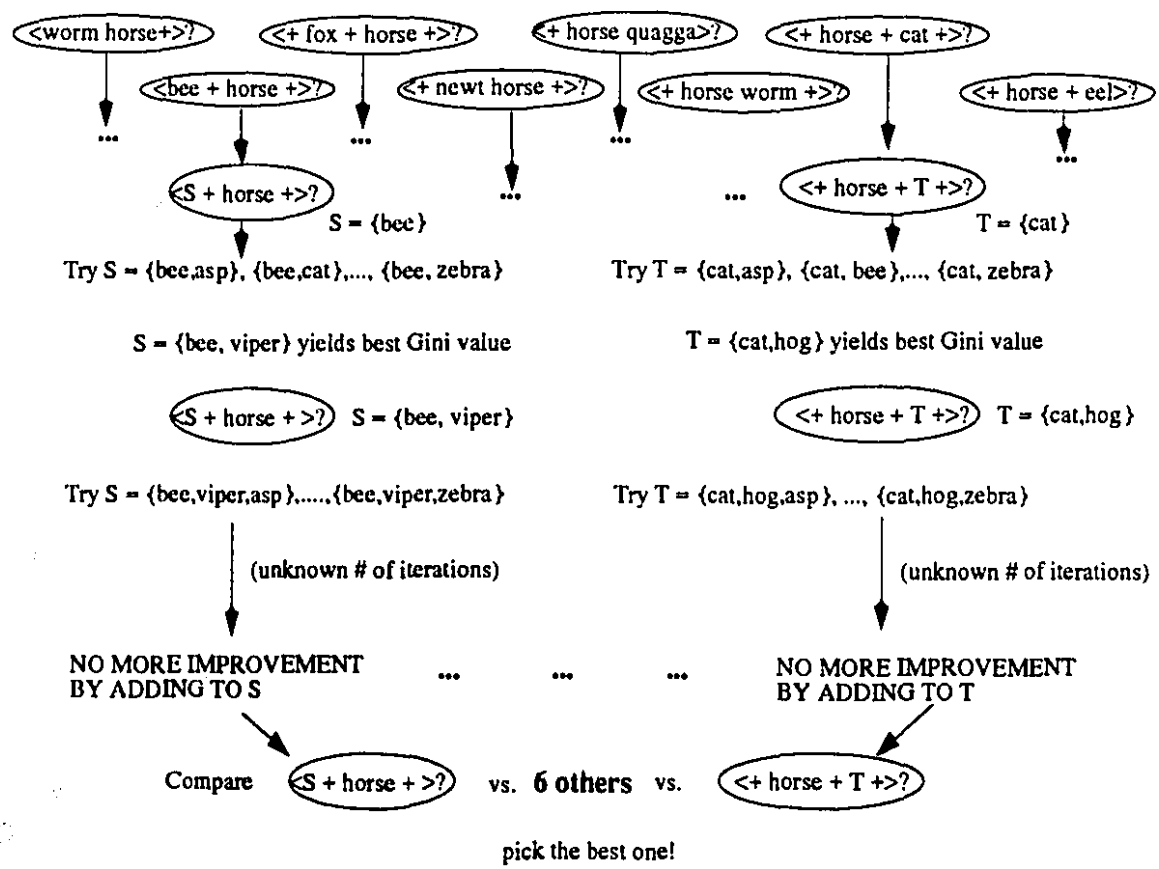
We will now see how to adapt the algorithms for growing KCTs (either single-symbol or set-membership) to the task of classifying substrings. The example given earlier was that of dealing with sentences that emerge from the local parsing phase in a form like "show me flights from CIT no sorry

Example

Vocabulary V= {asp, bee, cat, ..., zebra}

Known Structure = <+ horse +>.

Best Single-Symbol Questions (first 2 rows below):



Note: starting with known structure <+ horse +>, obtain the best single-symbol question for each of 8 possible operations on known structure, i.e. 8 single-symbol questions. From these, obtain 8 set-membership questions - pick the one yielding greatest impurity drop. For space reasons, only show generation of 2 of the 8 set-membership questions.

Note: best of the 8 single-symbol questions does not usually yield best of 8 set-membership ones!

Figure 6.3: Growing a Set-Membership KCT

from CIT to CIT stopping over in CIT". To generate the conceptual representation, one requires each occurrence of the category CIT to be labelled as follows: "show me flights from CIT⇐SCRAP no sorry from CIT⇐ORI to CIT⇐DEST stopping over in CIT⇐STOP". Once this has been done, the second, third and fourth city names mentioned in the original sentence are retrieved and assigned to the ORIGIN, DESTINATION, and STOPOVER slots respectively.

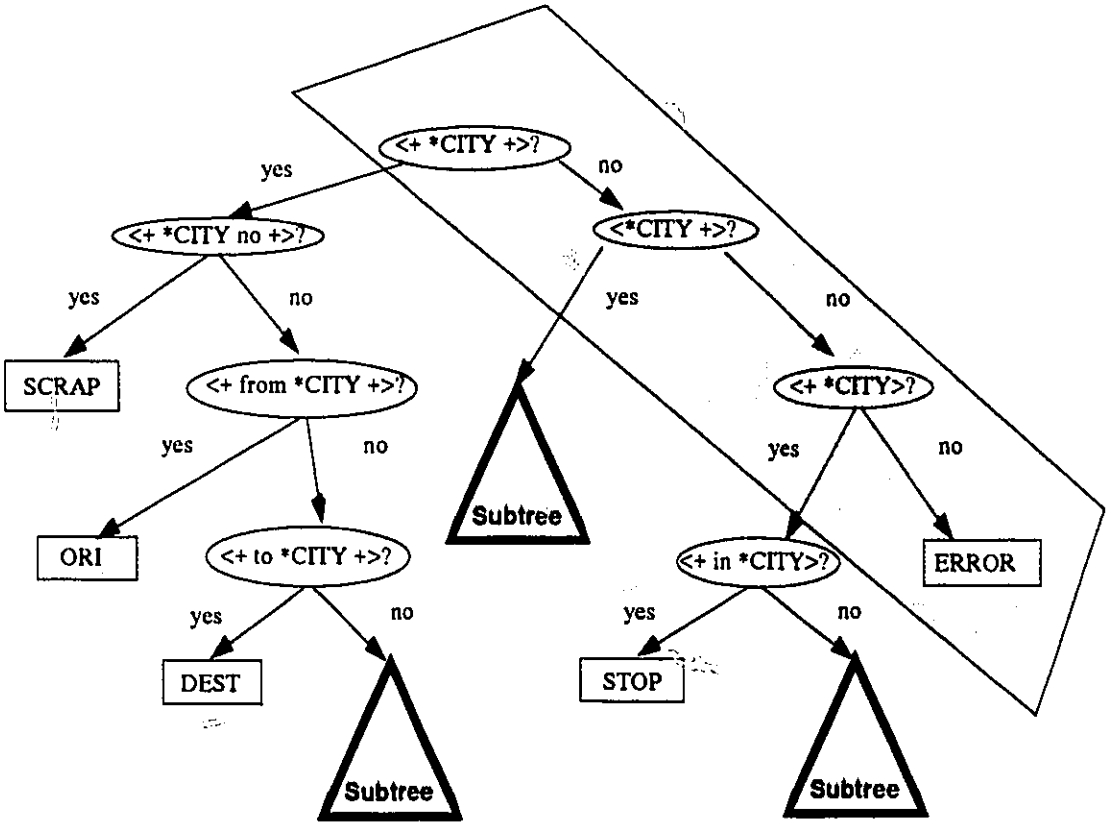
The crucial insight here is to submit the same sentence to a KCT as many times as there are substrings to be classified. Each time, the substring being classified is marked with a special symbol such as '*'. Figure 6.4 shows a single-symbol KCT for classifying CIT substrings.

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 KCT. This sentence matches first the expression $< +*CIT+ >$ in the root and then the pattern $< +*CIT\ no+ >$, so the first CIT is labelled "SCRAP". To classify the second city, we submit "show me flights from CIT no sorry from *CIT to CIT stopping over in CIT" to the KCT. This time, the sentence matches the pattern $< +from\ *CIT+ >$, so the second CIT is labelled "ORI". To classify the third and fourth CIT in the sentence, the sentence is resubmitted twice more after marking the current CIT with '*'. This single-symbol KCT correctly labels all four appearances of CIT in the sentence.

How can a KCT for classifying substrings be grown from training data? Note that certain nodes in figure 6.4, including the root, are shaded. These nodes are the **compulsory backbone** of any KCT for classifying CIT substrings. The growth of such a KCT begins from the compulsory backbone already specified by the system designer; otherwise, the KCT-growing algorithm is exactly the same.

As one might expect, the training data are sentences whose substrings have been correctly labelled. For instance, a typical training sentence for the CIT task might be "list fares for flights out of CIT⇐ORI to CIT⇐DEST". To convert this to a form useful for growing the tree, a preprocessor makes two copies of this: one is "list fares for flights out of *CIT to CIT" and is labelled "ORI", the other is "list fares for flights out of CIT to *CIT" and is labelled "DEST". These two sentences are treated as independent items for training. Thus, a sentence in which "CIT" appears N times yields N training data items.

This approach allows the KCT to classify one occurrence of CIT at a



Note: shaded area is compulsory backbone - put in place before tree-growing begins.

Figure 6.4: KCT for Classifying CITY Substrings

time: the one marked with a '*'. "CIT" without the '*' is treated just like a normal word in the lexicon, and may occur in a question. For instance, the expression $< +CIT+*CIT+ >$ would be matched by a sentence like "give me all CIT flights from *CIT today". The approach allows one to employ algorithms for growing either single-symbol or set-membership KCTs exactly as described above, starting with the compulsory part and utilizing a training corpus.

The only drawback of the approach is that because each substring being classified (the substring "CIT" in this case) is classified independently, two or more of them could be assigned the same label. Heuristics to deal with this problem can easily be devised. For instance, probabilities for the leaves of the KCT could be estimated from a disjoint subset of the training data - in case of a conflict, assign labels in a way that maximizes the overall probability. However, it is probably wiser to view assignment conflicts as a sign that a situation rarely or never encountered in the training corpus has been encountered, and that substring classification will consequently be unreliable.

In the version of substring KCTs used in the Nov92 ATIS benchmarks, a new type of question was considered, in addition to the types described earlier. The new type asked about the ordinality of a substring with respect to other substrings belonging to the same category. Thus, questions like: "is this the second CIT?" and "is this the third last CIT?" could be asked. This question type proved very useful. For instance, if two cities are mentioned, there is a good chance that the first will be the origin and the second the destination.

6.5 Related Work

6.5.1 Comparison with IBM Tree-Based Language Modeling

Though KCTs were devised to learn semantic rules, and the IBM technique described in Chapter 5 (section 5.5.3) was applied to language modeling, both employ classification trees to classify word sequences. How do the two approaches differ?

Recall that the IBM technique classifies the context within which a word

w_n occurs by asking questions about the identity of the preceding 20 words $w_{n-20}, ..., w_{n-1}$. For instance, a simple question in the IBM scheme could be "is w_{n-5} the word 'flight'?" Compound questions are Boolean combinations of simple questions. Thus, in the IBM technique questions refer to the position of a word in a sequence relative to a fixed location given by the current word. By contrast, when KCTs are employed to classify a complete word sequence, the only fixed initial positions are given by the beginning and the end of the sequence.

KCTs have been applied to two problems: classifying entire word sequences, and classifying subsequences within word sequences. The IBM technique has been applied only to the latter problem. It is not clear how it could be extended to the classification of an entire word sequence - one would need to specify a fixed location in the sequence with respect to which questions would be asked.

Consider the question "does the sequence match $< +flight\ time+ >$?" contained in the node of a single-symbol KCT. This question would yield "yes" on the following sentences (and an infinite number of others):

> Give me the flight time for Delta 105.
>
> What was the flight time again?
>
> The flight time for Delta 105 please.

Suppose the IBM technique was adapted to classify complete word sequences by letting the last word be w_n, the fixed position. Then the IBM question closest to this KCT question would be the horrendously complicated compound question: "Is ($w_{n-1} = time$ AND $w_{n-2} = flight$) OR ($w_{n-2} = time$ AND $w_{n-3} = flight$) OR ($w_{n-3} = time$ AND $w_{n-4} = flight$) OR ...?" This compound question will only be generated if each of the simple questions in it wins out over thousands of rivals, which is far from certain; furthermore, it requires much larger amounts of computation to generate than does the corresponding single-symbol KCT question. Note also that by fixing w_n at the end of the sentence, we lose the ability to ask about the beginning of the sentence (if it is more than 21 words long) whereas the KCT approach allows questions to focus on any part of indefinitely long sentences.

In exchange for its advantages, the KCT approach gives up the possibility of asking about precise positions within the word sequence. The questions in a KCT may refer to the first or last word in a sentence, and to a position

left or right of a previously identified keyword, but in general they cannot refer to the word at position i in the sequence (unless previous questions have specified the words present at positions 1 to i, or backwards from the last word at position n to i). As the name "Keyword Classification Tree" implies, the KCT approach focuses on keyword islands within the word sequence, specifying their left-to-right order without giving precise distances between them.

Thus the two approaches have different strengths and weaknesses: the IBM trees specify the words found at a given position but lack the ability to ask simple questions that place a keyword in a wide region of the word sequence, while KCTs deal effectively with keywords found anywhere within a region but cannot ask about precise locations. My personal opinion is that the latter approach more closely models the properties of natural language, which frequently exhibits insertions and deletions of irrelevant or mildly relevant words and phrases. Only experimentation with both approaches can settle this question. It might be interesting, for instance, to apply KCTs to language modeling, and the IBM trees to the generation of semantic rules.

Incidentally, the IBM method for generating compound questions from simple questions would be applicable to KCTs. That is, one could grow set-membership KCTs from single-symbol KCT questions in a different manner than that described above. At each node n of the growing set-membership tree, one would find the best single-symbol question for n and then grow a single-symbol subtree from it. The set-membership question for n would be generated by considering Boolean combinations of the questions in the single-symbol subtree at n. This approach was rejected because it leads to complicated known structures for the nodes of the set-membership tree, but it might be worth considering as a means of improving the performance of set-membership KCTs.

6.5.2 An Application of Classification Trees in Information Retrieval

A recent paper by Crawford, Fung *et al* [Cra91] describes the application of classification trees to an information retrieval task. The ultimate goal of this research is on-line monitoring of newswires for topics of interest. The raw material for the experiments was a collection of 730 Reuters articles

from 1981, dealing with a wide variety of topics including terrorism. The experimental aim was to grow on training data a tree classifier that accurately partitioned new articles into two classes: **terrorism** and **non-terrorism.**

Crawford, Fung *et al* are thus attempting to grow classification trees to learn semantic rules for natural language. From this point of view, their work is quite similar to mine. However, the questions contained in their trees are much simpler than those found in a KCT: they ask only whether a particular word w_i or set of words $\{w_i\}$ occurs or does not occur in an article. Word order is not taken into account.

For instance, one of their trees incorporates the following rules:

```
if article contains word "bomb"
   [ then if article contains words "injure" or "kill"
             then TERRORISM;
     else NON-TERRORISM;
   ]
else if article contains word "kidnapping"
 then TERRORISM;
else NON-TERRORISM.
```

This tree was grown on the 730 classified articles by the Breiman *et al* methodology, including cross-validation pruning. The estimated true error rate was 6%. The researchers grew several different trees from the same data by varying the costs associated with the two possible types of misclassification and showed that recall of **terrorism** articles could be increased at the expense of precision, or *vice versa*, in this way.

The researchers point out that these simple trees are likely to be insufficiently robust in the presence of new data. They therefore grew a *concept tree*, in which the questions in the tree ask about the presence or absence of *subconcepts*. The optimal concept tree was:

```
if article contains subconcept BOMBING
   [ then if contains subconcepts EXPLOSION or KILLING
             then TERRORISM;
     else NON-TERRORISM;
   ]
else if contains subconcept KIDNAP-EVENT and NAMED-TERRORIST
 then TERRORISM;
```

```
else NON-TERRORISM.
```

In order to grow this tree, it was necessary to hand-label the 730 articles in terms of the presence or absence of 18 possible subconcepts (an article can contain any or all of these). Again, experiments with different settings of the misclassification costs were carried out. The resulting trees performed somewhat better than the word-based trees.

The researchers suggest that concept trees could be implemented by growing word-based trees for subconcepts, then running the concept tree on the output of these word-based trees. This is an interesting idea, whose main disadvantage is the extra human time required in labeling articles with subconcepts (to permit the growing of word-based subconcept trees). The results of varying misclassification costs are also interesting. In the application of KCTs described in this thesis, it seemed reasonable to assume uniform misclassification costs, but it would be easy to vary misclassification costs in KCTs if the application called for it.

The Crawford, Fung *et al* classification tree is a greatly simplified version of the KCT that ignores word order. In the course of the growth of this tree, a small subset of the questions that would be generated by the KCT algorithms on the same data is considered. It seems likely that, in general, the KCT approach - which allows questions about the presence or absence of keywords, and their relative order - will outperform a classifier that can only ask about the presence or absence of keywords.

6.5.3 PACE: A Parallel Classifier

Another robust classifier for natural language that learns rules from a training corpus is described in [Cree92]. The interest of this work is the size of the task and the parallel implementation, rather than the nature of the rules employed by the system. Furthermore, in this case thorough, empirical comparison with a knowledge-based approach was possible: it was shown that the robust classifier performed much better than a hand-coded expert system designed for the same task.

The task itself is truly immense - it involves processing 22 million natural language responses to the US Census long form. Specifically, the system must analyze each response to determine which of 232 industry categories the individual's employer belongs to, and which of 504 occupation categories

defines the individual's specific job. Here is a sample response [Cree92 pg. 51]:

```
For whom did this individual work?  Essex Electric.
What kind of business or industry was this?  Photography-Battery Div.
General category (from a list):      Manufacturing.
What kind of work is this person doing?  Apprentice Electrician.
What are this person's most important activities?  Wiring Machinery.
Type of employer (from a list):       Private company.
What is this person's age?        25.
```

According to the census guidelines, for this response the correct industry category is "Photographic Equipment and Supplies" (code 380) and the correct occupation category is "Electrician Apprentices" (code 576).

The Census Bureau had previously designed an expert system called AIOCS to carry out this task, in time for the 1990 census. The creation of AIOCS required 192 person-months of work; on a set of test responses, it correctly assigned 57% of the industry codes and 37% of the occupation codes. PACE, the robust parallel classifier, required only 4 person-months to build. After training on 132,247 previously classified returns, it correctly assigned 63% of industry codes and 57% of occupation codes for the test responses. PACE was implemented on a CM-2 parallel computer with $132K$ simulated processors.

To classify responses, PACE uses a version of the k-nearest neighbour approach; the optimal k was found to be somewhere between 10 and 15. A response is assigned the code that is most frequent among the k nearest classified examples in the database. This approach supports an ingenious confidence score for the chosen code, which measures not only the distance between the current response and the neighbours with the same code, but also how close the next most plausible answer is to the one chosen.

The distance measure itself depends on what the authors call "conjunctive features". Words occurring in the response are given suffixes associated with the line they occur on: for instance, every word in the answer to the question "For whom did this individual work?" is given the suffix ".c"; every word in the answer to "What kind of business or industry was this?" is given the suffix ".i"; every word in the answer to "What kind of work is this person doing?" is given the suffix ".o". From then on, two words which are the same except for different suffixes are considered to be different words.

During training, all 4.5 million possible combinations of two words are considered as possible conjunctive features. A conjunctive feature often contains more information than the two words alone. For instance, the conditional probability of the industry code being "general machinery" given that "shop.i" occurs is 0.24, the probability of this industry code given that "machinist.o" occurs is 0.29, but the probability of the same code given that both "shop.i" and "machinist.o" occur is 0.93.

Various weighting schemes for the conjunctive features were considered, and are described in detail in [Cree92]. Because of the massively parallel implementation, it is possible to calculate weights for each of the 4.5 million conjunctive features in about 10 minutes. Furthermore, despite this huge number of parameters, classification of new responses proceeds at the rate of 10 per second.

An interesting observation was made in the course of PACE's development - it was found that morphological analysis did more harm than good. For instance, one would expect the words "attorney" and "attorneys" to behave very similarly. In fact, the word "attorneys" predicts the "Legal Services" code with 0.98% accuracy, while the word "attorney" only predicts the same code with 68% accuracy. "Blindly stripping prefixes and suffixes from words to arrive at a canonical stem must be used prudently, if at all. It is our belief, and the belief of some other researchers who use data-driven approaches... that the data rather than human intuition should drive the design of the application" [Cree92, pg. 53].

The researchers responsible for PACE cite the following advantages for their robust, massively parallel approach:

- Ease of programming: PACE took 4 person-months to build, while AIOCS took 192 person-months;
- Completeness and uniformity of coverage: because it was grown from training data mirroring the overall data mix, PACE was guaranteed to perform well on that mix;
- Scalability: as the amount of training data and hardware capabilities grow, PACE will improve;
- Ease of updating: it is easy to adjust system behaviour by removing old examples and adding new ones;

- Confidence scores: if a new example is identical to or close to stored examples, PACE can assure the user that it has high confidence in the result - low-confidence results can be submitted to human experts for checking;
- Justification: nearest neighbours can be listed as precedents for PACE's decisions.

PACE's classification rules are of limited interest; nevertheless, this work is important. The comparison between PACE and AIOCS shows that the economic tradeoff between computer effort and human effort is tilting towards the computer. Computation is becoming so ridiculously cheap that it makes more and more sense to devise brute-force, rather than knowledge-based, approaches to natural language tasks where possible. The work also suggests that parallel implementation of such brute-force approaches is an important consideration. A section in Chapter 7 deals with parallel implementation of KCTs.

Chapter 7

Computational Complexity of the KCT Algorithms

7.1 Introduction

This chapter discusses the time complexity of algorithms for growing and using KCTs. It shows that even under extremely unfavourable assumptions, the time required to grow a single-symbol or set-membership KCT by means of the iterative expansion-pruning algorithm is always polynomial in the amount of training data, the length of the longest string, and the size of the vocabulary. For both kinds of KCT, the chapter shows that classification takes relatively little time.

Table 1 summarizes the serial time complexity results for both kinds of KCT. The relevant parameters are D, the number of sentences in the training data (i.e. the total number of sentences in the two training texts used by the iterative expansion-pruning algorithm), L, the upper limit on the number of words in a sentence, and V, the size of the vocabulary.

The algorithms for growing KCTs described in this thesis are well-suited to parallelization. The last section of the chapter shows how different degrees of speed-up can be achieved along the continuum from coarse-grained to fine-grained parallel machines. This chapter thus establishes serial and parallel time dependencies for the KCT algorithms, and demonstrates that growing and using KCTs is relatively cheap in computational terms.

Table 7.1: SERIAL COMPLEXITY RESULTS FOR KCT ALGORITHMS

ALGORITHM	ASSUMPTIONS	COMPLEXITY
Single-symbol growth	balanced	$O(D^2 * \log D * L^2 * V)$
Set-membership growth	balanced	$O(D^2 * \log D * L^2 * V^2)$
Classification	balanced	$O(L * (\log V + \log D))$
Single-symbol growth	$D < 4 * L * V$, unbalanced	$O(D^3 * L^2 * V)$
Set-membership growth	$D < 4 * L * V$, unbalanced	$O(D^3 * L^2 * V^2)$
Classification	$D < 4 * L * V$, unbalanced	$O(D * L)$
Single-symbol growth	$D > 4 * L * V$, unbalanced	$O(D^2 * L^3 * V^2)$
Set-membership growth	$D > 4 * L * V$, unbalanced	$O(D^2 * L^3 * V^3)$
Classification	$D > 4 * L * V$, unbalanced	$O(L^2 * V)$

7.2 Time Complexity of Single-Symbol KCT Algorithms

There are two kinds of KCTs: single-symbol KCTs, in which each question refers to an individual item in the vocabulary, and set-membership KCTs, in which a question may refer to a set of symbols. We will assume that the number of classes is negligible compared to the smallest of the three parameters D, L, and V. For the ATIS system described in Chapters 8 and 9, the maximum value of D was 3254 and V was 592; L was arbitrarily and pessimistically set to 100.

The nature of the worst-case time analysis for both single-symbol and set-membership KCTs depends partly on whether we assume that the supply of training strings, or the supply of possible questions, runs out first. Since no question can be asked more than once along a path from the root to a leaf, it is imaginable that KCT expansion will stop because all possible questions are in the tree. We will therefore begin the analysis by asking two related questions:

1. What is the maximum possible depth of the tree, i.e. the largest number of YES-NO single-symbol questions that can be posed along a path from root to leaf?

2. What is the largest possible number of nodes in the tree?

To determine the maximum depth, note that the maximum number of "+" positions that can appear in a known structure is L. For each "+", at most $4V$ single-symbol questions can be asked: questions of the form "$< w >$?", "$< +w >$?", "$< +w+ >$?", and "$< w+ >$?", for every word w in the vocabulary. Suppose that for a particular string in the training text that has its own leaf, the answer is "NO" to all but one of these questions about a given $< + >$ (i.e. they are either of the wrong type or specify the wrong symbol). After $4V - 1$ "NO" answers we have established the identity of one of the symbols in the substring "+". We now start work on another "+". Since each set of $4V - 1$ questions establishes one of a maximum of L symbols in the string, the maximum number of questions in a path is $L * (4V - 1)$. Thus, the longest path has less than $4 * L * V$ questions in it.

Now, let us determine the maximum number of nodes in a single-symbol KCT. To get the biggest possible KCT, assume there is one leaf for every possible string. Since there are V possible symbols in the first position, V in the second position, and so on, there will be V^L leaves. In a binary tree, the total number of nodes must be less than twice the number of leaf nodes; therefore, the largest possible KCT has $O(V^L)$ nodes altogether. This is an enormous number. For instance, if $V = 100$ and $L = 10$, $V^L = 10^{20}$. It is reasonable to assume that the number of training sentences D is smaller than this. Given this assumption, the largest KCT is obtained when every training sentence yields a leaf node, i.e. when the number of leaves and the total number of nodes are both $O(D)$. Thus, our analysis begins with the assumption that the depth is at most $O(L*V)$, and that the number of nodes is $O(D)$.

The first scenario we consider makes a more optimistic assumption about the maximum depth of the KCT: that it is $O(\log D)$. The only justification for this assumption is that it reflects the experimental results: in practice, KCT depth seems to go up very slowly with D (see Chapter 9). This will be called the "balanced" scenario, though KCTs meeting this description need be balanced only in a very loose sense (e.g. a balanced KCT could have an average depth of $0.5 \log D$ and a maximum depth of $1000 \log D$). If the KCT is unbalanced, we will make the most pessimistic assumption possible about the depth. There are two possible scenarios: either $D < 4 * L * V$ or $D > 4 * L * V$. In the first case, the worst thing that could happen is that

the KCT consists of a single chain $D-1$ questions long, with each question separating only one string from the rest. The second case is similar, except that the supply of possible questions (of which there are about $4 * L * V$) runs out before the training sentences do.

Thus, three worst-case scenarios will be discussed:

1. The balanced scenario shown in figure 7.2, in which KCT depth is $O(\log D)$;

2. The unbalanced scenario shown in figure 7.3, in which $D < 4 * L * V$ and depth is $O(D)$;

3. The unbalanced scenario shown in figure 7.4, in which $D > 4 * L * V$ and depth is $O(L * V)$.

Before considering these scenarios separately, we will consider the "set-up phase" that precedes both growth and classification for single-symbol KCTs in all scenarios.

7.2.1 The Set-up Phase: Converting Sentences to Lexical Index Strings

Figure 7.1 shows how a sentence is converted to a lexical index string: every word in the sentence is replaced by its index in the vocabulary. When a KCT is about to be grown, the training sentences must be converted before the start of the iterative expansion-pruning algorithm. Similarly, when an input sentence is about to be classified by a KCT, conversion must first be carried out. It takes $O(\log V)$ time to look up a single word in the vocabulary and replace it; since each sentence has at most L words in it, the cost of converting a sentence is $O(L * \log V)$.

Thus, it costs $O(D * L * \log V)$ to convert the $O(D)$ sentences in the training text before KCT growth begins, and $O(L * \log V)$ to convert a single sentence before it is classified by a KCT. Since all KCT questions refer to lexical indices rather than words, sentence conversion reduces computation time for KCT growth: it replaces each later $O(\log V)$ lexical lookup by an $O(1)$ integer comparison.

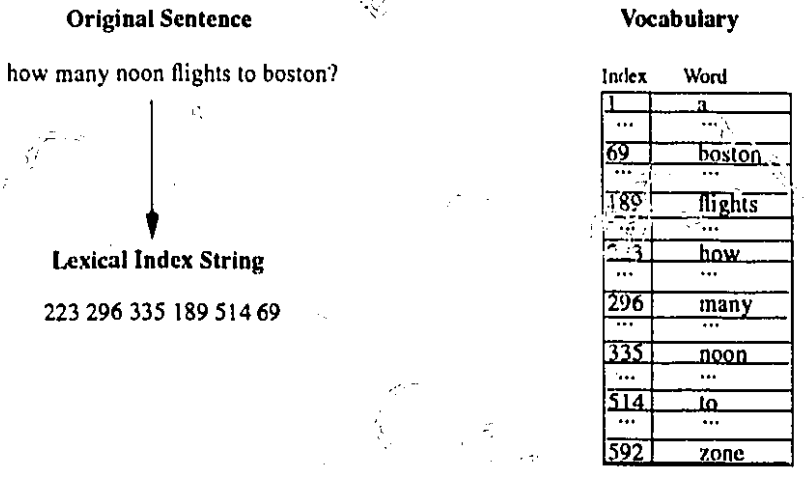


Figure 7.1: Converting Sentences to Lexical Index Strings (Setup Phase)

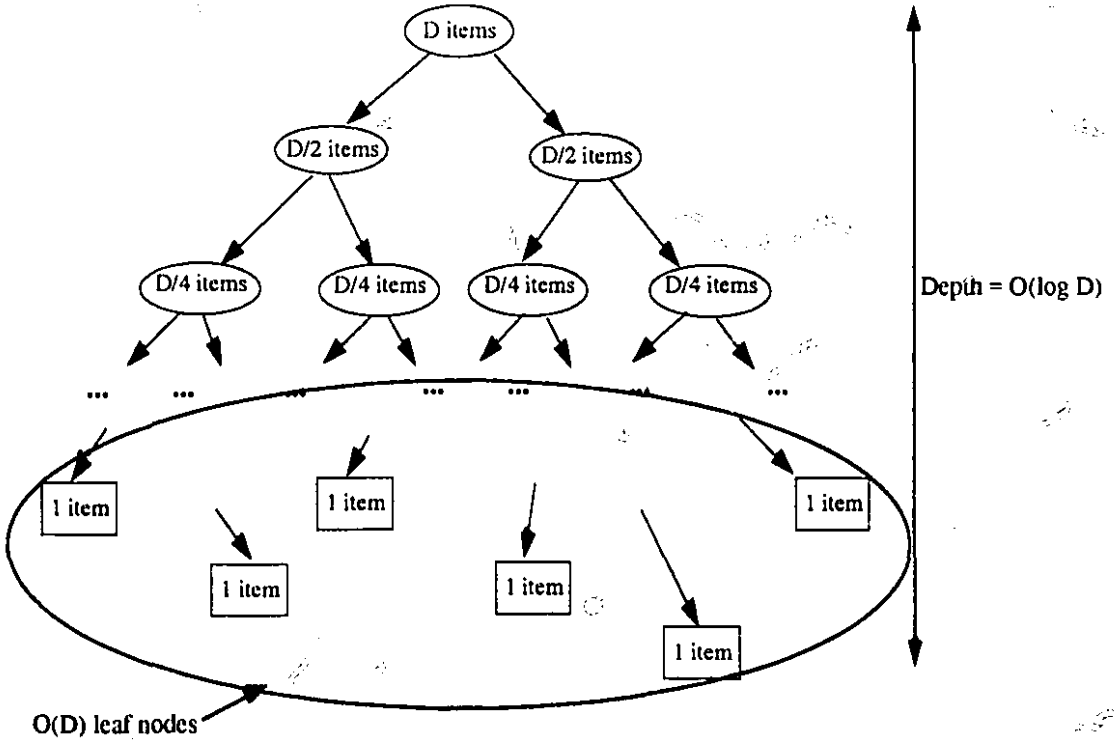


Figure 7.2: Balanced KCT, Depth = $O(logD)$

7.2.2 The Balanced Scenario

Figure 7.2 shows the balanced scenario in which KCT depth is $O(\log D)$. To maximize the work that must be done by the KCT-growing algorithm, we assume that the D data items divide evenly at each question as shown. How much time will it take to grow this KCT?

Consider a node that is being expanded during the expansion-pruning algorithm. We have established that the maximum number of single-symbol questions that can be posed at any node is $O(L * V)$. Consider a single training lexical index string s of length at most L that is being asked a particular question. In the worst case, the known structure is $< + >$, the question is "$< +i+ >$?" where i is the index of some word $w(i)$ in the

lexicon, and the answer will turn out to be "NO". In this case, i is compared to the second, third, ..., second last indices in s one after the other, requiring $O(L)$ time. Once a "YES" or "NO" answer has been obtained, a counter array "YEScount[c]" or "NOcount[c]" is incremented, where c is the class of the training string (an $O(1)$ operation).

Thus, when the root of the KCT is grown during the first expansion in the expansion-pruning algorithm, each question costs $O(L)$ and each is posed to D strings, so a question costs $O(D*L)$ time. After a question has been posed to all D training strings, the change in Gini impurity must be calculated from the arrays "YEScount[]" and "NOcount[]". This is an $O(C^2)$ operation where C is the number of classes, which we assume is trivial compared to D, L, and V. Picking the best question from those whose impurity has been calculated is an $O(L*V)$ operation (we just keep track of the best question so far, and its impurity, replacing both whenever an even better question is found). Thus the total work at the root to find the best question for D training strings is $O(D*L)$ work per question for $O(L*V)$ possible questions: $O(D*L^2*V)$ work (the $O(L*V)$ question comparison work is too small to affect the overall time complexity).

Once the best question for the root has been found, some of the data go to the YES child and some to the NO child. In each, a maximum of $O(L*V)$ questions will be considered. At this layer of the KCT, if we consider the YES child and the NO child together, D strings are asked $O(L*V)$ questions. Thus, the time taken for this layer will be the same as at the root: $O(D*L^2*V)$.

In fact, this will be the amount of work done at each layer of the tree. Since the depth is $O(\log D)$, we conclude that the total work done during the first expansion is $O(D*\log D*L^2*V)$. Note that the cost of the set-up phase, $O(D*L*\log V)$, is insignificant. If we had skipped this phase and done lexical lookups each time a string is asked a question, the cost of a question on a single string would have been $O(L*\log V)$ instead of $O(L)$ and we would have ended up with an extra factor of $\log V$ in the final result.

Because the question for each node has been selected, the pruning stage of the expansion-pruning algorithm takes less work. During the set-up phase, a new training text with D strings in it is converted in $O(D*L*\log V)$ time. The resulting lexical index strings are read into the root and the misclassification rate $R(root)$ is calculated in $O(D)$ time. Next, the question at the root is applied to each of the strings to obtain the answer "YES"

or "NO" in $O(D * L)$ time. The total work at the root is thus $O(D * L)$. The D strings are then partitioned between the two children and the same computations carried out at this level, and continued recursively at lower levels. Since each level has $O(D)$ strings to work on, each does $O(D*L)$ work. There are $O(\log D)$ levels, so the total work for this stage of pruning is $O(D* \log D * L)$. At the final phase of pruning, the algorithm works recursively upwards from the leaves, pruning them if their combined misclassification rate is higher than that of their parent. Using a stack, this can be done in time linear in the number of nodes in the tree, $O(D)$. Thus the time required for pruning is $O(D * L * \log V + D * \log D * L + D)$. If we assume that vocabulary size V is small compared to the number D of training strings, this is $O(D * \log D * L)$. One cycle of expansion followed by iteration is therefore $O(D * \log D * L^2 * V + D * \log D * L) = O(D * \log D * L^2 * V)$.

Finally, we must establish the number of cycles of expansion-pruning required by the iterative algorithm. Gelfand *et al* [Gel91] have proved that at the end of each such cycle, the size of the tree has either increased since the previous cycle, or stayed the same. If the size has stayed the same, convergence has occurred and the iteration stops - we have the final KCT. So assume that the size of the tree increases on each iteration. We have established that the maximum size of the KCT is $O(D)$. Thus, we will get the maximum number of iterations if we assume that the first pruned tree has size 1, the second has size 2, and so on until $O(D)$ is reached. There were $O(D)$ cycles, each requiring $O(D * \log D * L^2 * V)$ operations, so the total time required to grow a single, balanced KCT is $O(D^2 * \log D * L^2 * V)$. This is a very loose (pessimistic) upper bound; with some work it would probably be possible to prove a tighter upper bound. In particular, the assumption of $O(D)$ cycles of expansion and pruning is extremely pessimistic. When growing KCTs for the ATIS task, I have never encountered a situation where more than 4 cycles were needed, even for $D > 3000$.

Once the balanced KCT has been grown, how many operations are required to classify a given string? Recall that for a single string, the set-up phase is $O(L * \log V)$. At each node encountered during classification, determining the answer to the node's question may require looking at up to L positions in the lexical index string. Since there are at most $O(\log D)$ levels in a balanced KCT, classification of the lexical index string is $O(\log D * L)$. Thus, total time complexity for classification of a string by a balanced KCT is $O(L * \log V + \log D * L)$. This is negligible in any realistic case, especially

in speech understanding, where the earlier recognition stages are guaranteed to require orders of magnitude more time.

7.2.3 The Unbalanced Scenario with $D < 4 * L * V$

In this scenario, the length of the longest path is $O(D)$ nodes - the supply of training strings is exhausted before the supply of questions runs out. Using the same arguments as for the balanced scenario, it is straightforward to establish the time complexity of the iterative expansion-pruning algorithm. For the expansion step, we still have $O(D * L^2 * V)$ operations per level, but now we have $O(D)$ levels, implying $O(D^2 * L^2 * V)$ time complexity. Pruning is still cheaper than this, so the cost of one iteration cycle is $O(D^2 * L^2 * V)$; there is still a maximum of D cycles. Thus, in this scenario the time complexity of growing the tree is $O(D^3 * L^2 * V)$. Note that the substitution of $O(D)$ tree depth for $O(\log D)$ tree depth yields precisely $O(D/\log D)$ increase in complexity. Subsequent use of the tree to classify a string of maximum length L is clearly $O(D * L)$ - again, negligible in most imaginable circumstances.

7.2.4 The Unbalanced Scenario with $D > 4 * L * V$

In this scenario, there are enough training strings to fill out the longest possible path, which we established earlier has depth approximately $4 * L * V$. $O(D)$ strings must still be processed at each level of the tree, but the worst-case number of levels is $O(L * V)$. Thus $O(D * L^2 * V)$ work is done per level in an expansion step as before, but there are now $O(L * V)$ levels, so that an expansion step requires $O(D * L^3 * V^2)$ work.

Again, because the pruning step requires less time than the expansion step, this expression also describes the total number of operations required for one cycle of the iterative algorithm. By the argument used previously, the number of iterations is $O(D)$, so the total time for growing this kind of KCT is $O(D^2 * L^3 * V^2)$. When this kind of KCT classifies a string, the worst-case number of operations is still the product of maximum string length L with maximum depth, i.e. $O(L^2 * V)$.

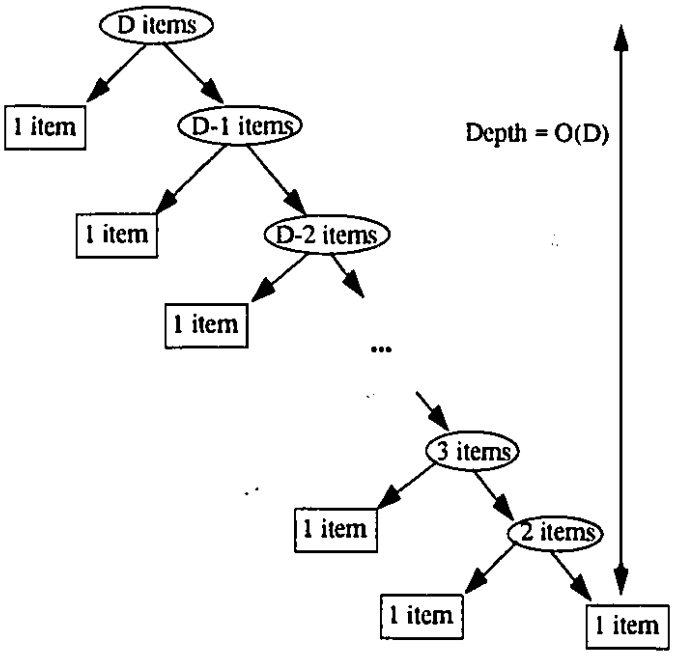


Figure 7.3: Unbalanced KCT, $D < 4 * L * V$, Depth $= O(D)$

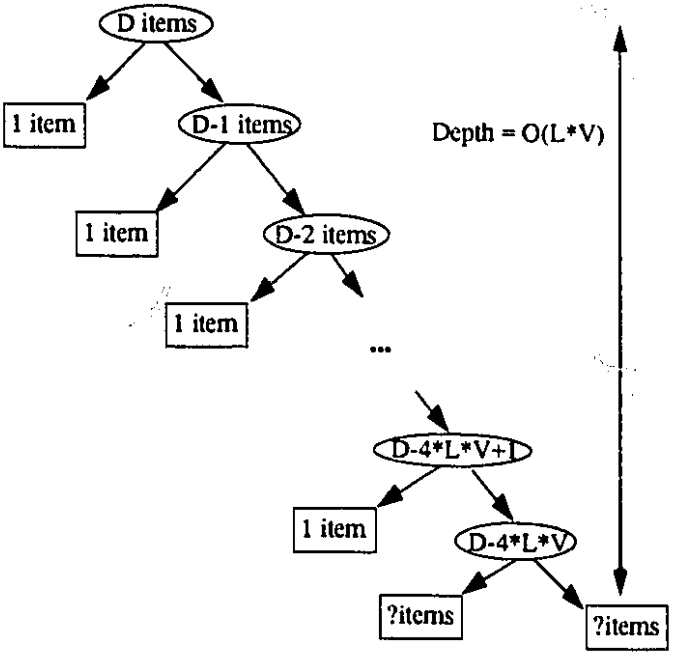


Figure 7.4: Unbalanced KCT, $D > 4 * L * V$, Depth $= O(L * V)$

7.3 Time Complexity of Set-Membership KCT Algorithms

In this section, I will discuss KCTs containing questions of the form: "within a given segment of the string, does any symbol belonging to the set of symbols X occur?" As described in Chapter 6, there are a number of different types of set-membership questions. For simplicity, I will usually assume that each set-membership question asks about a precise, fixed location of the string. The types of set-membership questions actually used are no more computationally expensive than this kind.

To begin the analysis, consider the time required to determine whether a particular string s yields the answer "YES" or "NO" in response to an arbitrary set-membership question.

7.3.1 Time Complexity of Set-Membership Questions

Figure 7.5 shows how a set-membership question is posed. The set of words X referred to by the question is stored in an array (also called "X" in the diagram) of V bits, with bits corresponding to the words in X set to 1 and the rest set to 0. The question also refers to a region of the sentence demarcated by left and right positions (not shown). As with the single-symbol case, assume that a set-up phase has already converted the sentence to a string of lexical indices.

To determine if the answer to the question is "YES" or "NO" for a given string s, traverse the string from the appropriate position on the left to the appropriate position on the right and check for each number i appearing there whether $X(i)$ is 1. If the answer is "YES" for any i, the answer for the string s as a whole is also "YES". In figure 7.5, the question is "$< +X+ >$?" so the numbers in s to be checked range from the second number (200 in the figure) to the second last number (3 in the figure). Since each check requires an $O(1)$ array access and comparison, and there are at most L of those, since s cannot have a length greater than L, the cost of obtaining an answer to any set-membership question on any string is $O(L)$. Thus, set-membership questions are no more costly to ask than single-symbol questions, apart from the cost of setting up the bit array X. This is the next aspect of set-membership questions we must look at: how are they generated?

Question: "IS STRING s OF FORM <+X+>, WHERE X IN SET {2,5,352}?"

Index -> 1 2 3 4 5 352 V

X = [0 | 1 | 0 | 0 | 1 | 0 |... | 0 | 1 | 0 |...| 0]

s = 35 200 9 14 ... 540 3 17

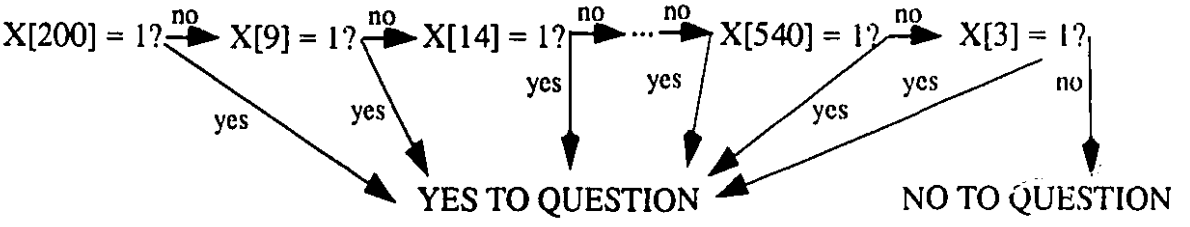


Figure 7.5: Asking Set-Membership Question (for 1 string, $O(L)$ work)

7.3.2 Generating Set-Membership Questions

Recall that each set-membership question is generated from the best single-symbol question at a fixed position in the known structure. Suppose there are N training strings in a given node. At a fixed position in the known structure, $O(V)$ single-symbol questions can be generated; each costs $O(L)$ to ask per training string (we may have to traverse a large fraction of the string to ask the question). Thus, the cost of finding the best single-symbol question at a fixed position in the known structure for N training strings is $O(L * N * V)$.

Figure 7.6 shows how set-membership questions are generated from a particular single-symbol question. Suppose the single-symbol question involves the word at position i in the lexicon. Then we initialize a bit array X to $X[i] = 1$, and set all other bits in X to 0. This involves $O(V)$ work.

To find a good two-element X, we leave $X[i] = 1$ and try $X[1] = 1$. By the argument of the previous section, measuring the Gini impurity of this question over N training strings requires $O(L * N)$ work. We next set $X[1]$ back to 0 and calculate the impurity when $X[2] = 1$; again, this requires $O(L * N)$ work. We continue in this way until we have $X[V] = 1$ and all two-element sets that include i have been tried. Clearly, $V - 1$ different settings have been tried at the cost of $O(L * N * V)$ time to obtain the two-element question involving i that yields the lowest impurity. Let this two-element question have the setting $X[i] = 1, X[j] = 1, i \neq j$, and all other elements of X set to 0.

To find the best three-element X, we start with the two-element setting and try setting a third bit to 1; there are $V - 2$ ways of doing this. To get the best three-element question based on the two-element question requires $O(L * N * V)$ time.

The process continues until the question obtained with $M + 1$ elements is no improvement on the M-element question. Conceivably, it could continue until $V - 1$ bits in X are set to 1. If this happened, the cost would be $O(L * N * V^2)$.

This is the cost for finding the set-membership question for one position in the known structure from N training strings. Since there can be $O(L)$ "+" in the known structure, the cost of the final set-membership question obtained at a single node from N training strings is $O(L^2 * N * V^2)$.

1. Start with a single-symbol question at a fixed position, involving word i in lexicon.

Set X = i:

Index -> 1 2 3 4 5 i V

X = [0 | 0 | 0 | 0 | 0 | 0 |... | 0 | 1 | 0 |...| 0]

2. Find best two-element question involving word i.

2.1: try X = {1,i} Index -> 1 2 3 4 5 i V

X = [1 | 0 | 0 | 0 | 0 | 0 |... | 0 | 1 | 0 |...| 0]

Cost = O(L*N)

2.2: try X = {2,i} Index -> 1 2 3 4 5 i V

X = [0 | 1 | 0 | 0 | 0 | 0 |... | 0 | 1 | 0 |...| 0]

Cost = O(L*N)

...

2.V: try X = {V,i} Index -> 1 2 3 4 5 i V

X = [0 | 0 | 0 | 0 | 0 | 0 |... | 0 | 1 | 0 |...| 1]

Cost = O(L*N)

Lowest impurity question: X = {i,j}

Index -> 1 2 3 i j V

X = [0 | 0 | 0 |... | 0 | 1 | 0 |...| 0 | 1 | 0 |...|0]

Total cost for two-element question = O(L*N*V)

3. Find best three-element question involving i,j.

3.1: try X = {1,i,j} Index -> 1 2 3 i j V

X = [1 | 0 | 0 |... | 0 | 1 | 0 |...| 0 | 1 | 0 |...|0]

...

3.V: try X = {i,j,V} Index -> 1 2 3 i j V

X = [0 | 0 | 0 |... | 0 | 1 | 0 |...| 0 | 1 | 0 |...|1]

Total cost for three-element question = O(L*N*V)

...

Up to V-1 elements, so cost is O(L*N*V*V)

Figure 7.6: Finding Set-Membership Question from Single-Symbol Question (N training strings)

7.3.3 Determining Time Complexity for the Set-Membership Scenarios

Just as with the single-symbol KCT, we must answer two questions to complete the time complexity analysis:

1. What is the maximum possible depth of the set-membership KCT, i.e. the largest number of YES-NO set-membership questions that can be posed along a path from root to leaf?

2. What is the largest possible number of nodes in the KCT?

Consider a fixed position in the string. What is the length of the longest chain of questions that can be asked about this position? Initially, when nothing is known about this position, X can be set to any proper non-empty subset of the vocabulary. Thus, there are $2^V - 2$ possible questions that can be asked, since each element of the vocabulary can be included or not included in the set X, and we must subtract 2 to eliminate the two cases where X is empty or equal to the whole vocabulary (which give rise to pointless questions). To get the longest possible chain of questions, one might reason that it can be obtained via a series of "NO" answers. There are $2^V - 2$ possible questions that can initially be asked about the position; once one of them is chosen, there are $2^V - 3$ possible questions to be asked if the answer to the first question is "NO". Once one of these has been chosen, there are $2^V - 4$ possible questions left over for the NO child to use - and so on. Following this reasoning, the number of questions that could be asked along a path from root to leaf about a fixed position would be $2^V - 2$ (implying that the maximum depth for a set-membership KCT is $O(L * 2^V)$, since we have L positions in the longest string).

This reasoning is fallacious. Each answer to a question, whether it is "YES" or "NO", greatly restricts the scope of the questions that follow. Imagine that as we move along the chain of questions about a particular position in the known structure, we are filling in an array A of V integers. Suppose that $A[i] = 1$ means that word i occurs at this position, $A[i] = 0$ means word i does not occur at this position, and $A[i] = -1$ means that it is not known whether word i occurs at this position or not. At the root, for a given position in the known structure, $A[i] = -1$ for all i.

Now, consider the set-membership question: "does an element of the set Y occur at this position?" In the NO child of this question, we can set $|Y|$

elements of array A, those corresponding to the elements of Y, to 0. In the YES child of this question, we know that none of the elements of $V - Y$ can occur at the given position, so $|V - Y|$ elements of array A are set to 0. Thus, every time a question is encountered, at least one element of array A is set to 0. We can only continue to ask questions about a fixed position if some of the V positions in A are still set to -1. Therefore, the maximum number of set-membership questions encountered along a path from root to leaf about a fixed position in the known structure is V. The maximum number of positions in the known structure is $O(L)$; therefore, the maximum set-membership KCT depth is $O(L * V)$.

We now have a situation analogous to that described in the section on single-symbol KCT algorithms: there, also, the maximal depth of the KCT was $O(L * V)$. This is fortunate - it means that many of the arguments employed in that section can be recycled! For set-membership KCTs, we again have a three-fold division:

1. The balanced scenario shown in figure 7.2, in which KCT depth is $O(\log D)$;

2. The unbalanced scenario shown in figure 7.3, in which $D < 4 * L * V$ and depth is $O(D)$;

3. The unbalanced scenario shown in figure 7.4, in which $D > 4 * L * V$ and depth is $O(L * V)$.

For all of these, we know that the time complexity of finding the set-membership question for a node from N training strings during the expansion phase of the expansion-pruning algorithm is $O(L^2 * N * V^2)$. Finding set-membership questions by means of the greedy heuristic, rather than finding the best single-symbol question, has increased the worst-case complexity of the KCT-growing algorithm by a factor of V.

Here is the time complexity for each of the three scenarios:

1. The balanced scenario - work per level of expanding tree is $O(D * L^2 * V^2)$, there are $O(\log D)$ levels, so one expansion-pruning cycle is $O(D * \log D * L^2 * V^2)$. Since there can be at most D cycles, the time complexity of growing a balanced set-membership KCT is $O(D^2 * \log D * L^2 * V^2)$.

2. The unbalanced scenario for $D < 4 * L * V$ - the number of levels is $O(D)$ rather than $O(\log D)$, but otherwise the analysis is the same as in the balanced scenario. Therefore, the time complexity of growing an unbalanced set-membership KCT is $O(D^3 * L^2 * V^2)$.

3. The unbalanced scenario for $D > 4 * L * V$ - work per level is still $O(D * L^2 * V^2)$, but the number of levels is $O(L * V)$, so the work required for one cycle of the iterative expansion-pruning algorithm is $O(D * L^3 * V^3)$. The maximum number of cycles is $O(D)$, so the worst-case time complexity for growing a set-membership KCT is $O(D^2 * L^3 * V^3)$.

We saw that classification of a string at a node takes no more time (within a constant factor) for set-membership trees than it does for single-symbol trees, so the previous time complexity results for classification still hold.

7.4 Parallel Time Complexity of KCT Algorithms

> The most important Connection Machine data structure is the tree. Trees are used by themselves and as components of other data structures, such as graphs, arrays, and butterflies... Trees are useful because they provide a fast way of collecting, combining, and spreading information to and from the leaves.
>
> W. Daniel Hillis [Hil85 pp. 97-98]

This quotation from one of today's most respected researchers in the field of massively parallel computation suggests that the KCT algorithms might be unusually well-suited to parallel implementation. This turns out to be the case. In fact, there are three different "dimensions" of parallelism that may be exploited independently in the course of growing an KCT, allowing us to pick an implementation that suits the degree of parallelism available on the machine we are using.

The most obvious way of using parallelism is to assign a processor to each node of the expanding tree. The time complexity at the root node, which receives D strings, will be what it is in the serial implementation. However, if the tree is balanced, and the training strings are divided fairly evenly

between the leaves, the time required at the *next* level is halved: there are two processors running simultaneously, each working with $O(D/2)$ strings. By the same logic, the time will be halved again at the next level, and so on. Under these assumptions - which are admittedly rather strong - if the time taken at the root node is T, the total time is $T*(1+0.5+0.25+...) = 2*T$ (ignoring communication costs). In effect, this reduces the serial time complexity by a factor equivalent to the maximum depth of the tree. This would reduce the time complexity for growing a balanced tree (single-symbol or set-membership) by a factor of $\log D$. However, that this will only happen if the training strings are evenly distributed across leaves. If the tree is unbalanced or the strings concentrated in a small fraction of leaf nodes, assigning a processor to each node makes no difference.

Fortunately, the remaining two "dimensions" of parallelism we will explore give *guaranteed* speedup, independent of the shape of the tree being grown and the nature of the training data. Suppose that at least $O(V)$ processors are available for the tree-growing task. Assign one processor to each symbol in the vocabulary, so that every time questions involving the V symbols are being compared to find the best one, the impurities are calculated in parallel. Each processor will ask a question, each involving at most $O(L)$ work (in either the single-symbol or the set-membership case), for its assigned symbol at each of $O(L)$ positions in the known structure over $O(D)$ strings. It also keeps track of which of these questions yields the greatest drop in impurity. This takes $O(D*L^2)$ time for each processor. In $O(\log V)$ time, the V processors can then compare results to give the best question [Hil85 pg. 100]. Finding the best question thus takes $O(D*L^2+\log V)$ time. On the reasonable assumption that $\log V$ is small compared to $D*L^2$, this is $O(D*L)$ - an $O(V)$ speedup over the serial algorithm.

In fact, this speedup will be attained at each level of the expanding tree, no matter what its shape or the distribution of training strings: looking at possible questions during the expansion step will be faster by a factor of order $O(V)$ everywhere. The pruning step is *not* speeded up, because here the questions have already been chosen at each node and there is no need to go through the vocabulary at each level; however, the time taken for each iteration does shrink by $O(V)$, because it is determined by the computationally more expensive expansion step. The total time complexity is thus obtained by dividing the serial expressions by V.

Finally, if we can afford $O(D)$ processors, we can assign one of them to each training string and have it carry out the necessary computations on that string at each node. Naturally, this produces an $O(D)$ speedup, this time during pruning as well as during expansion.

Which of these parallelization strategies is adopted depends, of course, on how many processors are available. Note that if we parallelize by vocabulary item or by training string but have a number of processors less than the vocabulary size V or the number of training strings D respectively, we still get speedup proportional to the number of processors. For instance, if we have exactly $D/2$ processors, we can work on half the training strings in one round and the other half in the next round, getting a $D/2$ speedup - the speedup varies smoothly with number of processors, rather than showing quantum jumps. If we have a truly enormous number of processors, for instance $O(D * V)$, we could combine two or even three of these strategies, getting a speedup of $O(D * V)$ or better.

Classification of a single string is so fast in the serial implementation that it is probably not worthwhile to parallelize it. Obviously, parallelism would come in handy for a KCT-based system that looked at all N hypotheses output by the recognizer. In designing the KCT-based robust matcher described in Chapter 8, I kept an open mind about whether to use KCTs in parallel or in series (the implementation runs on a serial machine). As it turned out, it was much more convenient to build the matcher out of about 100 KCTs that function in parallel; each handles a different linguistic aspect of the problem. Thus, the KCT-based robust matcher could be implemented in parallel with a speedup of 10 – 100 at the cost of very little effort if a parallel machine were available.

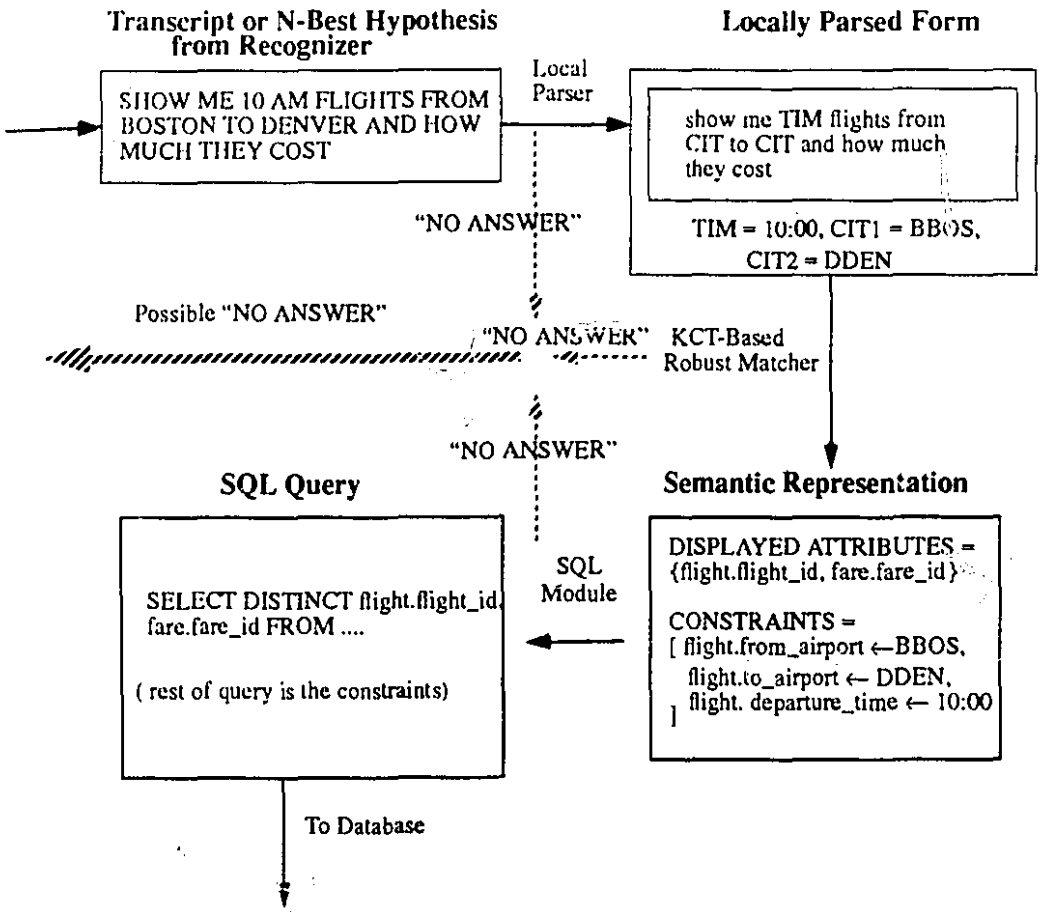
Chapter 8

CHANEL: A KCT-Based Linguistic Analyzer for ATIS

8.1 System Structure for Two ATIS Tasks

Figure 8.1 shows the structure of the linguistic analyzer built at CRIM for the November 1992 ATIS benchmarks. This linguistic analyzer was called CHANEL, for "CRIM Hybrid Analyzer for Natural Language" (the word "hybrid" is a reminder that the analyzer consists of a local chart-based parser plus the Robust Matcher). The Robust Matcher component was trained on class "A" (acceptable) but not class "D" (context-dependent) or class "X" (unacceptable) sentences from ATIS 2, the 1992 ATIS training data release. For the NL (natural language) benchmarks the input to the matcher is a transcript of what a user said, while for the SLS (spoken language systems) benchmarks the input is the most probable word sequence hypothesis output by the CRIM recognizer. Originally, it was planned to train the robust matcher for the SLS task on labelled recognizer output. However, the experiments described in the next chapter suggested that even for the SLS task, it was better to use transcript training data. Thus, exactly the same robust matcher was used for both NL and SLS benchmarks.

Note from figure 8.1 that word sequences are processed by a **local parser** before being submitted to the matcher. However, I also considered an alternative, more complex architecture in which some processing by KCTs precedes local parsing; this alternative will be discussed in the last section of



Tasks of the KCT-Based Robust Matcher

1. must decide which attributes to display
2. must decide whether TIM applies to flight.departure_time or flight.arrival_time
3. must decide how to assign each CIT to flight.from_airport, flight.to_airport, or flight_stop.stop_airport
4. must resolve ambiguities (if any) for other semantic categories: e.g. AIL, AIP, DAT, DAY, FNB, etc.
5. may decide to send "NO ANSWER" (as may Local Parser or SQL Module)

Figure 8.1: Linguistic Processing in 1992 CRIM ATIS System

this chapter. Also note that the decision to send a "NO ANSWER" response may be made by the local parser, the robust matcher, or the SQL module. When such a decision is made, further processing is aborted and "NO ANSWER" is sent to the ATIS comparator. For interaction with real users, an appropriate message such as "I'm sorry, would you mind repeating your question?" could appear on the screen. The meta-rules for deciding when "NO ANSWER" is appropriate will be discussed in the last section of this chapter.

Before discussing the robust matcher itself, I will explain the roles of the modules on either side of it, with the aid of some typical examples from the ATIS task.

8.2 Examples of ATIS Data

Four examples drawn from the most recent NIST-supplied ATIS data will illustrate the nature of the data used to train KCTs, and the nature of the tasks carried out by the different modules of the complete system. Each example shows a transcript of a user question, followed by a representation for it in the CRIM semantic representation language, followed by the desired SQL database query. NIST supplies both MIN and MAX versions of the SQL query for each sentence; a correct answer will include at least the information requested by the MIN query and no more than the information requested by the MAX query. The MIN queries are shown here. The training data for KCTs is derived from thousands of sentence-SQL query pairs: the input to the KCT-growing algorithms consists of sentences that are labelled to reflect the contents of their SQL "translations". The goal of the complete system is to reverse the process - to generate the correct SQL query from a word sequence.

Each of the SQL queries shown below begins with a list of attributes between the words "SELECT DISTINCT" and "FROM". We will call these the **displayed attributes**, since they represent the columns in the database that will be displayed to the user. A displayed attribute consists of a table name, followed by a dot, followed by the name of a column in the table. The rest of a query is called the **constraints**.

In Example 1 the displayed attribute is *flight.flight_id*, in Example 2 it is *fare.fare_id*, in Example 3 the displayed attributes are *ground_service.city_code*,

ground_service.airport_code, and *ground_service.transport_type*, and in Example 4 the displayed attribute is *flight.flight_id*. Occasionally, the displayed information is a function of an attribute rather than an attribute. For instance, a sentence beginning "HOW MANY FLIGHTS..." is often translated into an SQL query beginning "SELECT count(*) FROM flight". These rare cases can be treated in the same way as displayed attributes; they create no additional difficulties.

Despite the apparent simplicity of the displayed attributes and the apparent complexity of the constraints in the examples, determining the correct set of displayed attributes from the word sequence is as hard as determining the constraints. The complicated nesting shown within the constraints for each SQL query is a reflection of the large number of database joins needed to construct the constraints, rather than a reflection of any inherent difficulty in decoding the constraints in the sentence. If the database were structured somewhat differently, much of this apparent complexity would disappear. Of course, many sentences in the ATIS domain are constructed in a manner that makes determination of the constraints difficult, though that is not the case for these four examples.

8.3 The Semantic Representation Language

8.3.1 Details of the Representation

Every ATIS system has some kind of semantic representation language for the output of the natural language component. The function of the semantic representation is to carry information from the natural language component to the component that generates the SQL query; semantic representation languages therefore tend to have some of the characteristics of natural language and some of the characteristics of database query languages.

The semantic representation language at CRIM was developed by Ying Cheng, Bassem Khalife, and Charles Snow, and strongly resembles SQL. As Example 4 shows, a sentence may permit more than one correct representation; it does not matter which is actually generated, provided it yields the correct SQL query. The representation is divided into two parts, the **displayed attributes** and the **constraints**. The displayed attributes are the columns of the database the user wishes to see; each is in the form "rela-

```
SHOW ME FLIGHTS FROM BOSTON TO DENVER

DISPLAYED ATTRIBUTES = {flight.flight_id}
CONSTRAINTS =
[
 flight.from_airport <- BBOS,
 flight.to_airport <- DDEN
]

( SELECT DISTINCT flight.flight_id FROM flight WHERE ( flight . from_airport
 IN ( SELECT airport_service . airport_code FROM airport_service WHERE
 airport_service . city_code IN ( SELECT city . city_code FROM city WHERE
 city.city_name = 'BOSTON' )) AND flight . to_airport IN ( SELECT
 airport_service . airport_code FROM airport_service WHERE
 airport_service . city_code IN ( SELECT city . city_code FROM city WHERE
 city.city_name = 'DENVER' )) )   ) ;
```

Example 1

```
ALL RIGHT WHAT I'D LIKE TO DO IS FIND THE CHEAPEST ONE WAY FARE FROM BOSTON
TO DENVER

DISPLAYED ATTRIBUTES = {fare.fare_id}
CONSTRAINTS =
[
 fare.one_direction_cost <- $MIN,
 fare.round_trip_required <- !NULL,
 flight.from_airport <- BBOS,
 flight.to_airport <- DDEN
]

( SELECT DISTINCT fare.fare_id FROM fare WHERE ( fare.one_direction_cost =
( SELECT MIN ( fare.one_direction_cost ) FROM fare WHERE
fare.round_trip_required = 'NO' AND fare . fare_id IN ( SELECT
flight_fare . fare_id FROM flight_fare WHERE flight_fare . flight_id IN
( SELECT flight . flight_id FROM flight WHERE ( flight . from_airport IN
( SELECT airport_service . airport_code FROM airport_service WHERE
airport_service . city_code IN ( SELECT city . city_code FROM city WHERE
city.city_name = 'BOSTON' )) AND flight . to_airport IN ( SELECT
airport_service . airport_code FROM airport_service WHERE
airport_service . city_code IN ( SELECT city . city_code FROM city WHERE
city.city_name = 'DENVER' )) )  )) ) AND fare . fare_id IN ( SELECT
flight_fare . fare_id FROM flight_fare WHERE flight_fare . flight_id IN
( SELECT flight . flight_id FROM flight WHERE ( flight . from_airport IN
( SELECT airport_service . airport_code FROM airport_service WHERE
airport_service . city_code IN ( SELECT city . city_code FROM city WHERE
city.city_name = 'BOSTON' )) AND flight . to_airport IN ( SELECT
airport_service . airport_code FROM airport_service WHERE
airport_service . city_code IN ( SELECT city . city_code FROM city WHERE
city.city_name = 'DENVER' )) )  )) )   ) ;
```

Example 2

```
I WOULD LIKE INFORMATION ON GROUND TRANSPORTATION IN THE CITY OF ATLANTA
FROM AIRPORT TO DOWNTOWN

DISPLAYED ATTRIBUTES =
{ground_service.city_code,ground_service.airport_code,
 ground_service.transport_type}
CONSTRAINTS =
[
 ground_service.airport_code <- ATL,
 ground_service.city_code <- MATL
]

( SELECT DISTINCT ground_service.city_code , ground_service.airport_code ,
ground_service.transport_type FROM ground_service WHERE
( ground_service . airport_code IN ( SELECT airport . airport_code FROM
airport WHERE airport.airport_code = 'ATL' ) AND ground_service . city_code IN
( SELECT city . city_code FROM city WHERE city.city_name = 'ATLANTA' ) )    ) ;
```

Example 3

```
SHOW ME ALL THE FLIGHTS BETWEEN DALLAS FORT WORTH AND EITHER SAN FRANCISCO
OR DENVER THAT DEPART BETWEEN FIVE AND SEVEN P M

DISPLAYED ATTRIBUTES = {flight.flight_id}
CONSTRAINTS =
[
 flight.from_airport <- DFW,
 flight.departure_time <- (>=17:00&<=19:00)
]
&
[
 [
  flight.to_airport <- SFO
 ]
 |
 [
  flight.to_airport <- DEN
 ]
]

or

DISPLAYED ATTRIBUTES = {flight.flight_id}
CONSTRAINTS =
[
  flight.from_airport <- DFW,
  flight.departure_time <- (>=17:00&<=19:00)
  flight.to_airport <- SFO
]
 |
[
  flight.from_airport <- DFW,
  flight.departure_time <- (>=17:00&<=19:00)
  flight.to_airport <- DEN
]
```

Example 4 (first part)

```
( SELECT DISTINCT flight.flight_id FROM flight WHERE
( ( flight . from_airport IN ( SELECT airport_service .  airport_code
FROM airport_service WHERE airport_service . city_code IN
( SELECT city . city_code FROM city WHERE city.city_name = 'DALLAS' ))
OR flight . from_airport IN ( SELECT airport_service . airport_code FROM
airport_service WHERE airport_service . city_code IN ( SELECT city . city_code
FROM city WHERE city.city_name = 'FORT WORTH' )) ) AND ( ( flight . to_airport
IN ( SELECT airport_service . airport_code FROM airport_service WHERE
airport_service . city_code IN ( SELECT city . city_code FROM city WHERE
city.city_name = 'SAN FRANCISCO' )) OR flight . to_airport IN
( SELECT airport_service . airport_code FROM airport_service WHERE
airport_service .  city_code IN ( SELECT city . city_code FROM city WHERE
city.city_name = 'DENVER' )) ) AND ( flight.departure_time >= 1700 AND
flight.departure_time <= 1900 ) ) )    ) ;
```

Example 4 (continued)

tion.attribute". They are given as a list.

The constraints are made up of one or more *frames*, each of which is surrounded by square brackets '[' and ']'. Frames may be combined by means of the AND operator '&' or the OR operator '|'; they may also be nested. A frame may also be negated by the NOT operator '!' (though we have not yet encountered an example where this was necessary). Frames contain *descriptors* of the form *relation.attribute* $\Leftarrow$ *value*; descriptors in the same frame are implicitly ANDED together.

The representation also allows for *functions*, which are always preceded by the symbol '$' and which may occur in either the displayed attributes or the constraints (usually the latter). Example 2 above shows how the constraint "cheapest" for a one-way flight is translated into the function MIN acting on *fare.one_direction_cost*. In the case of "latest flight ..." we would include among the constraints the descriptor *flight.departure_time* $\Leftarrow$ *$MAX*. A trickier example would be "the latest afternoon flight ..." which would yield the descriptor *flight.departure_time* $\Leftarrow$ *$MAX* (>= 1200 & <= 1800). Functions among the displayed attributes are treated similarly: for instance, a sentence beginning "how many flights..." would have the attribute list {*flight.flight_id* $\Leftarrow$ *$COUNT*}. The possible functions are MAX, MIN, AVG,

and COUNT.

Finally, as Example 2 above indicates, the symbol '!' can be used to negate a value, and the symbol NULL is also used for certain special cases. A typical example of the former would be translating the constraint "not flying on Thursdays .." by the descriptor *flight.flight_days* $\Leftarrow$ *!THURSDAY*.

8.3.2 Discussion of the Representation

As stated above, semantic representation languages for the ATIS task tend to lie somewhere along the spectrum between natural languages and database query languages. The semantic representation language used in CHANEL is more SQL-like than is the case for other ATIS systems. In particular, the language employs exactly the same table, attribute, and function names that will be used in the SQL query. Can this approach be defended?

The main argument against employing a representation that is closely related to the database query language and to the structure of the database is that it reduces portability of the system to other domains. This is a red herring. In many natural language systems, apparently general names for variables and values serve as a Potemkin village, concealing profound domain-dependence [Win86]. It would be surprising if any of the ATIS natural language systems ported easily to a new task, no matter what representation they use. Insofar as the representations used by other groups are genuinely domain-independent, this would not reduce the total amount of work required to port the systems using them to a new domain - work saved on the natural language component would have to be paid for by extra work on the SQL module that translates semantic representations into database queries.

There are reasons for thinking that CHANEL is actually **more** portable than most. For many potential applications, the only change required in the representation would be a change in the possible names of attributes, values, and functions. It would probably be necessary to rewrite the SQL module and the local parser, and the number and roles of the KCTs in the robust matcher would change. Since the KCT-growing algorithms are domain-independent, the only human work required to grow new KCTs would be that involved in collecting training data. Though this might be considerable, it seems unlikely that it would be less than that involved in writing a new set of parsing rules by hand.

Thus, there is no *a priori* reason to employ an apparently more general representation. On the other hand, there is at least one strong reason to employ an SQL-like representation. The principle behind the robust KCT-based matcher (as with AT&T's system) is that of maximizing the extent to which parsing rules are learned automatically from data. This principle would be completely fulfilled if we eliminated the semantic representation and built a system that learned rules for mapping word sequences directly onto SQL. This is impractical. However, we get close to it by employing an SQL-like representation and thus minimizing the amount of human expertise required to build the SQL module.

8.4 The SQL Module

Charles Snow of CRIM did most of the work on this module, and provided the following description.

The SQL module has two main tasks:

1. to convert a query expressed in the semantic representation language into a valid SQL query;

2. to manage the interaction with the Oracle relational database - i.e., to send the SQL query to the database and process the result.

The representation maps fairly straightforwardly into SQL. The module first builds the component of the query for the displayed attributes, then builds the list of constraints, processing constraints one at a time.

The list of constraints is assembled using a context stack wherein each bracketed set of input lines [] represents a distinct context. Within each context, the constraints are expressed as $(descriptor, value) = (d, v)$ pairs, e.g. $(flight.airline\_code, AA)$. d is a relation name followed by an attribute name, and v is either a scalar value or a function. Scalar values are implicitly related to the descriptor by an equality operator unless an explicitly provided operator (e.g., <, >, !, etc.) is supplied. For each of these, the SQL is generated to express the d and the way it is related to its corresponding v. This is straightforward except in cases where d is different from one or more relations in the list of displayed attributes (see below). If v is a function, it may be an SQL function such as MIN or MAX (the notation

identifies these functions by preceding them with '$') or a function proper to the representation. SQL functions are understood to be directly applicable to the corresponding d, and generally applicable throughout the context in which they appear.

In evaluating (d, v) pairs it is necessary to ensure that the resulting intermediate result will be join conformable with the displayed attributes; thus, the module is capable of generating joins as required. In cases where the displayed attributes are drawn from more than one relation, generating the join code becomes more involved.

There are accommodations for certain short-hand notations adopted for reasons of expediency. For example, the module allows an airport code attribute to be expressed as any of: 2-letter state code, 3-letter airport code, or 4-letter city code, and arranges always to provide a correct list of airport codes.

Interaction with the database is mediated by C-callable routines in the Oracle-supplied function library. The SQL module writes out a 'ref' file for each query expressing the results in ATIS Common Answer Specification (CAS) format, as well as providing a 'transcript' of the generated SQL with the values returned from Oracle.

8.5 The Local Parsing Module

Evelyne Millien of CRIM built this module, and kindly volunteered to write this description.

The local parser looks for words or phrases that are semantically important because they provide constraints that should be incorporated in the SQL query. These words or phrases are replaced by three-letter symbols in the version of the input sentence sent to the KCT-based robust matcher, which is not allowed to see the original words covered by the symbol. The meanings associated with each symbol are stored by the local parser, since they will have to be recovered to produce the SQL query.

For instance, the local parser would convert the sentence "DELTA FLIGHTS FROM BOSTON TO DENVER SERVING BREAKFAST" into "AIL flights from CIT to CIT serving MEA", the version seen by the robust matcher. The assignments $AIL = DL, CIT_1 = BBOS, CIT_2 = DDEN, MEA = breakfast$ will be stored until the robust matcher has made all its decisions

and is ready to generate the conceptual representation. In some cases, the order of symbols may be changed before the locally parsed sentence is sent to the robust matcher.

The local parser is based on a method combining two classes of unification grammars: lexical grammars [Dym90] and definite clause grammars (DCGs) [Per80]. A lexical grammar is composed of two main parts:

1. the lexicon, where the syntactic and semantic properties of words are described;
2. the rules describing the mechanisms of word combination, according to constraints given by the lexicon.

DCG rules are used mainly to define categories where no word predominates over others, such as flight numbers (sequences of digits) or codes (sequences of letters).

The parser carries out bottom-up chart parsing. The parsing algorithm is based on the original algorithm described in [Dym90], which was modified to parse sentences locally and in a robust way. In particular, the parser can skip over unknown words within a phrase. The semantic symbols appearing in the output of the local parser are:

- ACO - aircraft code;
- AFA - aircraft maker;
- AIL - airline;
- AIP - airport;
- ATY - aircraft type (e.g. Boeing 707);
- CCO - city code;
- CIT - city name;
- CLA - class (e.g. first class);
- DAY - day of the week;
- DAT - date;

- ECO - economy;
- FCO - fare code;
- FNB - flight number;
- MCO - meal code;
- MEA - meal;
- PRI - price;
- RCO - reservation code;
- ROU - round-trip;
- TIM - time of day.

Some examples of input to the local parser and the corresponding output and stored values:

- input: "ARE ANY OF THE FLIGHTS ON A BOEING SEVEN FIFTY SEVEN", output: "are any of the flights on a ATY", stored: $ATY = 757$;
- input: "PLEASE RESERVE UNITED FLIGHT ONE FORTY THREE", output: "please reserve FNB AIL", stored: $FNB = 143, AIL = UA$;
- input: "LEAVING SAN FRANCISCO INTERNATIONAL ...", output: "leaving AIP ...", stored: $AIP = SFO$;
- input: "BETWEEN THREE AND FOUR P M ON FRIDAY OCTOBER FIRST", output: "TIM on DAY DAT", stored: $TIM = (\geq 1500\& \leq 1600), DAY = friday, DAT = (92/10/1)$;
- input: "... WITH FARES TWO HUNDRED AND SIXTY EIGHT DOLLARS", output: "... with fares PRI", stored: $PRI = 268$;
- input: "WHAT DOES THE FARE CODE Q W MEAN", output: "what does the FCO mean", stored: $FCO = QW$.

Some examples of problems encountered:

- "I NEED A LATE FLIGHT FROM SAN FRANCISCO TO PHILADELPHIA ON OCTOBER NINTH" - according to the ATIS Principles of Interpretation, a "late flight" normally means between 8 pm and 3 am. In this example, the mention of the date implies the flight is between 8 pm and midnight; the local nature of the parser however prevents it from relating the word "late" to the date.
- "LIST THE AMERICAN AIRLINES FLIGHTS THAT USE A SEVEN THIRTY SEVEN" - the local parser may interpret "SEVEN THIRTY SEVEN" as a time or a flight number (a possible solution would be for it to consider the preceding article "a" or the word "use", instead of skipping over them).
- "HOW ABOUT SIX THIRTY P M" - since "HOW" is not analyzed, "ABOUT" modifies the time and the interpretation is $TIM = (\geq 1800\& \leq 1900)$ instead of $TIM = 1830$.

8.6 The Robust Matcher

8.6.1 Overall Structure

Figure 8.2 shows the internal structure of the KCT-based robust matcher in CHANEL. The matcher is made up of a large number of KCTs, each handling a different part of the task of translating a sequence of words into the appropriate semantic representation. Word sequences in CHANEL are processed by a *local parser* before being submitted to the matcher. However, an alternative, more complex architecture was also considered. Note from figure 8.2 that the robust matcher has two parts corresponding to the two parts of the semantic representation: the part that chooses the *displayed attributes* and the part that helps generate the *constraints*. Much of the work required to generate the constraints is done by the local parser: the part of the matcher that deals with constraints simply resolves some ambiguities left over after local parsing, on the basis of global information ignored by the local parser. Thus, the constraints part of the matcher must act after local parsing has been carried out. On the other hand, the part of the robust matcher that chooses the displayed attributes might have been invoked *before* local parsing. If this approach had been adopted, figure 8.2 would show a

word sequence first entering an **attribute-choosing module**, then the local parser, then a **constraint ambiguity resolution module**.

As described in Chapter 9, experiments were carried out to determine which architecture yielded the best results for ATIS data. It turned out that KCTs grown and tested on sequences that have been locally parsed, so that semantically important phrases have been replaced by generic symbols, are better at identifying the attributes to be displayed than are KCTs grown and tested on "raw" word sequences that have not undergone local parsing. Thus, the architecture shown in figures 8.1 and 8.2 was empirically determined to be better than the most plausible alternative.

For other domains of discourse, a more complex architecture might be preferable. It happens that in ATIS, the semantically important phrases - city names, times, dates, flight numbers, airline names, costs, and so on - can usually be recognised by the local parsing module independently of the global context. In other domains, local parsing decisions might depend on global information; for instance, on the topic of a sentence. In such domains, the system might include a KCT-based "router" that helped direct input to the appropriate local parsing module, as well as the main KCT-based robust matcher.

8.6.2 Choosing the Displayed Attributes

In the 1992 ATIS training data, the SQL translations of class A sentences contain 99 different attributes between an initial "SELECT DISTINCT" and the following "FROM". Attributes appearing in this position are called **displayed attributes**. A given query may have any number of displayed attributes, though rarely more than three. Among the 3254 SQL translations available at the time the robust matcher was built, *flight.flight_id* is the most common attribute, appearing in 2408 of the queries. Several of the 99 different displayed attributes in the class A data appear only once: an example is *city.time_zone_code*. Functions may also appear at the beginning of the SQL translation, though they are rare. In the class A SQL queries, 7 different function calls appear in this position: $MIN(flight.departure\_time)$ and 6 different calls to $count()$, of which $count(flight)$ is the most common. These 7 function calls were treated as if they were attributes.

To determine the set of displayed attributes for a particular word sequence, a separate set-membership KCT was grown for each of the 106 pos-

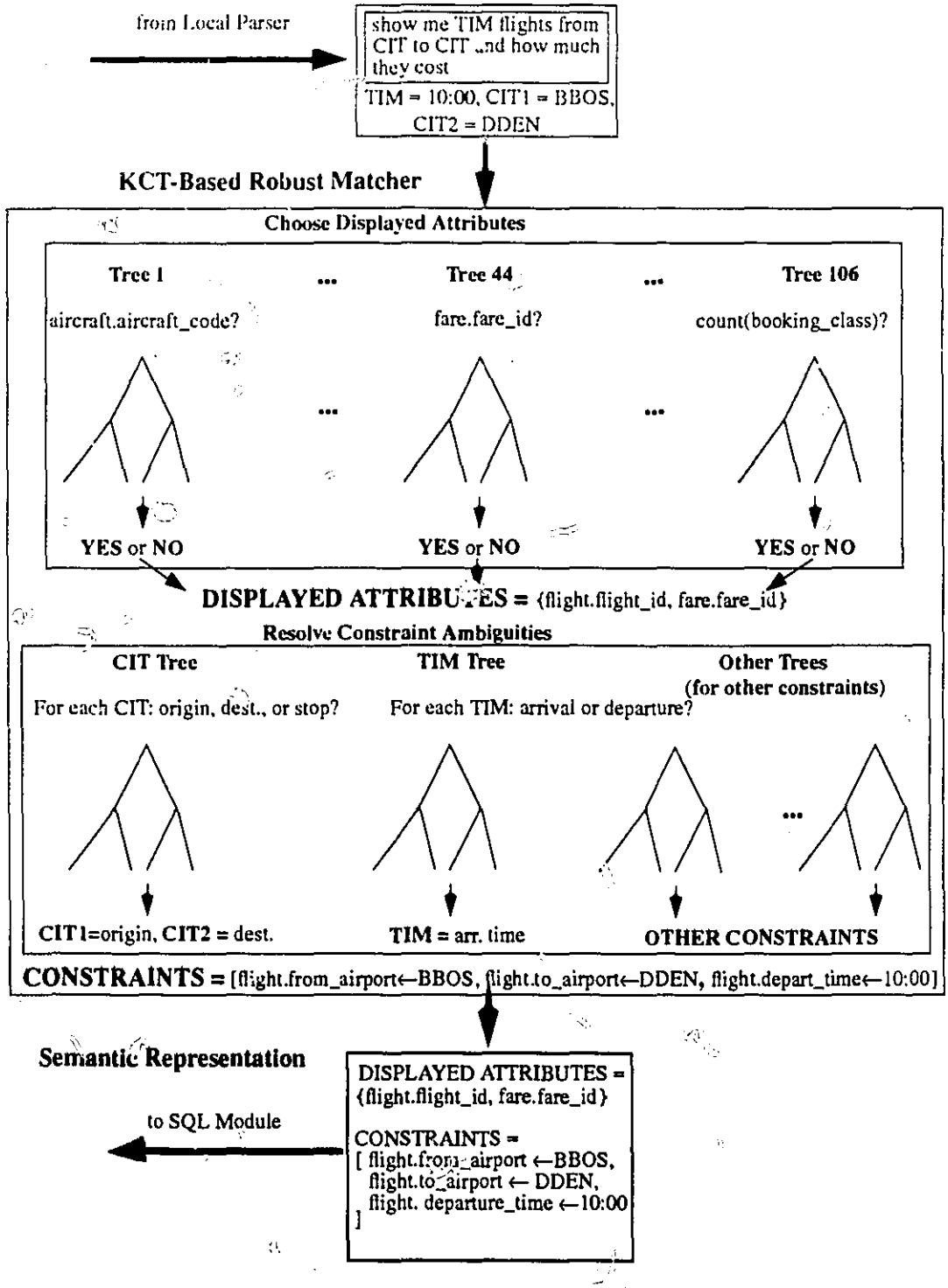


Figure 8.2: The KCT-Based Robust Matcher

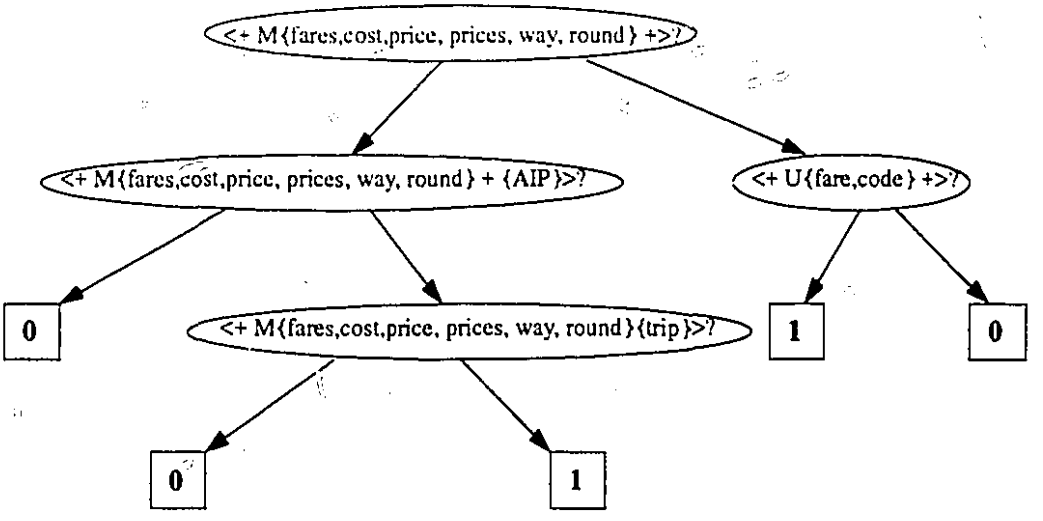
```
SHOW ME FLIGHTS FROM BOSTON TO DENVER => 0
ALL RIGHT WHAT I'D LIKE TO DO IS FIND THE CHEAPEST ONE WAY FARE FROM BOSTON
TO DENVER => 1
I WOULD LIKE INFORMATION ON GROUND TRANSPORTATION IN THE CITY OF ATLANTA
FROM AIRPORT TO DOWNTOWN => 0
SHOW ME ALL THE FLIGHTS BETWEEN DALLAS FORT WORTH AND EITHER SAN FRANCISCO
OR DENVER THAT DEPART BETWEEN FIVE AND SEVEN P M => 0
```

Example 5: Part of Training Data for *fare.fare_id* KCT

sible attributes and functions. Example 5 shows part of the training data used to grow the *fare.fare_id* KCT; sentences whose SQL translations display this attribute are labelled 1, and all other sentences are labelled 0. Each KCT decides independently whether or not its attribute appears in the set; thus, each of these 106 KCTs makes a simple YES-NO decision on the basis of the word sequence. Figure 8.3 shows a set-membership KCT that decides whether or not the attribute *fare.fare_id* should be displayed (this was not the KCT actually used, which is shown in the next chapter).

Note that each KCT can ask about any part of a word sequence, and that each is grown independently of the others. If different attributes are highly correlated, so that they are usually displayed together in the training data, their KCTs may strongly resemble each other. I have observed this for the three attributes *ground_service.city_code*, *ground_service.airport_code*, and *ground_service.transport_type* in the ATIS data. In an extreme case, two attributes that were always displayed together would yield identical KCTs. Thus, there may be a certain amount of redundancy in the set of KCTs; given the speed with which KCTs classify word sequences, this cannot be considered a serious disadvantage. Once these KCTs have been grown, it is easy to find the list of displayed attributes for an incoming word sequence: simply pass the sequence through the 106 KCTs and for each attribute, decide whether or not it should be displayed on the basis of the answer "YES" or "NO" obtained from the appropriate tree.

When originally designing the Robust Matcher, I believed that set-membership KCTs would turn out to be better classifiers than single-symbol KCTs. After the performance of a series of experiments described in the next chapter, it was clear that the two types of KCTs have roughly the same classification



Key

This tree determines whether attribute "fare.fare_id" should be displayed

1	means "fare.fare_id" displayed	"+" means 1 or more unknown words "M" means 1 or more words in the set must occur here
0	means "fare.fare_id" NOT displayed	"U" means exactly 1 of the words in the set must occur here

Example (after local parsing)	**Prediction**	**Correct**
(MON = sum of $$, FCODE = fare code, AIP = airport, CIT = city name)		
show me all flights that are less than MON round trip	**0**	**0**
what is the meaning of fare code FCODE and FCODE ***NOTE:** has both "fare" and "code", so no U{fare,code}	**0**	**0**
what is the cost of limousine service at AIP	**0**	**0**
what is the cost for a one way trip from CIT to CIT	**1**	**1**

Figure 8.3: Set-Membership KCT for *fare.fare_id* (grown on ATIS 2 data)

accuracy (at least for the amount of training data currently available). Since single-symbol KCTs can be grown overnight, while set-membership KCTs take much longer to grow, a decision was made to use single-symbol KCTs to pick the displayed attributes and functions for the Nov92 ATIS benchmarks. Details about these KCTs are given in the next chapter.

8.6.3 Classifying Constraints

Global Constraints

Some constraints can be considered as properties of the entire sentence. For these global constraints, we can grow KCTs that classify the entire string, like those that select the displayed attributes. The 7 global constraint KCTs in the benchmark were single-symbol. They were:

1. MAX and MIN for a fare ("cheapest", "most expensive" and so on);
2. MAX and MIN for arrival and departure times;
3. constraints on stops;
4. the fact that a meal should be served;
5. constraints on the nature of ground transportation desired;
6. the fact that the flight should be connecting;
7. MAX and MIN for aircraft capacity.

A constraint is included in this list if it tends to be spread out over a sentence instead of being localized. For instance, it would be difficult for the local parser to deal with "give me as low a fare for the morning flight to Atlanta as possible". Example 6 shows training data for the *time* global constraint KCT; the labels $1, 2, 3, 4, 5$ stand for default, $MIN(departure\_time)$, $MAX(departure\_time)$, $MIN(arrival\_time)$, $MAX(arrival\_time)$ respectively.

```
SHOW ME RETURNING FLIGHTS FROM BOSTON TO DENVER => 1
LAST PLANE INTO ATLANTA => 5
GIVE ME THE LAST FLIGHT OUT OF BOSTON TO CHICAGO  => 2
TO OAKLAND GETTING IN AS SOON AS I CAN => 4
```

Example 6: Part of Training Data for *Time* Global Constraint KCT

Local Constraints

The *constraints* component of the robust matcher must also deal with substrings identified by the local parser whose role is ambiguous. The section entitled "Classifying Substrings" of Chapter 6 gives as an example the identification of the role of a city name and shows how to grow a KCT that carries out this identification. For the Nov92 benchmarks, 3 of these local constraint KCTs were grown, one each for the codes AIP, CIT, and TIM. Apart from multiple-frame cases (below) an AIP or CIT can be an origin, a destination, a stopover, a site served by an airline, or a location for ground transportation; a TIM can be an arrival time or a departure time. The local constraint KCTs are set-membership, simply because there was no time left to switch to single-symbol KCTs after it was discovered these performed just as well.

Multiple frames were the biggest problem encountered in dealing with local constraints. Whenever an "OR" appears in the SQL translation of a sentence, the conceptual representation should contain two or more frames separated by the symbol |. Two examples of sentences that should generate multiple frames are: "show me ground transportation in Oakland, San Francisco and Denver" and "flights from Boston to Dallas or San Francisco or Atlanta". Note that an "and" in natural language may translate to an "OR" in SQL. To make matters worse, it is often unclear how other constraints in the sentence should be distributed across frames: do they apply to all of them or just to the nearest one?

A variety of labels was devised to cover these cases for the Nov92 version of CHANEL. For instance, in a normal single-frame sentence, the origin was labelled 1 and the destination 2 (each is a CIT or AIP). However, in a multiple-frame sentence with one origin and several destinations like "flights from Boston to Dallas or San Francisco or Atlanta", the origin was labelled

14 and each of the destinations 15. For the case with several origins and one destination, the labels were 16 for all origins and 17 for the destination. Another set of labels covers clauses where each contains an origin-destination pair; yet another set covers multiple-frame departure and arrival times.

In practice, multiple-frame cases encountered in the test usually yielded "NO ANSWER". This aspect of the system needs work. Fortunately, single-frame cases form the overwhelming majority of sentences, and the local constraint trees worked fairly well on these.

8.6.4 Meta-rules

The Nov92 version of CHANEL contained simple hand-coded meta-rules that looked for incompatibilities in the partially completed representation and returned "NO ANSWER" if they were found. They govern the following cases:

1. Clashing roles for global constraints. For instance, the user wants a flight to be the cheapest and also the most expensive (in theory this would be possible if there was a single flight meeting the other constraints, in practice it is a sign something has gone wrong with the interpretation).

2. Clashing roles for local constraints. For instance, two different cities are specified as the origin by the CIT local constraint KCT, or two different days of the week are specified as the departure day by the DAY local constraint KCT.

3. Incompatibilities between the displayed attributes and the constraints. For instance, the displayed attributes involve only the table *ground_service*, but a CIT in the sentence is classified as the *flight.from_airport*, or the whole sentence has the global constraint "latest" (the *ground_service* table does not contain temporal information).

Note that there are no meta-rules for mutually incompatible displayed attributes. In many cases where none of the meta-rules are invoked and a representation is produced, "NO ANSWER" is sent by the SQL module because it cannot perform a join invoked by the representation. As will be discussed in the final chapter, next year's system will probably contain more sophisticated meta-rules that are learned automatically from training data.

Chapter 9

Results

This chapter describes experiments carried out to determine which type of KCT yields the most accurate classification of sentences, and some aspects of KCT-growing (such as the number of iterations required to grow KCTs, and the dependence of KCT size on the amount of training data). It also describes the KCTs actually used by the current Robust Matcher, and the performance of this Robust Matcher on the November 1992 ATIS benchmarks.

9.1 Experiments with Different KCT Types

Recall that for each of 106 attributes and functions, an independent KCT is grown to decide whether the attribute should be in the list of displayed attributes for a given sentence or not. The availability of class A ATIS2 sentences, labelled in a way that showed for each attribute and function whether or not it was displayed, made it possible to perform experiments testing different types of KCT. In each of the experiments, 106 KCTs were grown on a subset of the ATIS2 class A data. The experiments illustrate the properties of KCTS - and, more important, show how to obtain optimal performance from KCTs.

9.1.1 Classification Accuracy

The 106 KCTs grown on a subset of ATIS2 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 MIN response prescribed by DARPA. This means that if 105 of the KCTs make the correct yes-no decisions about whether the attribute for which each is responsible should be chosen for a given sentence, but the remaining KCT makes the wrong decision, we have "failure". Thus each of the experimental results below represents the combined output of 106 KCTs of a given type.

KCTs for displayed attributes can vary along several orthogonal dimensions:

- Trained on a small amount of data vs. trained on a large amount of data;
- Single-symbol vs. set-membership;
- Trained on and inputting partially parsed data vs. inputting pure word sequences;
- Trained on "well-shuffled" data vs. trained on "lumpy" data;
- For the KCTs that will handle recognizer output, trained on NL data (transcripts) vs. trained on word sequence hypotheses output by the recognizer.

Figures 9.1 and 9.2 show how classification accuracy for sentences depends on these factors. Before beginning this set of experiments, my expectations were that:

1. More training data would improve performance;
2. Set-membership KCTs would perform better than single-symbol KCTs;
3. KCTs grown to handle sentences already processed by the local parser would perform better than KCTs whose input is raw word sequences;
4. Training data should be similar to test data for good performance, but the exact composition of training data is not vitally important;
5. KCTs trained on recognizer output would perform better on such output than KCTs trained on transcripts.

Two of these five assumptions were correct, three incorrect:

1. Performance does improve with amount of training data;

2. Set-membership KCTs may not classify more accurately than single-symbol KCTs;

3. Preprocessing by the local parser does increase accuracy;

4. It is important that the training data resemble as closely as possible a random sample from the population of sentences to be encountered, and these data should also be split up between the two files required by the iterative expansion-pruning algorithm in a random way;

5. Transcript-trained KCTs seem to perform better than recognizer-trained KCTs on speech recognizer output.

Figure 9.1 illustrates the first four points, using KCTs trained and tested on NL data (transcripts). For all KCTs except those in the curve marked "lumpy", the test data consisted of 542 ATIS A sentences - parsed for the KCTs trained on parsed data, unparsed for the other KCTs. The KCTs whose training data and input are preparsed use a slightly different lexicon from the other KCTs, since they input sentences containing codes such as AIP, CIT, TIM, and so on as well as ordinary words. The KCT-growing algorithms treat these codes in precisely the same way as other words.

First, note that the figure contains an encouraging message: as the amount of training data grows, performance can be expected to improve. Except for the curve labelled "lumpy", there is no indication that any of the KCT types shown are approaching an asymptote. Indeed, 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 KCT-based approach.

The curve labelled "lumpy single-symbol unparsed" requires explanation. Originally, the ATIS2 NL training data were split up into 25 sequential chunks, each containing about 4% of the data. The two training data files and the test data file for the iterative expansion-pruning algorithm were made up of random combinations of the chunks. For instance, one of the data points on the "lumpy" curve was obtained by training a single-symbol KCT with one file consisting of chunks 7, 11, and 4, and the other file consisting of chunks 9, 21, and 3. All other curves shown in the figure were trained and tested on concatenations of files obtained by taking every nth sentence in

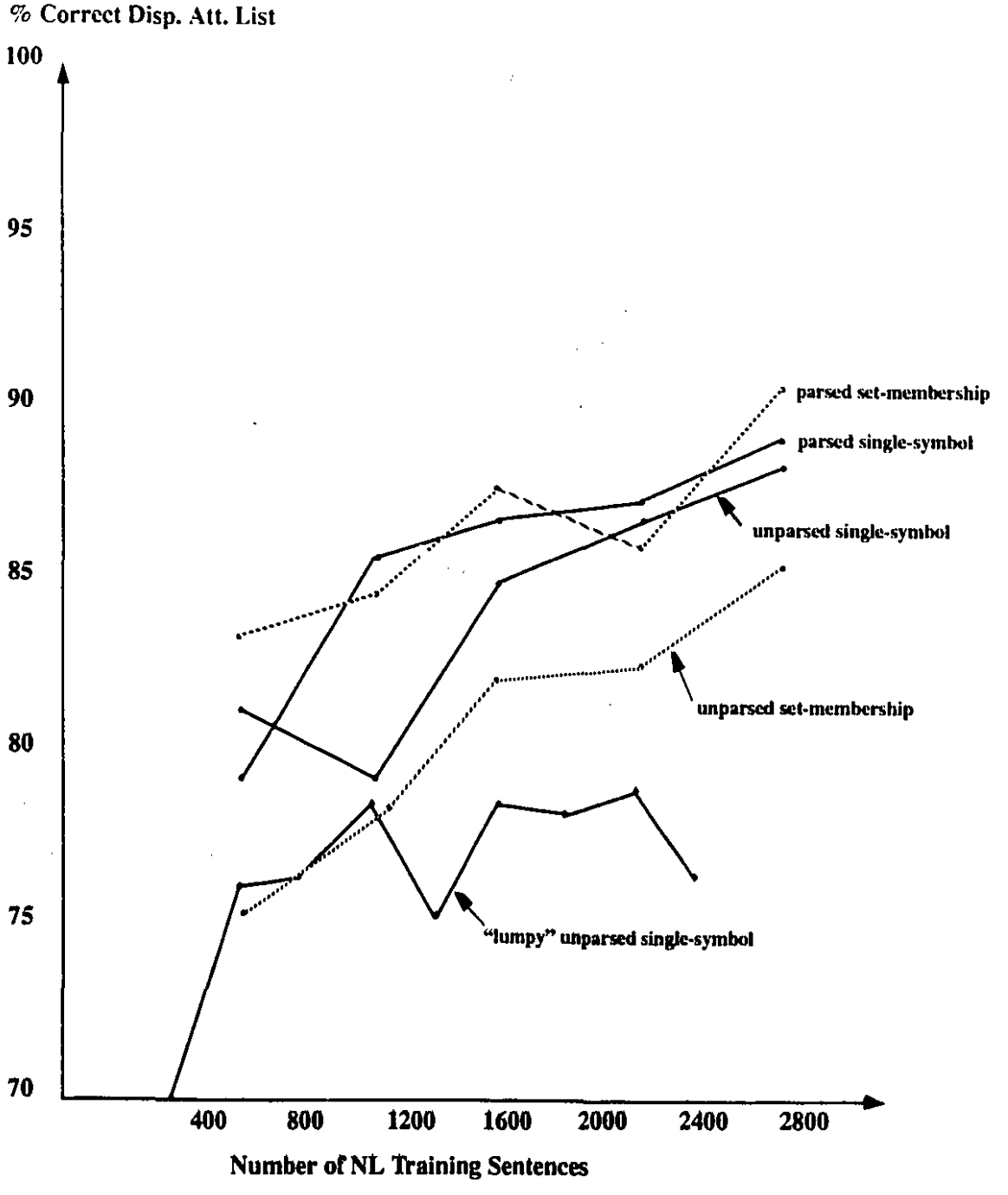


Figure 9.1: Classification Accuracy on NL Data for Various KCT Types

the training data. Comparison between the "lumpy" curve and the other "unparsed single-symbol" curve show that the latter is a considerably better classifier; other experiments not shown here confirm this trend. Both forms of training involve random sampling - why the difference between the final results?

The ATIS2 sentences are ordered by speaker. Thus, when they are split up into sequential chunks, there is a good chance that most of the class A utterances of a given speaker will remain together in the test file or in one of the training files for the KCT that yielded the "lumpy" curve. The other, "smooth" method for splitting up data makes it certain that adjacent utterances will be separated. Apparently, "lumpy" training data may cause KCTs to learn rules that are biased in favour of idiosyncratic expressions used by speakers who are over-represented in the training data; "lumpy" test data may exaggerate the effect of this problem by presenting the KCTs with several instances of totally new expressions that are over-represented in the test data. A good analogy is opinion polling, where reliable results can be obtained only if the sample is as close to random as possible; a large sample with a slight bias is a worse predictor of general trends than a much smaller, truly random sample.

Figure 9.1 confirms that it is a good idea to carry out local parsing before KCTs classify the input sentences. It is less clear whether single-symbol or set-membership KCTs are better classifiers. The latter take much more time to grow. Thus, I decided that the KCTs for choosing displayed attributes in the November 1992 benchmarks would be single-symbol KCTs trained on parsed, "smoothly" chosen sentences. Another important decision still had to be made: for the SLS benchmark, was it better to train the KCTs on recognizer output or on NL data? KCTs grown on recognizer output differ somewhat from their NL-trained counterparts (figures 9.4 and 9.5 in the next section give an example).

Figure 9.2 illustrates the performance of single-symbol KCTs on the top hypothesis from the speech recognizer, after the latter has undergone local parsing. The comparison is between KCTs trained on speech output (files containing labelled, parsed versions of the top hypothesis) and KCTs trained on transcripts. The results in this graph are **not** comparable to the results shown in figure 9.1, because they exclude word sequence hypotheses for which the local parser yielded "NO ANSWER". The local parser tends to discard an appreciable proportion of the speech output in this manner; if discarded

sentences were counted as wrong answers, the percentages correct in figure 9.2 would be somewhat lower. Disappointingly, the graph seems to show better performance for the transcript-trained KCTs. This seems to refute the hypothesis that the speech-trained KCTs have learned error-correcting rules. However, more sophisticated methods of training the KCTs might still accomplish this goal; the next chapter suggests possible approaches.

9.1.2 Properties of KCTs

Surprisingly, almost all KCTs irrespective of final size required three or fewer cycles of the expansion-pruning algorithm to converge. It would be interesting to try to prove tight upper bounds for the number of iterations required by the algorithm.

To illustrate the growth of KCTs as a function of training text size, two of the 106 displayed attributes were selected. Figure 9.3 shows the size of the single-symbol and set-membership KCTs grown on NL data to deal with *fare.fare_id* and *flight.flight_id.* As the amount of training data grows, the number of nodes tends to increase, but there are times when added data causes little growth or even shrinkage of the KCT. These pauses in KCT growth can be viewed as particularly successful generalization steps in the learning process - the KCT has just replaced a complex rule with a more parsimonious rule that covers more data.

9.2 The November 1992 ATIS Benchmarks

9.2.1 KCTs Grown for the November 1992 Robust Matcher

The single-symbol KCTs for displayed attributes and the single-symbol KCTs for global constraints were grown on 3248 class A ATIS2 NL sentences; the set-membership constraint KCTs were grown on the same sentences, plus 687 class A and D February 1992 NL sentences. Note that when the local constraint KCT for a given grammatical code is trained, each copy of the code counts as a single item of training data. For instance, the set-membership KCT for CIT (city names) was trained on 3935 class A and D sentences, but since there was an average of under two occurrences of CIT per sentence, it

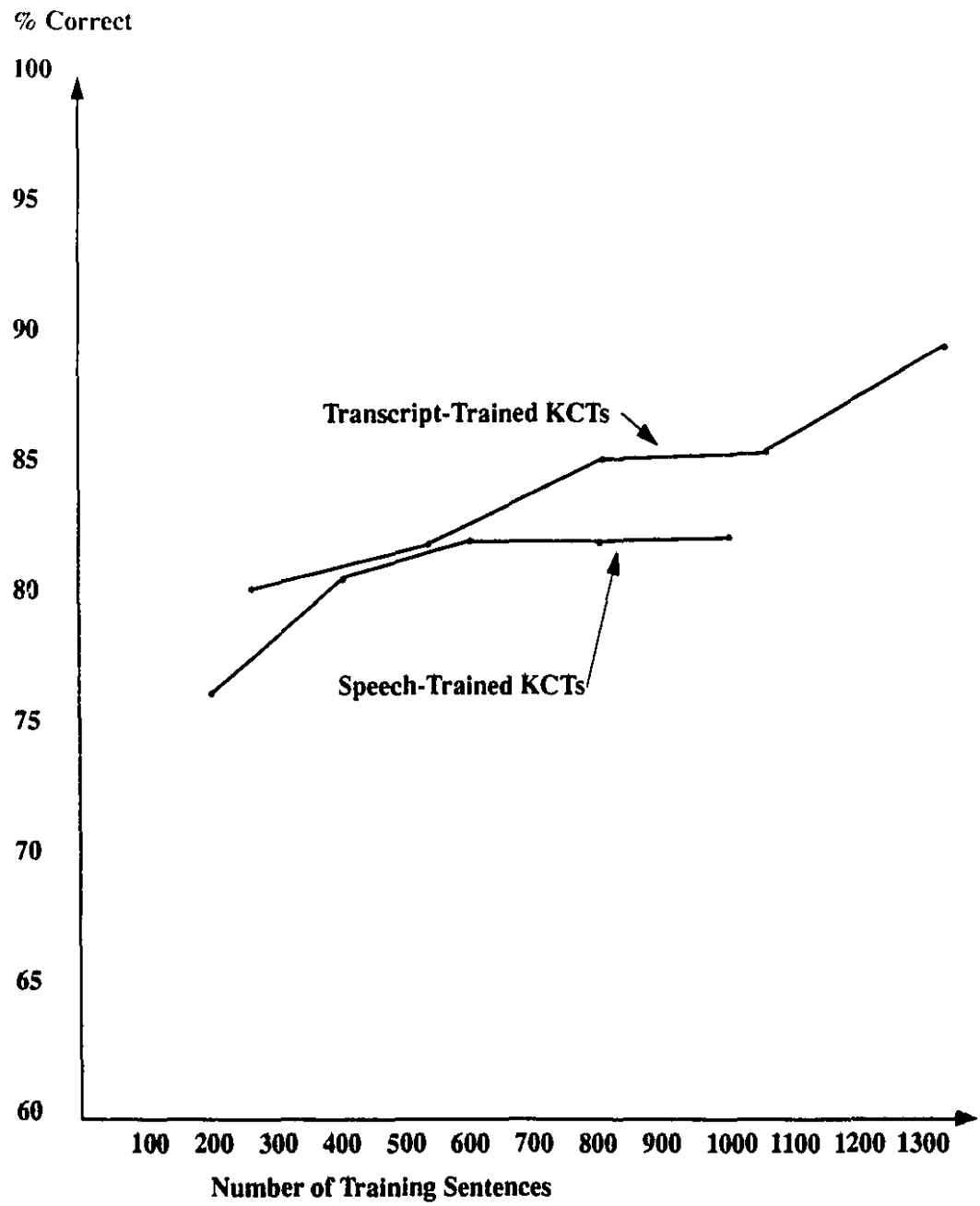


Figure 9.2: Single-Symbol KCTs for Displayed Attributes Tested on Parsed Speech Data

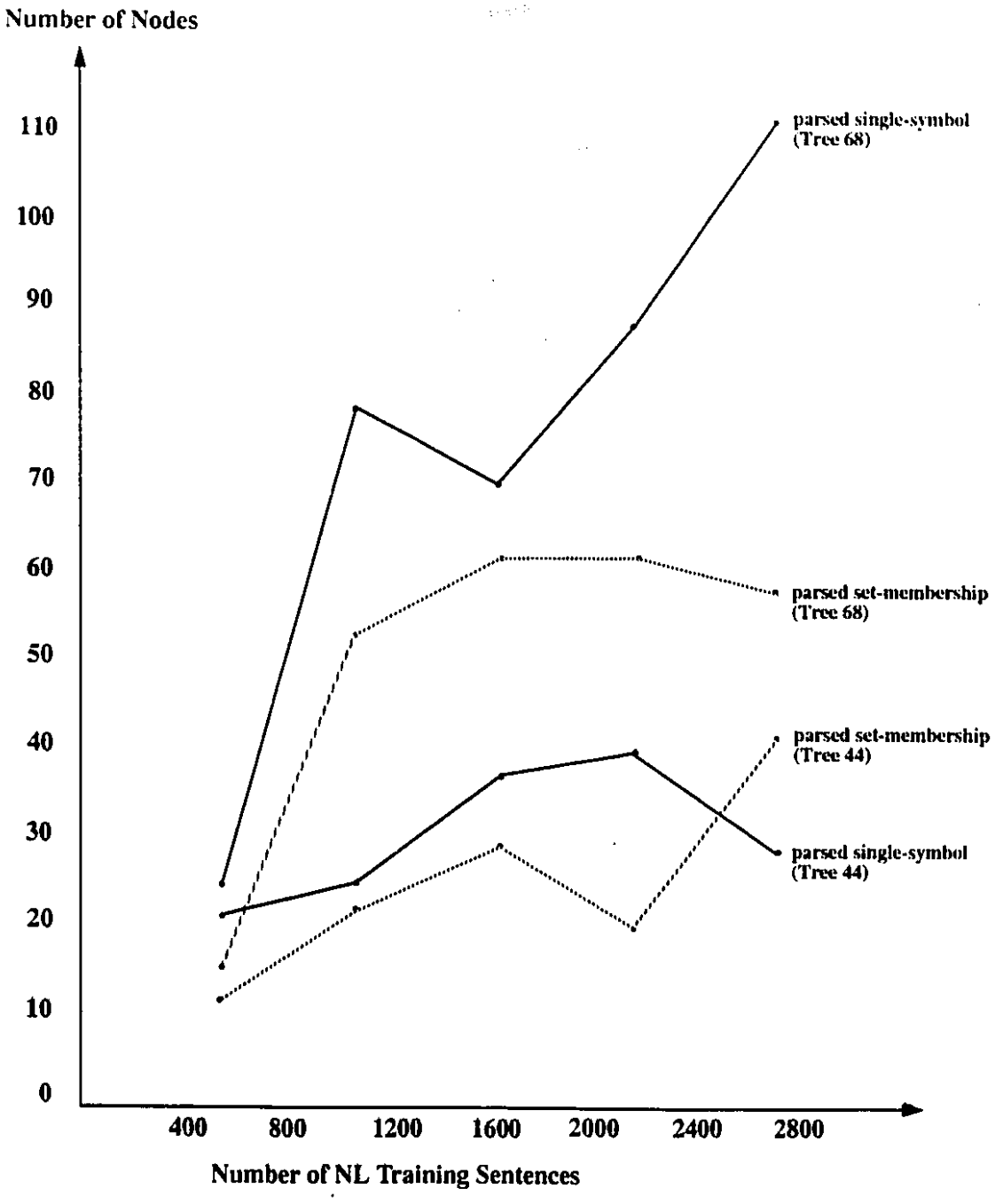


Figure 9.3: KCT Sizes vs. Size of Training Data (Tree 44 = *fare.fare_id*, Tree 68 = *flight.flight_id*)

is closer to the truth to say the training data consisted of 7028 occurrences of CIT. Similarly, the training data for AIP (airport name) consisted of 221 occurrences of AIP, and the training data for TIM (times) consisted of 980 occurrences of TIM.

The 106 KCTs for displayed attributes ranged in size from 1 node to 127 nodes. The two largest, the KCTs for attributes *fare.fare_id* and *flight.flight_id*, had 37 and 127 nodes respectively. The 7 KCTs for global constraints were small; the smallest had 5 nodes, the largest 15 nodes. The set-membership local constraint KCTs have the following sizes: the AIP KCT has 25 nodes, the CIT KCT has 157 nodes, the TIM KCT 59 nodes.

Figure 9.4 shows the *fare.fare_id* displayed attribute KCT grown on NL data and used in the benchmarks; figure 9.5 shows a KCT grown on the most probable output of the recognizer for the same utterances. Sentences that end up in a YES leaf will put *fare.fare_id* in the displayed attribute list, while the rest will not. The expression $M(w)$ matches one or more occurrences of the word w; the question "< 11?" means "does the sentence have 11 or fewer words?" Note that near the root, the two KCTs are very similar - the differences between them consist of minor rearrangements of the same questions. The internal nodes tend to have more descendants in their NO subtree than in their YES subtree, as is characteristic of single-symbol KCTs.

9.2.2 Benchmark Results

The November 1992 ATIS benchmark results are shown in figure 9.6-9.8. Two CRIM systems are shown: CHANEL and another one called NEURON. Since CHANEL was designed for and trained on class A utterances, class D and X results have been omitted. A context-dependent version of CHANEL for class D was created at the last moment, but since this work lies outside the parameters of the problem described in this thesis and was not done by me, results for this version have also been omitted; the D utterances will be tackled seriously next year.

Figure 9.9 shows the systems ranked by a simple measure of robustness (which is **not** one of the official benchmarks). This measure is the ratio of NL weighted error to SLS weighted error, divided by the proportion of correctly recognized words. The rationale is that for a fixed NL to SLS error ratio, systems that achieve this ratio with a higher percentage of incorrectly

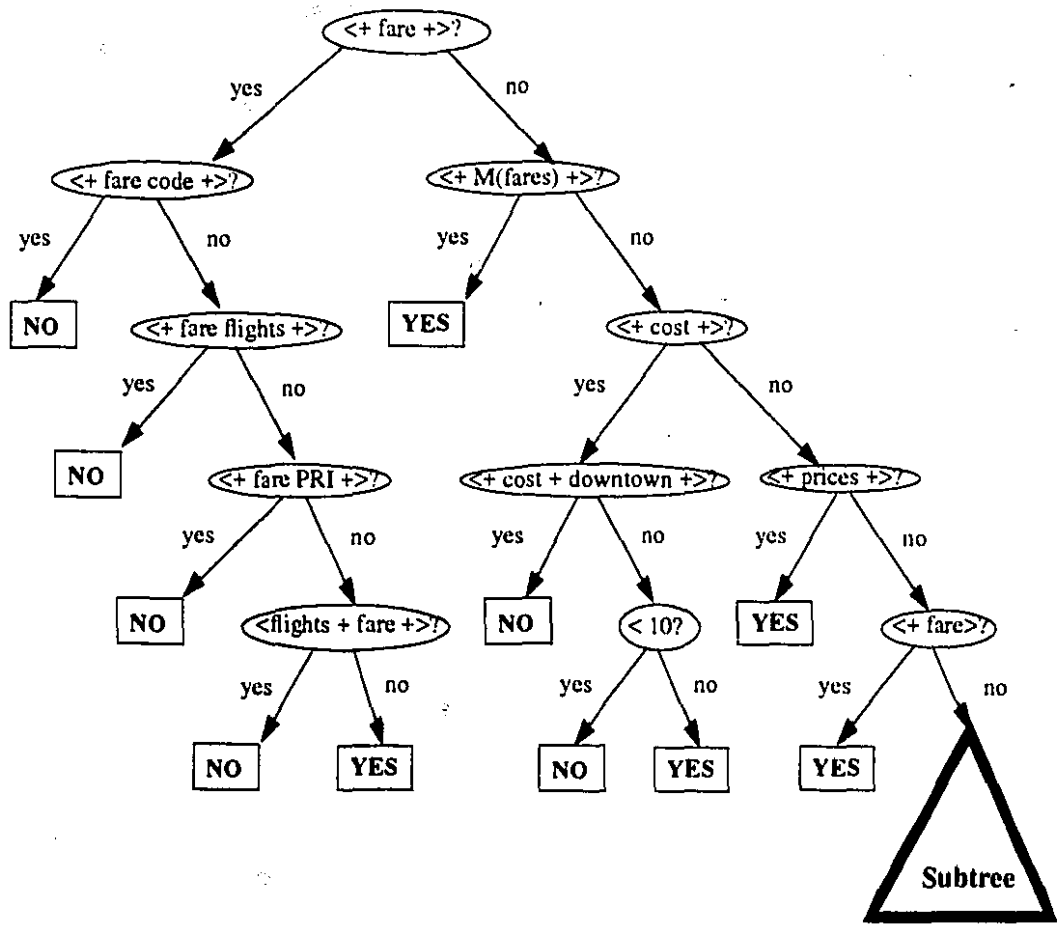


Figure 9.4: Single-Symbol KCT for *fare.fare_id* trained on NL Data (number of nodes = 37)

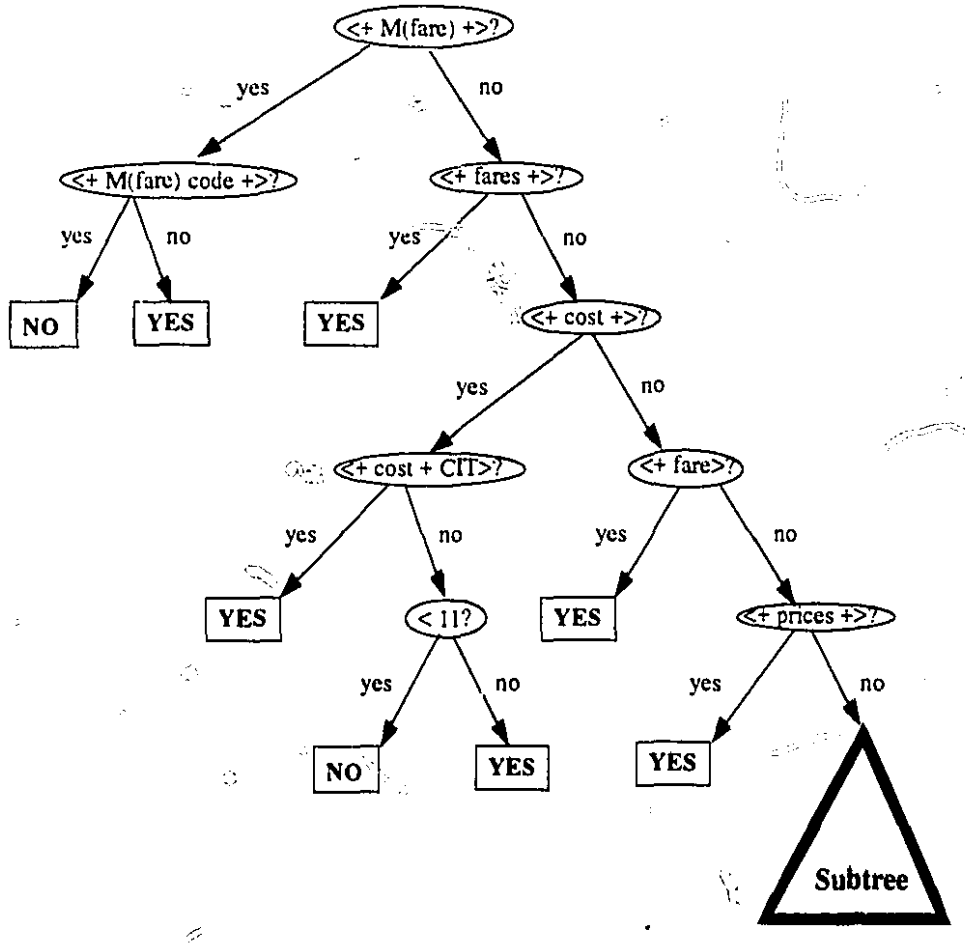


Figure 9.5: Single-Symbol KCT for *fare.fare_id* trained on Speech Data (number of nodes = 33)

System	#True	#False	#NA	#Utt	Weighted Error
ATT	343	64	20	427	34.7
BBN	379	19	29	427	15.7
CMU	399	24	4	427	12.2
CHANEL	298	44	85	427	40.5
NEURON	356	62	9	427	31.1
INRS	244	158	25	427	79.9
MIT-LCS	380	31	16	427	18.3
PARAMAX	278	39	110	427	44.0
SRI1	362	30	35	427	22.2
SRI2	383	19	25	427	14.8

Figure 9.6: November 1992 ATIS NL Test Results (Class A only)

System	%Corr	Sub	Del	Ins	Err	%Utt. Err
ATT	93.8	4.4	1.8	1.8	8.0	45.4
BBN	96.7	2.3	1.0	0.8	4.0	25.3
CMU	96.1	2.7	1.2	0.5	4.4	30.7
CRIM	88.9	8.0	3.1	2.4	13.5	57.8
INRS	0.0	0.0	100.0	0.0	100.0	100.0
MIT-LCS	93.5	4.4	2.2	1.3	7.8	38.2
SRI	96.0	3.2	0.9	1.1	5.2	34.2

Figure 9.7: November 1992 ATIS SPREC Test Results (Class A only)

System	#True	#False	#NA	#Utt	Weighted Error
ATT	303	88	36	427	49.6
BBN	361	35	31	427	23.7
CMU	383	40	4	427	19.7
CHANEL	249	65	113	427	56.9
NEURON	296	113	18	427	57.1
INRS	0	0	427	427	100
MIT-LCS	347	50	30	427	30.4
SRI1	337	46	44	427	31.9
SRI2	352	38	37	427	26.5

Figure 9.8: November 1992 ATIS SLS Test Results (Class A only)

System	NL W. Err.	SLS W. Err.	SPREC Corr.	Robustness
ATT	34.7	49.6	0.938	0.75
BBN	15.7	23.7	0.967	0.69
CMU	12.2	19.7	0.961	0.64
CHANEL	40.5	56.9	0.889	0.80
NEURON	31.1	57.1	0.889	0.61
MIT-LCS	18.3	30.4	0.935	0.64
SRI1	22.2	31.9	0.960	0.72
SRI2	14.8	26.5	0.960	0.58

Figure 9.9: Results for $NL\ W.\ Err/(SLS\ W.\ Err. * SPREC\ Prop.\ Corr.)$

recognized words show greater robustness. By this measure, the CHANEL system is the most robust; interestingly, the AT&T system, whose parameters were also automatically trained, is the second most robust. Of course, the measure may unduly favour systems with a high percentage of misrecognition; CHANEL's robustness cannot be fully evaluated until the group's NL and SPREC results are closer to the median results for other groups.

Overall, the results are quite satisfying. Almost all the groups represented have participated in several previous ATIS evaluations; the linguistic analyzers developed by these groups represent a large amount of code and a large number of person-years. Except for the AT&T system, none of the non-CRIM systems involve a machine learning approach. While development

of the KCT-growing algorithms and programs took a considerable amount of time, applying them to ATIS took relatively little time. If one excludes work on the SQL module, the development of CHANEL for ATIS took two people (myself and E. Millien) 3.5 months (August to mid-November 1992). It is the rules learned by the KCTs that made this rapid development possible. Most of the global sentence-level knowledge is locked up in them, so that the 4000 lines of 'C' code in the Robust Matcher module are mainly concerned with manipulating the output yielded by KCTs on input sentences.

The experiments shown in figure 9.2 demonstrate that for ATIS, transcript-trained KCTs tested on recognizer output perform better than recognizer-trained KCTs. For this reason, the version of CHANEL used in the benchmarks consisted entirely of transcript-trained KCTs. After submission of the benchmark results, a version of CHANEL with recognizer-trained displayed attribute KCTs (still using transcript-trained constraint KCTs) was tested informally on the SLS class A benchmark data. The result confirmed the wisdom of the decision to train on NL data: the partly speech-trained version of CHANEL had weighted error of 60.1 (instead of 56.9).

9.3 Analysis of Errors

All errors discussed here are for class A. The system produced 129 wrong answers (F plus NA) on the NL benchmark and 178 wrong answers on the SLS benchmark. 115 of the 427 sentences yielded the wrong answer on both benchmarks, 63 sentences that yielded the correct NL answer gave the wrong SLS answer, and 14 sentences managed to yield a wrong NL answer but a correct SLS answer (i.e. a somewhat distorted version of the sentence produced by the recognizer yielded the correct answer, even though the correct version yielded the wrong answer).

9.3.1 Analysis of NL Errors

Many of the errors made by the robust matcher on the November 1992 NL benchmarks involved an incompatibility between the displayed attributes and the constraints. For instance, questions about the *ground_service* relation should never involve other relations. Nevertheless, the current robust matcher sometimes produces a conceptual representation whose displayed

attributes mention the *ground_service* relation but whose constraints mention other relations. For the sentence "tell me about ground transportation to DDEN", the matcher might wrongly classify DDEN as a *flight.to_airport* despite having generated the correct list of displayed attributes (all drawn from *ground_service*). This kind of error arises because KCTs are independent: each KCT produces an answer that does not take into account the answers output by other KCTs. The next chapter will suggest some solutions to this problem.

Figure 9.10 breaks down the NL errors in more detail. If a sentence gives rise to more than one error, all are included in the histogram; thus the sum of errors shown is larger than the number of wrong answers (F plus NA). First, consider the non-KCT errors. The most important type of non-KCT error is labelled "DAT": because of an incompatibility between the local parser's encoding of dates and the SQL module's decoding of dates, most sentences that included them yielded an incorrect SQL query. This simple bug has been fixed. Another omission that will be easy to fix involved two codes, "ODT" and "LEG", connected with time intervals that overlap midnight and flight legs respectively. There was a deeper problem with the SQL module that caused it to send "NO ANSWER" when displayed attributes were drawn from different relations; this may require rewriting of parts of the module. The cases shown under "local parser problem" are straightforward, often involving words missing from the lexicon - such as a "red-eye" flight and a "turboprop" airplane. Thus, most of the non-KCT errors can be remedied fairly easily.

The trickiest type of error that arises in the KCT part of CHANEL involves multiple frames in the representation; recall that these are required to generate an "OR" in the SQL query. Though CHANEL's training data included several different multiple frame cases which were labelled as such, the system did not handle these cases well in the test data. Perhaps the approach was correct and there were simply too few multiple frame examples in the training data for the KCTs to learn good rules. However, I tend to believe that this problem calls for a different approach (as yet undetermined).

The remaining error types do not pose deep problems. The number of displayed attribute errors is high; however, determining the list of displayed attributes is a difficult problem. The KCT experiments described earlier give every reason to believe that more training data will reduce the frequency of this type of error. Close examination of these errors confirms this view.

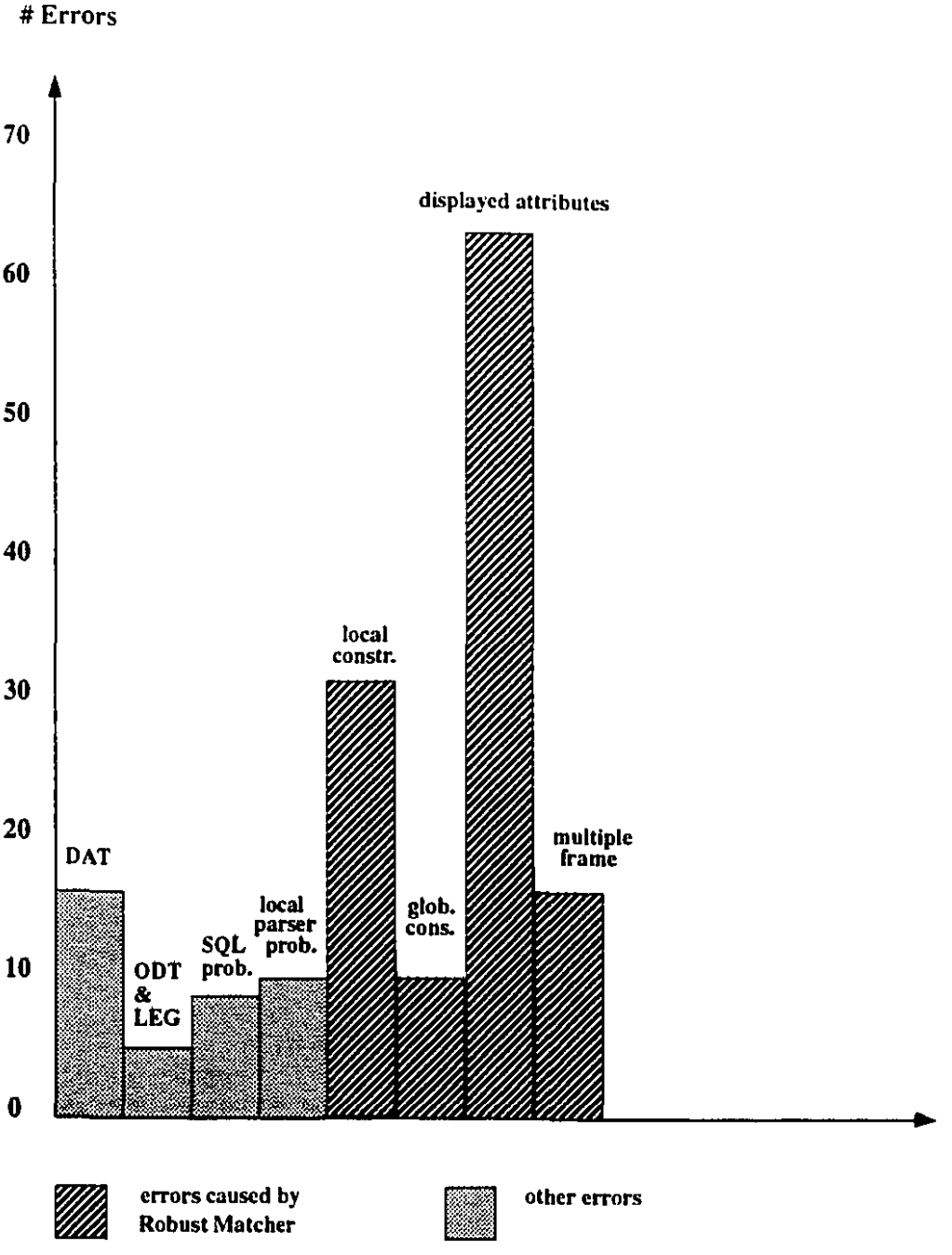


Figure 9.10: Histogram of NL Errors

For instance, 25% of them occurred when the sentence contained the word "airfares", which had occurred exactly once in the training data but was frequent in the test data. When this word appeared without other fare-related words, the *fare.fare_id* KCT output "no" instead of "yes". Clearly, more training data would have eliminated all these errors.

The same applies to errors in the local and global constraints. Furthermore, as described in the next chapter, it should be possible to reduce overall error by implementing automatically generated meta-rules that will ensure mutual consistency among the displayed attributes and constraints.

9.3.2 Analysis of SLS Errors

Benign SLS Errors

"Benign" recognition errors cause the incorrect word sequence hypothesis output by the recognizer to yield the correct conceptual representation in the SLS benchmark, even though the corresponding NL transcript containing the correct word sequence yielded an incorrect representation. Of the 427 class A sentences, 14 experienced benign errors and 63 experienced malign errors. Thus the number of sentences with benign recognition errors is surprisingly high: almost 25% of the number of malign errors. Study of the 14 sentences with benign errors leads me to suspect that the phenomenon arises partly because the language model and the KCTs were trained on the same data. In many cases of misrecognition, the language model "edits" the true word sequence to produce a hypothesis that more closely resembles sentences found in the training data. This process of "editing" may help the KCTs, which work best on sentences similar to those in the training data. For instance, the true NL word sequence may contain a rare word that confuses the KCTs; in the SLS version, this word may be "edited out" by the language model, allowing the KCTs to yield the correct answer.

Malign SLS Errors

"Malign" recognition errors cause utterances whose NL version yielded the correct representation to yield an incorrect SLS answer. Among the 63 class A sentences that yielded the correct NL answer but a false answer or "NO ANSWER" on the SLS benchmark, I have identified 88 errors. These are

broken down by category in figure 9.11. The "local parser" errors involved cases where the local parser could conceivably have recovered from a recognition error (for instance, the string "December mean 11th" could conceivably have been correctly parsed as the date "December the 11th"). The "local constraint", "global constraint", and "displayed attribute" errors might be reduced by carrying out some training of KCTs on SLS as well as on NL, using one of the methods suggested in the next chapter.

The "hopeless" errors are ones where the best hypothesis produced by the recognizer is so misleading that recovery is impossible; they involve deleted, inserted, or substituted constraint values. There are surprisingly few cases of substitution within the same grammatical category - where, for instance, the name of one city is replaced by the name of another city. Instead, most of these errors involve either deletion of a constraint value or the insertion or substitution of a grammatical category that does not occur in the NL version of the sentence. Often, the new category is one that seldom co-occurs with the other constraints, or with the displayed attributes.

This observation offers some hope for an improved SLS version of the robust matcher. Above, we mentioned the possibility of automatically generating meta-rules for spotting incompatible combinations of displayed attributes and constraints. As will be discussed in the next chapter, such meta-rules for SLS would support a strategy in which not just the top hypothesis, but the N-best hypotheses, could help to provide the conceptual representation. This would help not only with the "hopeless" type of error, but also with the "local constraint", "global constraint", and "displayed attribute" error types.

Examining this possibility will be a high priority for the coming year; however, it will not be the highest priority. Note that of 178 sentences that did not yield the correct answer in the SLS benchmark, 115 also failed to yield the correct answer in the NL benchmark. Thus, working to improve the robust matcher for NL will be the single most important way of helping SLS performance.

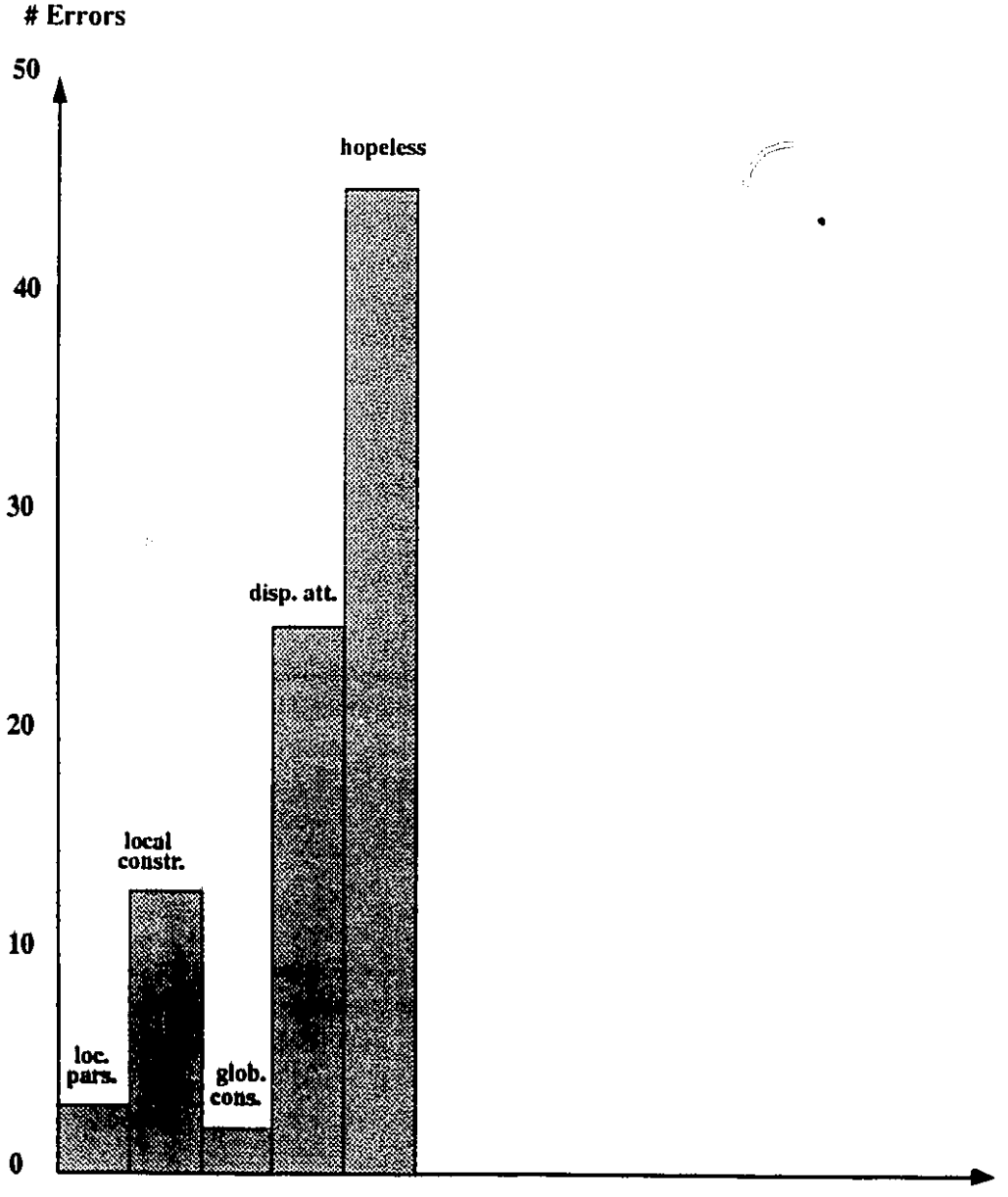


Figure 9.11: Histogram of SLS Errors without corresponding NL Errors

Chapter 10

Discussion

10.1 Original Contributions of the Thesis

This thesis makes two original contributions:

1. Keyword Classification Trees, a new method for learning and classifying sequences of symbols. Because KCTs explicitly model gaps, the task of learning general patterns from training sequences becomes much easier in certain domains. The method is particularly useful in the analysis of word sequences.

2. The KCT-based Robust Matcher, a linguistic analyzer for speech understanding that combines a chart-based local parser with a global analyzer that uses a large number of KCTs (over 100 for the ATIS task) to carry out semantic analysis. ATIS was used as a testbed for the KCT-based Robust Matcher; however, this kind of robust matcher should be widely applicable in text and speech understanding.

For each of these contributions, the thesis describes related work by others:

1. Chapter 6 discusses a tree-based language model employed by IBM, and a classification tree approach to information retrieval devised by Crawford, Fung *et al.* Neither of these approaches has been applied to speech understanding. However, these approaches also differ from the KCT approach in a more profound way. The IBM trees only ask

questions about fixed positions in a sentence, e.g. the identity of word 11. By contrast, KCTs can ask about regions of the sentence defined in terms of words known to be present, e.g. the identity of the words between the start of the sentence and the word "flight". The Crawford-Fung trees are much more limited than the other two: they can only ask whether certain words are in the sentence or not, and cannot ask about relative order of words at all.

2. Chapter 3 discusses the linguistic analyzers in several speech understanding systems. Many of them contain robust matchers: linguistic analyzers that grab semantically important phrases and stick them into appropriate slots. These matchers incorporate a very relaxed notion of syntactic correctness, and skip over irrelevant words; all are hand-coded. The KCT-based robust matcher adopts the same overall philosophy as other robust matchers, but most of its rules are grown from data rather than hand-coded. The only linguistic analyzer for the ATIS task that contains automatically generated rules is the one built at AT&T. The AT&T system learns rules for segmenting a sentence into "concepts"; there is a one-to-one mapping between segments and units of meaning. Each KCT in the KCT-based robust matcher looks at the whole sentence; thus, the use of KCTs permits information in a given segment of the sentence to play a role in building several different parts of the conceptual representation.

10.2 Advantages and Disadvantages of KCT-Based Robust Matcher

10.2.1 Advantages

The main advantages of using KCTs to build a robust matcher are:

- The savings in human effort, and potential improvement in performance, achieved by growing rules from training data instead of hand-coding them.
- Speed: KCTs process sentences very quickly, that is in time linear in the length of the sentence; they also take up very little storage.

The surprising result that single-symbol KCTs seem to perform about as well as set-membership KCTs means that growing KCTs for the matcher can be done fairly quickly. For instance, the set of 106 single-symbol KCTs for choosing displayed attributes in the ATIS robust matcher was grown on about 3250 training sentences in less than 10 hours (on a Sun SPARC-2 with 28 MIPs).

- Because KCTs ask about only a few words in a sentence, recognition errors in the remaining words do not affect performance.

- Increased portability: if the database structure, the database language, or the task domain are changed, the part of the robust matcher consisting of KCTs can be adapted quickly by regrowing the trees on new data. The new data could be obtained via bootstrapping, beginning with a Wizard of Oz scenario for collecting the first 500 or so sentences. The results in the previous chapter for the ATIS task show that while a *high* level of performance can only be attained by training on $2000 - 3000$ sentences, a *reasonable* level of performance for a KCT-based system can be attained by training on a few hundred sentences. Thus a prototype trained on 500 sentences would be good enough to gather more data (with a wizard standing by); after a few iterations, an excellent system would be available.

- Unlike neural nets, KCTs learn regular expressions made up of words and gaps which can easily be interpreted by people, and used in different contexts. For instance, one could design a recognizer whose lexical search algorithm was guided by KCT-generated expressions.

- The potential error correction achieved by growing KCTs on recognizer output. In the case of ATIS, transcript-trained KCTs outperformed recognizer-trained KCTs. As described below, new experiments using ATIS data will be carried out to see if the potential for SLS error correction may be realized by using more sophisticated training methods. In any case, it is possible that non-ATIS tasks exist for which recognizer-trained KCTs will perform better.

10.2.2 Disadvantages

The main disadvantages (potential or actual) of the KCT-based robust matcher are:

- The need for a large corpus of labelled training data.
- The heuristic quality of KCTs: that is, the lack of a formal syntactic theory to guide interpretation at the global, sentence level.
- Some difficulties caused by splitting up the task of interpretation between a hand-coded, chart-based local parser for semantically important phrases and a KCT-based robust matcher at the sentence level.

These disadvantages will now be discussed in more detail.

The question of training data involves an important trade-off. For a given level of performance, one must decide which requires more work: hand-coding rules for a task, or collecting training data and then writing a program that learns the rules from these data. The automatic learning approach is not suitable for all natural language tasks. Like the AT&T group, the CRIM group chose to hand-code the local parsing component of the sentence interpretation task; this was the job of E. Millien's chart parser. The rules for parsing place names, times, prices, and dates are easily written down: most people learn them from the blackboard in primary school. A program designed to learn them from scratch would require a large corpus of training data.

On the other hand, Chapter 3 demonstrates that at the sentence level hand-coding appropriate rules requires considerable human effort. The resulting systems seem to require a great deal of testing on new data and subsequent rewriting of rules in order to reach acceptable performance levels. The natural language ATIS data was collected primarily because of demand by creators of hand-coded linguistic analyzers. Thus, the development of hand-coded systems may require the collection of the same quantity of data as is required by systems that learn from data; the latter require much less effort once the data have been collected. At the sentence level, the trade-off seems to favour automatic learning.

The absence of a formal syntactic theory to guide sentence interpretation in the KCT-based robust matcher was a deliberate decision rather than an oversight. All robust matchers implicitly operate on the assumption that

sentence production and sentence interpretation need not be modelled in the same way. Available syntactic theories try to explain production rather than interpretation, and may therefore be of limited utility for interpretation. Robust matchers carry out interpretation by skipping over large segments of the sentence to focus on semantic "islands" that carry most of the meaning. Usually, the precise form of the patterns that yield the interpretations is hidden in the rules of the robust matcher. The behaviour of the KCT-based robust matcher is easier to characterize formally. It looks for specific regular expressions made up of words and parsed phrases separated by gaps, which were learned from training data. Thus, the KCT-based approach might be seen as the adoption of an unusual syntax that permits gaps, rather than as a rejection of syntax at the sentence level.

The very loose coupling between the local parsing module and the global matcher created some difficulties that might have been resolved if more two-way communication between these two layers had been permitted. For instance, the chart parser often had difficulty interpreting numbers, which can form part of a date, a time, a flight number, a price, or one of the codes found in the ATIS database. Global information about what type of phrase to expect in this position of this type of sentence might have helped resolve some ambiguities of this kind; the chart parser looked at neighbouring words, but did not attempt to globally categorize the sentence.

Similarly, the KCTs in the global matcher had no access to the contents of phrases previously parsed by the local parser. For instance, they might see three phrases marked "CIT" in an incoming sentence, but had no way of knowing which of these three place names, if any, represented the same city. (The meta-rules in the matcher did have access to this information). Furthermore, the local parser had no way of telling the matcher which of its parses were reliable and which were tentative. The division of labour between the local and global components worked well overall; the next section will suggest some improvements.

10.3 Improvements

Since the two contributions of the thesis are KCTs and the KCT-based Robust Matcher, one must distinguish between possible improvements to KCTs (which could be applied outside natural language processing) and improve-

ments to the KCT-based Robust Matcher. For the latter, one must distinguish between specific improvements to CHANEL designed to raise performance on next year's ATIS benchmarks, and more general improvements that affect the overall concept of the KCT-based Robust Matcher.

10.3.1 Improvements to KCTs

- Surprisingly, single-symbol KCTs obtained about the same classification results as the more sophisticated set-membership KCTs. This may be due to the heuristic used to generate set-membership questions, or to the definition of set-membership questions, or both. It would be worthwhile to experiment with a different heuristic for generating word sets, but in my opinion the most promising changes to set-membership KCTs would involve a narrower definition of the permissible questions. The set-membership questions currently allowed are too liberal, allowing the KCT to group together sentences that are dissimilar except for their label. Questions involving more than one word could be restricted to what the Appendix calls U set-membership questions; these allow exactly one word in a set of words to appear in a given location. This might encourage the KCT to learn synonyms and quasi-synonyms; as a side-effect, it would greatly speed up the process of growing set-membership KCTs.

- Another possible improvement would be the addition of questions about gap length. I have already experimented with questions about the total length of a word sequence: these were hardly ever chosen, implying that they yield little useful information. It would be interesting to consider, at each node of the growing single-symbol or set-membership KCT, questions about the length of each gap in the known structure. These would be of the form "is $|+| < N$?" and "is $|+| = N$?" with N varying over reasonable values, say from 1 to 20. The computational cost of considering these questions is trivial. This modification would allow the KCTs to generate patterns like those generated by the IBM approach, with more precise positional information than the patterns generated by the current KCTs.

- Chapter 7 discusses ways of speeding up the KCT-growing process by running it on parallel machines. If methods can be devised for collecting

training data continuously, as discussed in the next section, KCTs could be grown incrementally. That is, instead of regrowing a KCT from scratch each time a new batch of data arrives, we could add and prune only at the nodes of the existing KCT. This would allow us to grow KCTs on arbitrarily large amounts of training data without incurring enormous computational costs.

- Finally, it would be interesting to experiment with applications of KCTs in other domains. They could easily be applied to the natural language tasks mentioned at the beginning of this chapter: information retrieval and language modelling for speech recognition. They could also be applied to other domains, unrelated to natural language, in which the analysis of strings is important. Molecular biology is one such domain: perhaps KCTs have a role to play in looking for patterns in DNA, or in the analysis of the amino acid chains that make up proteins.

10.3.2 Improvements to KCT-Based Robust Matchers

Specific Improvements to CHANEL

- The most important improvement that must be made to CHANEL in time for the 1993 benchmarks will be the easiest to achieve: all KCTs will be retrained, using data from the November 1992 test as well as the earlier training data.

- At the time I was building the KCT-based Robust Matcher component of CHANEL for the Nov92 benchmarks, I believed that the set-membership KCTs would be more accurate sentence classifiers than single-symbol KCTs. Thus, the original version of the system consisted entirely of set-membership KCTs. When the experiments described in the last chapter showed that single-symbol KCTs were preferable, there was enough time left to grow single-symbol KCTs for the displayed attributes and global constraints, but not enough time to rewrite the program that generates KCTs for classifying local constraints. Next year's system will consist entirely of single-symbol KCTs.

- The program for growing KCTs that classify local constraints must be rewritten anyway, to allow these KCTs to depend on the displayed attributes (in the Nov92 version of the system, the output of each KCT is completely independent of all other KCT outputs). Recall that analysis of errors on the Nov92 NL benchmark showed a large number of cases in which displayed attributes and constraints were incompatible: for instance, one might involve the *ground_service* relation and the other the *flight* relation. The algorithms for growing KCTs that classify local phrases will be modified so that these KCTs are permitted to ask questions about the displayed attributes found by other KCTs.

- Another minor change must be made to the algorithm that grows the KCTs that classify cities and airports. Currently, these KCTs have no way of knowing whether a given CIT or AIP is a repetition of a CIT or AIP mentioned earlier - they only see the symbols CIT or AIP, not their contents. Thus, different copies of a repeated city or airport name may be classified differently. In future, repeated CITs and AIPs will be "tied" so that all occurrences receive the same tag.

- The handling of "multiple frame" cases was unsatisfactory. These involve sentences like "show me flights mornings from Denver to Boston and from Denver to New York", for which the natural language "and" should be translated into an SQL "OR" separating two groups of constraints (note the ambiguity about whether "mornings" carries over to the second group). As described in Chapter 8, an *ad hoc* list of multiple frame cases designed to resolve the most frequent multiple-frame ambiguities was devised after study of a large number of ATIS sentences, and training sentences correspondingly labelled. It should be possible to generate a list of possible cases in a more principled way, and thus to design a system with a more elegant structure. In particular, the next version of the KCT-based Robust Matcher should include a method for segmenting utterances that consist of conjoined sentences.

- Another possible improvement involves the local parser. In the current version, error-correcting KCT-based rules operate at the sentence level but not locally; some of the more common local ambiguities could be resolved by using KCTs at this level as well. For instance, the chart parser sometimes had difficulty with numbers, which may be part of

a date, a time, a flight number, a price, or a special code. A KCT could learn rules for resolving ambiguities involving a number from labelled training data; these rules might turn out to depend on words distant from the number itself. Incorporating some KCTs would not mean giving up rule-based local parsing, which works well - the KCTs would act to assist local parsing in borderline cases, thereby increasing robustness.

- Recall that the same system was used for the Nov92 NL and SLS benchmarks. Originally, I had planned to use KCTs trained on labelled recognizer output in the SLS system; however, experiments described in the previous chapter showed that even on SLS data, NL-trained KCTs performed better than SLS-trained ones. Two new approaches to SLS training will be tried:

  1. KCTs will be trained on data made up of all NL data plus all available SLS data.
  2. KCTs will be trained on NL data, then retrained on SLS data. The iterative expansion-pruning algorithm makes this easy to do.

- Careful study of sentences in the Nov92 benchmark which yielded correct answers for NL but not SLS turned up many strange combinations of displayed attributes and constraints in the conceptual representations output for SLS data. For instance, meal codes show up in the company of other attributes and constraints that are completely unrelated to them. This is an encouraging finding: it suggests that semantic meta-rules for spotting wrong answers generated by the SLS module could be grown automatically. This is not a good application for KCTs, since the order of constraints within the sentence seems to be irrelevant. Instead, an ordinary classification tree that asks about the identity of the displayed attributes and constraints in the representation and classifies the representation as "sensible" or "strange" can be grown on NL and SLS data. If the answer is "strange", the system should send "NO ANSWER" or try the next hypothesis on the N-best list.

General Improvements to KCT-Based Robust Matchers

Looking beyond ATIS, one can see several possibilities for research into improved KCT-based Robust Matchers:

- Using dialogue to collect training data;
- Better meta-rules for determining that there is something wrong with a conceptual representation;
- Hierarchies of KCTs;
- KCT-driven lexical search.

Chapter 3 gives arguments for building dialogue capabilities into a spoken language system. A system with such capabilities could train itself, with the assistance of the users. User answers to system questions like "I'm sorry, was it the fares you wanted to see?" would enable the system to label the user's earlier utterances and thus add to its stock of training data. Such a system would be capable of learning the meaning of new combinations of words as its KCTs incrementally grew bigger. Without any intervention by the system designers, the system's performance would gradually improve. This improvement would involve expansion of natural language coverage for concepts that were already in the system's knowledge representation. Completely new concepts would still require intervention by the designers. Nevertheless, the prospect of a spoken language system that improves with use is enticing.

The meta-rules are the rules that assess an interpretation as a whole, determining whether it should be accepted or rejected. They may be applied to a word sequence hypothesis before an interpretation is generated; word sequences judged unlikely to yield a correct interpretation will be rejected. In an N-best system, they can look at interpretations derived from each of the N word sequence hypotheses and pick one of them. Formulating good meta-rules for a robust matcher is a hard problem. In a conventional parser, the meta-rules are a byproduct of the grammar: a word sequence hypothesis is judged unacceptable if it cannot be parsed. By contrast, robust matchers are designed to overlook all kinds of syntactic oddities - it is hard to decide how peculiar a hypothesis has to be before it is rejected. For the specific case of ATIS, the meta-rule classification tree proposed above would depend on semantic criteria; unusual combinations of subunits (displayed attributes and

constraints) in the conceptual representation would lead to rejection. This approach may make sense for other problems as well. In any case, meta-rules are clearly an important topic for further investigation.

Crawford, Fung *et al* employed a hierarchy of classification trees in the information retrieval application: lower-level trees looked for concepts, upper-level trees made decisions about higher-level concepts on the basis of information supplied by lower-level trees. For the ATIS task, this kind of hierarchy proved unnecessary. However, for more complex natural language processing tasks, one can envisage a KCT hierarchy in which each level of KCTs analyses the output of the level below and feeds its decisions to the level above. The idea could prove valuable in the design of dialogue systems, where the lowest level would deal with sentences and each successive level would deal with a higher level of discourse.

Finally, a KCT-based system might carry out search in a completely different way, looking only for keywords that appear in the nodes of some KCT used by the system and using "garbage models" for the gaps. There are several possibilities:

1. Search could be carried out iteratively on each KCT, first determining whether the answer to the question in the KCT's root node is more likely to be "yes" or "no", then recursively descending to the appropriate descendant nodes.

2. Hypotheses at the leaf nodes - each consisting of a regular expression interspersed with gaps - could compete with each other simultaneously.

3. The problem with the two preceding methods is that they would not provide ***a priori*** probability estimates for ending up at a given KCT leaf node; only acoustic probabilities would be taken into account. However, using the patterns learned by KCTs, it would be possible to derive a stochastic context-free grammar (SCFG) whose terminals would be keywords plus the gap symbol. Because of the relatively small number of terminals, the probability parameters in the grammar could be estimated from training data without excessive amounts of computation. Given a language model based on such a grammar and a partial sentence consisting of a prefix, islands, and gaps, there exists a computationally tractable technique for computing the tightest possible upper bound on the probability of the best parse tree generating the

partial sentence [Cor91]. Using this new technique for calculating upper bounds, the rules encoded in the SCFG would guide lexical search.

It seems likely that these approaches would speed up lexical search, since no time would be wasted on non-keywords. Conceivably, they might also improve the proportion of correct interpretations, since search would focus on the problem of deciding between competing interpretations and not split up probabilities between word sequence hypotheses. On the other hand, the information contained in current language models would be lost. The question can be argued convincingly both ways, which is one of the reasons KCT-based lexical search would make a good research topic.

10.4 Conclusion

The thesis describes a method for learning classification rules for sequences from training data, and shows how that method can be applied to semantic problems in speech understanding. It has identified four key issues for future development of trainable robust matchers:

1. The need for quasi-automatic means of acquiring and labelling sample sentences, in order to overcome the data collection bottleneck.

2. The need for a division of labour between different levels of the system and between hand-coding and automatic learning that minimizes human effort while maximizing robustness.

3. The need for good meta-rules, preferably learned from training data, that will increase the probability of obtaining the correct interpretation of an utterance.

4. The possibility of KCT-driven lexical search.

The most important idea in the thesis is the importance of gaps. Classification trees seemed the most obvious way of learning patterns with gaps in them, but they are probably not the only way. In the long run, they will prove to be too heuristic to yield a completely satisfactory solution to the problem. The ideal solution would involve probabilistic grammars that include gaps and can be learnt from training data by efficient methods that provide probabilistic performance guarantees, as described in Chapter 4.

Appendix A

KCT-Growing Details

A.1 Single-Symbol KCTs

As described in the body of the thesis, the set of possible single-symbol questions was generated at each non-zero gap + in the known structure for a set of strings as follows. (Questions about the total length of the string were also generated but turned out to be of little practical importance, since they are hardly ever chosen). Iteratively setting v to be each of the possible symbols (words) in the vocabulary, produce these questions:

1. Is the + equal to v?
2. Is the + equal to $v+$?
3. Is the + equal to $+v$?
4. Is the + equal to $+v+$?

This description was accurate but incomplete: it ignores the problem of symbols that appear more than once in a string. Suppose the question selected for the root node was of the last type - for instance, "is the + equal to $+q+$?" Since at the root node the known structure for the string is $< + >$, the YES child of the root node will contain strings of the form $< + q + >$. Now suppose the question selected to fill this YES child is: "is the first + equal to $+m$?" In other words, the question is: "is the form of the string $< + m\ q + >$?"

If strings are not allowed to contain the same symbol twice, there is no problem. However, this is a rather unrealistic requirement; let us see what happens when it is violated. Suppose we are using this KCT to classify the string $< a\ x\ y\ q\ r\ m\ q\ z >$. At the root node, this string is assigned to the YES child, because it is of the form $< +\ q\ + >$. But the correct answer to the question "is the form of the string $< +\ m\ q\ + >$?" depends on whether we look at the first q or the second q in the string.

The questions actually asked at each non-zero gap + are therefore as follows; each type has been given a mnemonic one-letter name:

1. **J** questions "join" the edges of a gap together; they are of the type "Is the + equal to v?" (v is set to every item in the vocabulary)

2. **L** questions place a symbol v on the "left" side of a gap; they are of the type "Is the + equal to $v+$?"

3. **R** questions place a symbol v on the "right" side of a gap; they are of the type "Is the + equal to $+v$?"

4. **U** questions identify a "unique" symbol v; they are of the type "Is the + of the form $+v+$, where there is precisely one v with no other v to left or right of it within the original +?"

5. **T** questions identify a pair of "twin" adjacent symbols v; they are of the type "Is the + of the form $+vv+$, where there are precisely two adjacent v with no other v on either side of them within the original +?"

6. **N** questions identify two "non-adjacent" symbols v surrounding a gap; they are of the type "Is the + of the form $+v+v+$, where there are two non-adjacent v which may or may not have more v in the gap between them, but such that there is no v to the left of the leftmost of these two v and no v to the right of the rightmost of these two v within the original +?"

7. **M** questions establish that there is one or "more" symbol v in a gap; they are of the type "Is the + of the form $+M(v)+$, where $M(v)$ can be of the form v, vv, or $v + v$, and there is no v on either side of the $M(v)$ within the original +?"

The **M** question type really means, "Is there at least one v within the +?" and is just an OR of the **U**, **T**, and **N** question types. Note that these three question types cover all possible cases of duplication. Hence, strictly speaking, the **M** question type is unnecessary; however, there are probably many occasions when all that matters is whether a given word occurred at least once in a gap, and not how often it occurred. We will need questions to break down an $M(v)$ in the known structure for a set of strings into the three constituent cases. That is, wherever a YES answer to the **M** type of question has put an $M(v)$ (where v is some vocabulary item) in the known structure, we will consider the following questions:

1. **U**: Is $M(v)$ of the form v?
2. **T**: Is $M(v)$ of the form vv?
3. **N**: Is $M(v)$ of the form $v + v$ (where the + may have zero, or any other number of v in it)?

Suppose the root node contains the **M** question "Is the string of the form $< +M(q)+ >$?" The strings in the root node's YES child will then have the known structure $< + M(q) + >$, where the two + do **not** contain a 'q'. Any of the seven kinds of + questions can then be asked about either of the two + positions in the known structure, but the only permissible questions involving the $M(q)$ position are **U**: "is the form of the string $< + q + >$?", **T**: "is the form of the string $< + q\ q + >$?", and **N**: "is the form of the string $< + q + q + >$?"

If the question actually chosen for the YES child of the root is the **N** one, "is the form of the string $< + q + q + >$?", a YES to this question of course implies that the known structure is $< + q + q + >$. Further questions in this case are generated in the usual way from the three +. A NO answer to "is the form of the string $< + q + q + >$?" means that the known structure is $< + M(q) + >$, where $M(q)$ is either q or $q\ q$; the next question may either establish which of these two forms the $M(q)$ possesses, or focus on one of the two +.

A.2 Set Membership KCTs

The idea behind set-membership KCTs was that of automatically grouping together words that are rough semantic equivalents. For instance, words like "taxi", "limousine", "bus" often identify a ground transport query, and probably appear in parallel positions in similar sentences. If a single-symbol KCT asks about these words, it is obliged to split off the sentences containing them into three separate groups and learn further rules about each group separately. A set-membership KCT could include a question like: "Is the sentence of the form $< + v + >$, where v is one of *taxi*, *limousine*, or *bus*?" Thus, the question's YES child would pool three superficially different but actually very similar kinds of sentences, permitting more efficient learning of rules.

The set-membership question types we will consider are called **J**, **L**, **R**, **U**, **T**, **N**, **M**, and **P**. As their mnemonic names indicate, all but the **P** type are analogous to the question types described in the previous section. Each type refers to a set X that is a proper non-empty subset of the vocabulary V. The **P** type deals with the case where we know that certain strings all contain a symbol x belonging to a set X at a given position, and we want to "partition" the strings into those where x belongs to Y (a non-empty proper subset of X) and those where x belongs to $X - Y$. The rest operate on a gap, symbolized "+", within the known structure of the string. Questions of types **U**, **T**, **N**, and **P** can also operate on a string segment containing a segment $M(X)$ identified by a YES answer to an earlier **M** question.

Here is how these question types operate on a gap +, letting x represent any single symbol drawn from the set X and letting X^c represent a non-empty string made up solely of symbols from the complement of X:

- **J** question - "can + be rewritten as x, where x is any symbol in X?"
- **L** question - "can + be rewritten as $x+$, where x is any symbol in X?"
- **R** question - "can + be rewritten as $+x$, where x is any symbol in X?"
- **U** question - "can + be rewritten as $X^c x X^c$, where x is any symbol in X?"
- **T** question - "can + be rewritten as $X^c x y X^c$, where x, y are any symbols in X?"

- **N** question - "can + be rewritten as $X^c x + y X^c$, where x, y are any symbols in X?"

- **M** question - "can + be rewritten as $X^c M(X) X^c$, where $M(X)$ is a substring of the form x, xy, or $x + y$, and x, y are any symbols in X?"

The **P** type is a little different - it operates on a fixed position known to contain a single symbol drawn from the set X, i.e., on a position that can be denoted x:

- **P** question - "can x be rewritten as y, where y is a single symbol drawn from the set Y which is a proper non-empty subset of X?"

Note that the known structure for strings in the YES child of this question contain a y (symbol in Y) at the relevant position, while the strings in the NO child contain a z there, where z is a symbol in $X - Y$.

Operating on a substring of form $X^c M(X) X^c$ identified by a YES answer to an earlier **M** question, type **U**, **T**, or **N** questions separate the three subcases (x, y represent arbitrary symbols in X):

- **U** question - "can $X^c M(X) X^c$ be rewritten as $X^c x X^c$?"

- **T** question - "can $X^c M(X) X^c$ be rewritten as $X^c x y X^c >$?"

- **N** question - "can $X^c M(X) X^c$ be rewritten as $X^c x + x X^c$?"

Finally, type **P** questions can also be applied to substrings of form $X^c M(X) X^c$:

- **P** question applied to $X^c M(X) X^c$ - "can the $M(X)$ be rewritten as $M(Y)$, where Y is a proper non-empty subset of X, and $M(Y)$ is a substring of the form y, yw, or $y + w$, such that y, w are elements of Y?"

Again, note that the YES and NO answers to this question are complementary - the strings in the NO child of the question must have a substring of form $M(Z)$ at the relevant position, where $Z = X - Y$. Both the $M(Y)$ in the YES child and the $M(Z)$ in the NO child of this **P** question can be manipulated to produce new **U**, **T**, **N**, and **P** questions.

The segments of a string written as X^c above may also be written + in the known structure of a string (when the known structure is used to generate

new questions). Even if every X^c is treated as a + in the known structure for the purpose of generating new questions, the algorithm for generating and choosing set-membership questions will never pick a question for this + that involves a set with a symbol from X in it. Whether or not it's worth keeping track of the fact that a particular section of a string is X^c, i.e. contains no symbols drawn from X, is an implementation question more than a theoretical question.

Bibliography

[Ang88] "Learning from Noisy Examples", D. Angluin and P. Laird, *Machine Learning*, 1988, V. 2, no. 4, pp. 343-370.

[Ang87] "Learning Regular Sets from Queries and Counterexamples", *Information and Computation*, D. Angluin, 1987, V. 75, pp. 87-106.

[Ang83] "Inductive Inference: Theory and Methods", D. Angluin and C. Smith, *Computing Surveys*, Sept. 1983, V. 15, no. 3, pp. 237-269.

[Ang82] "Inference of Reversible Languages", D. Angluin, *Journal of the ACM*, July 1982, V. 29, no. 3, pp. 741-765.

[Ang78] "On the Complexity of Minimum Inference of Regular Sets", D. Angluin, *Information and Control*, 1978, V. 39, pp. 337-350.

[App92] "SRI International February 1992 ATIS Benchmark Test Results", D. Appelt and E. Jackson [DAR92].

[Aus91] "BBN HARC and DELPHI Results on the ATIS Benchmarks - February 1991", S. Austin, D. Ayuso, *et al* [DAR91 pp. 112-115].

[Bah91] "Context Dependent Modeling of Phones in Continuous Speech Using Decision Trees", L. Bahl, P. de Souza, *et al* [DAR91 pp. 264-269].

[BahWLa] "A Maximum Likelihood Approach to Continuous Speech Recognition", L. Bahl, F. Jelinek, and R. Mercer [WL90 pp. 308-319].

[BahWLb] "A Tree-Based Statistical Language Model for Natural Language Speech Recognition", L. Bahl, P. Brown, *et al*, [WL90 pp. 507-514].

[Bak79] "Trainable Grammars for Speech Recognition", J. Baker, *Proceedings of the 97th Meeting of the Acoustical Society of America*, 1979.

[Bie72] "On the Synthesis of Finite-State Machines from Samples of their Behavior", A. Biermann and J. Feldman, *IEEE Trans. Comput. C.*, 1972, V. 21, pp. 592-597.

[Bob92] "Syntactic-Semantic Coupling in the BBN DELPHI System", R. Bobrow, R. Ingria, and D. Stallard [DAR92].

[Bre84] "Classification and Regression Trees", L. Breiman, J. Friedman, R. Olshen, and C. Stone, Wadsworth Inc., 1984.

[Bro92] "Analysis, Statistical Transfer, and Synthesis in Machine Translation", P. Brown, S. Della Pietra, *et al*, *Fourth Int. Conf. on Theoretical and Methodological Issues in Machine Translation (TMI 92)*, Montreal, 25-27 Aug. 1992, pp. 83-98.

[Bro88] "A Statistical Approach to Machine Translation", P. Brown, J. Cocke, *et al*, *Int. Conf. on Computational Linguistics (COLING 88)*, Budapest, 22-27 Aug. 1988, pp. 71-76.

[Car92] "High Performance Connected Digit Recognition Using Codebook Exponents", R. Cardin, Y. Normandin, and R. De Mori, *Proc. ICASSP 92*, V. I, pp. 505-508, San Francisco, 1992.

[Chan91] "Synthesis and Recognition of Sequences", S. Chan and A. Wong, *Trans. IEEE PAMI*, Dec. 1991, V. 13, no. 12, pp. 1245-1255.

[Char91] "Bayesian Networks Without Tears", E. Charniak, *AI Magazine*, Winter 1991, V. 12, no. 4, pp. 50-63.

[Cho91] "Optimal Partitioning for Classification and Regression Trees", P. Chou, *Trans. IEEE PAMI*, Apr. 1991, V. 13, no. 4, pp. 340-354.

[Coo76] "Grammatical Inference by Hill-Climbing", C. Cook, A. Rosenfeld, and A. Aronson, 1976, *Inf. Sci.*, V. 10, pp. 59-80.

[Cor91] "Computation of Probabilities for a Stochastic Island-Driven Parser", A. Corazza, R. De Mori, R. Gretter, and G. Satta, *Trans. IEEE PAMI*, 1991, V. 13, no. 9, pp. 936-950.

[Cra91] "Classification Trees for Information Retrieval", S. Crawford, R. Fung, L. Appelbaum, and R. Tong, 1991, *Proc. 8th International Workshop on Machine Learning*, Northwestern University, Evanston, Illinois.

[Cree92] "Trading MIPs and Memory for Knowledge Engineering", R. Creecy, B. Masand, S. Smith, and D. Waltz, Aug. 1992, *Communications of the ACM*, V. 35, no. 8, pp. 48-64.

[Cres73] "The Use of Grammatical Inference for Designing Programming Languages", S. Crespi-Reghizzi, M. Melkanoff, and L. Lichten, *Commun. ACM*, Feb. 1973, V. 16, no. 2, pp. 83-90.

[Cres72] "An Effective Model for Grammatical Inference", S. Crespi-Reghizzi, in *Information Processing 71*, 1972, ed. by B. Gilchrist, pp. 524-529, Elsevier-North Holland.

[DAR92] *Proceedings of the 1992 DARPA Speech and Natural Language Workshop*, to be published by Morgan Kauffmann Inc.

[DAR91] *Proceedings of the 1991 DARPA Speech and Natural Language Workshop*, Feb. 19-22, 1991, Morgan Kauffmann Inc.

[DAR90] *Proceedings of the 1990 DARPA Speech and Natural Language Workshop*, June 1990, Morgan Kauffmann Inc.

[Dym90] "A Symmetrical Approach to Parsing and Generation", M. Dymetman, P. Isabelle, and F. Perrault, 1990, *COLING 90*, pp. 90-96.

[Dud73] "Pattern Classification and Scene Analysis", R. Duda and P. Hart, 1973, Wiley Co.

[ErmWL] "The HEARSAY-II Speech Understanding System", L. Erman and V. Lesser [WL90 pp. 235-245].

[Fu86a] "Grammatical Inference: Introduction and Survey - Part I", K. S. Fu and T. Booth, *Trans. IEEE PAMI*, May 1986, V. 8, no. 3, pp. 343-359.

[Fu86b] "Grammatical Inference: Introduction and Survey - Part II", K. S. Fu and T. Booth, *Trans. IEEE PAMI*, May 1986, V. 8, no. 3, pp. 360-375.

[Fu86c] "A Step Towards Unification of Syntactic and Statistical Pattern Recognition", K.S. Fu, *Trans. IEEE PAMI*, May 1986, V. 8, no. 3, pp. 398-404.

[Fu82] "Syntactic Pattern Recognition and Applications", K. S. Fu, Prentice-Hall Inc., 1982.

[Gai78] "Maryanski's Grammatical Inferencer", B. Gaines, *IEEE Trans. Comput.*, 1978, C-28, pp. 62-64.

[Garc90] "Inference of k-Testable Languages in the Strict Sense and Application to Syntactic Pattern Recognition", P. Garcia and Enrique Vidal, *Trans. IEEE PAMI*, Sept. 1990, V. 12, no. 9, pp. 920-925.

[Garc87] "Local Languages, the Successor Method, and a Step Towards a General Methodology for the Inference of Regular Grammars", P. Garcia, E. Vidal, and F. Casacuberto, *Trans. IEEE PAMI*, Nov. 1987, V. 9, no. 6, pp. 841-845.

[Gare79] "Computers and Intractability: A Guide to the Theory of NP-Completeness", M. Garey and D. Johnson, W. H. Freeman Co., 1979.

[Gars87] "The Computational Analysis of English", R. Garside, G. Leech, and G. Sampson, Longman Group Ltd, 1987.

[Gel91] "An Iterative Growing and Pruning Algorithm for Classification Tree Design", S. Gelfand, C. Ravishankar, and E. Delp, *Trans. IEEE PAMI*, Feb. 1991, V. 13, no. 2, pp. 163-174.

[Gol78] "Complexity of Automaton Identification from Given Data", E. M. Gold, *Information and Control*, 1978, V. 37, pp. 302-320.

[GraWL] "Vector Quantization", R. Gray [WL90 pp. 75-100].

[Hau88] "Proceedings of the 1988 Workshop on Computational Learning Theory", ed. by D. Haussler and L. Pitt, 1989, Morgan Kaufmann Inc.

[Hil85] "The Connection Machine", W. Daniel Hillis, MIT Press, 1985.

[Hir92] "Multi-Site Data Collection for a Spoken Language Corpus", L. Hirschman [DAR92].

[Hon91] "Recent Progress in Robust Vocabulary-Independent Speech Recognition", H.-W. Hon and K.-F. Lee [DAR91 pp. 258-263].

[Hop79] "Introduction to Automata Theory, Languages, and Computation", J. Hopcroft and J. Ullman, Addison-Wesley Co., 1979.

[Jack91] "A Template Matcher for Robust NL Interpretation", E. Jackson, D. Appelt, *et al* [DAR91 pp. 190-194].

[Jaco90] "SCISOR: Extracting Information from Online News", P. Jacobs and L. Rau, *Commun. ACM*, Nov. 1990, pp. 88-97.

[Jai91] "PARSEC: A Connectionist Learning Architecture for Parsing Spoken Language", A. Jain, *Ph.D. Thesis*, Dec. 1991, Carnegie Mellon University, CMU-CS-91-208.

[Jel92] "Principles of Lexical Language Modeling for Speech Recognition", F. Jelinek, R. Mercer, and S. Roukos, in "Advances in Speech Signal Processing", ed. by S. Furui and M. Sondhi, 1992, (pp. 651-699), Marcel Dekker Inc.

[Kea90a] "The Computational Complexity of Machine Learning", M. Kearns, MIT Press, 1990.

[Kea90b] "Efficient Distribution-free Learning of Probabilistic Concepts"(extended abstract), M. Kearns and R. Schapire, in *31st Annual Symposium on Foundations of Computer Science*, Oct. 1990, V. 1, pp. 382-391.

[Kia91] "Recursive Optimal Pruning of Tree-Structured Vector Quantizers", S.-Z. Kiang, G. Sullivan, *et al*, *Proc. ICASSP 91*, M1.5, pp. 2285-2288, 1991.

[KlaWL] "Review of the ARPA Speech Understanding Project", D. Klatt [WL90 pp. 554-575].

[Kno76] "A Method for Inferring Context-free Grammars", B. Knobe and K. Knobe, *Information and Control*, 1976, V. 31, pp. 129-146.

[Kub92] "BBN BYBLOS and HARC February 1992 ATIS Benchmark Results", F. Kubala, C. Barry, *et al* [DAR92].

[Kud88] "Efficient Regular Grammatical Inference Techniques", M. Kudo and M. Shimbo, *Pattern Recognition*, 1988, V. 21, no. 4, pp. 401-409.

[Kuh90] "A Cache-Based Natural Language Model for Speech Recognition", R. Kuhn and R. De Mori, *Trans. IEEE PAMI*, June 1990, V. 12, no. 6, pp. 570-583.

[Kuh92] "Corrections", R. Kuhn and R. De Mori, *Trans. IEEE PAMI*, June 1992, V. 14, no. 6, pp. 691-692.

[Lar91] "Applications of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm", K. Lari and S.J. Young, *Computer Speech and Language*, V. 5, pp. 237-257, 1991.

[Lar90] "The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm", K. Lari and S.J. Young, *Computer Speech and Language*, V. 4, pp. 35-56, 1990.

[LeeWL] "Context-Dependent Phonetic Hidden Markov Models for Speaker-Independent Continuous Speech Recognition", K.-F. Lee [WL90 pp. 347-365].

[Li88] "On the Learnability of Finite Automata" (extended abstract), M. Li and U. Vazirani [Hau88 pp. 359-370].

[LowWL] "The HARPY Speech Understanding System", B. Lowerre and R. Reddy [WL90 pp. 576-586].

[Mak85] "Vector Quantization in Speech Coding", J. Makhoul, S. Roucos, and H. Gish, *Proc. IEEE*, V. 73, pp. 1551-1588, 1985.

[Mar77] "Inference of Finite-State Probabilistic Grammars", F. Maryanski and T. Booth, *IEEE Trans. Comput.*, 1977, C-26, pp. 521-536.

[Mel88] "Lexical Transfer: Between a Source Rock and a Hard Place", A. Melby, *Int. Conf. on Computational Linguistics (COLING 88)*, Budapest, 22-27 Aug. 1988, pp. 411-413.

[Moo90] "SRI's Experience With the ATIS Evaluation", R. Moore, D. Appelt, *et al* [DAR90 pp. 147-148].

[Nad91] "An Iterative Flip-Flop Approximation of the Most Informative Split in the Construction of Decision Trees", A. Nadas, D. Nahamoo, *et al*, *Proc. ICASSP 91*, S8.10, pp. 565-568, 1991.

[Norm92] "Computer Speech Understanding Scientific Report", Y. Normandin (Technical Project Leader) *et al*, Centre de Recherche Informatique de Montréal, July 1992.

[Norm91] "Hidden Markov Models, Maximum Mutual Information Estimation, and the Speech Recognition Problem", Y. Normandin, Ph.D. Thesis, McGill University Dept. of Electrical Engineering, March 1991.

[Nort92] "Recent Improvements and Benchmark Results for the Paramax ATIS System", L. Norton, D. Dahl, and M. Linebarger [DAR92].

[Nort91] "Augmented Role Filling Capabilities for Semantic Interpretation of Spoken Language", L. Norton, M. Linebarger, *et al* [DAR91 pp. 125-133].

[Oeh91] "Unbalanced Tree-Growing Algorithms for Practical Image Compression", K. Oehler, E. Riskin, and R. Gray, in *Proc. ICASSP 91*, M1.7, pp. 2293-2296, 1991.

[O'S87] "Speech Communication", D. O'Shaughnessy, Addison-Wesley, 1987.

[Ost91] "Integration of Diverse Recognition Methodologies Through Reevaluation of N-Best Sentence Hypotheses", M. Ostendorf, A. Kannan, *et al* [DAR91 pp. 83-87].

[Pal92] "DARPA February 1992 ATIS Benchmark Test Results", D. Pallett, N. Dahlgren, *et al* [DAR92].

[Par86] "Conversational Language Comprehension Using Integrated Pattern-Matching and Parsing", R. Parkison, K. Colby and W. Faught, in "Readings in Natural Language Processing", ed. by B. Grosz *et al*, pp. 551-562, 1986, Morgan Kaufmann Inc.

[Pea84] "Heuristics", J. Pearl, Addison-Wesley, 1984.

[Per80] "Definite Clause Grammars for Language Analysis", F. Pereira and D. Warren, *Artificial Intelligence*, 1980, V. 13, pp. 231-278.

[Pie92a] "A Speech Understanding System Based on Statistical Representation of Semantics", R. Pieraccini, E. Tzoukermann, *et al*, *Proc. ICASSP 92*, pp. I-193 - I-196, Mar. 1992.

[Pie92b] "Progress Report on the Chronus System: ATIS Benchmark Results", R. Pieraccini, E. Tzoukermann, *et al* [DAR92].

[Pie91] "Stochastic Representation of Conceptual Structure in the ATIS Task", R. Pieraccini, E. Levin, and C.-H. Lee [DAR91 pp. 121-124].

[Pit90] "Prediction-Preserving Reducibility", L. Pitt and M. Warmuth, *Journal of Computer and System Sciences*, Dec. 1990, V. 41, no. 3, pp. 430-467.

[Pit89] "The Minimum Consistent DFA Problem Cannot Be Approximated Within Any Polynomial", L. Pitt and M. Warmuth, in *Proceedings of the 21st ACM Symposium on the Theory of Computing*, 1989, pp. 421-432.

[Pit88a] "Computational Limitations on Learning from Examples", L. Pitt and L. Valiant, *Journal of the ACM*, 1988, V. 35, no. 4, pp. 965-984.

[Pit88b] "Reductions Among Prediction Problems", L. Pitt and M. Warmuth, in *Proceedings of the 3rd IEEE Conference on Structure in Complexity Theory*, 1988, pp. 60-69.

[Pol92] "Experiments in Evaluating Interactive Spoken Language Systems", J. Polifroni, L. Hirschman, *et al* [DAR92].

[Pop69] "Conjectures and Refutations", K. Popper, pg. 37, 1969 edition, Routledge and Kegan Paul Ltd.

[Pri92] "Subject-Based Evaluation Measures for Interactive Spoken Language Systems", P. Price, L.Hirschman, *et al* [DAR92].

[Pri90] "Evaluation of Spoken Language Systems: the ATIS Domain", P. Price [DAR90 pp. 91-95].

[Pri88] "The DARPA 1000-Word Resource Management Database for Continuous Speech Recognition", P. Price, W. Fisher, *et al*, *Proc. ICASSP 88*, 1988.

[PrV91] "Automatic Learning of Structural Language Models", N. Prieto and E. Vidal, *Proc. ICASSP 91*, pp. 789-792.

[RabWL] "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", L. Rabiner [WL90 pp. 267-296].

[Ric84] "Efficient Regular Grammatical Inference for Pattern Recognition", M. Richetin and F. Vernadat, 1984, V. 17, no. 2, pp. 245-250.

[Ris91] "Lookahead in Growing Tree-Structured Vector Quantizers", E. Riskin and R. Gray, *Proc. ICASSP 91*, M1.6, pp. 2289-2292, 1991.

[Scha91] "The Design and Analysis of Efficient Learning Algorithms", R. Schapire, *Ph.D. Thesis*, Massachusetts Institute of Technology, Feb. 1991.

[Schw92] "New Uses for the N-Best Sentence Hypotheses Within the Byblos Speech Recognition System", R. Schwartz, S. Austin, *et al*, *Proc. ICASSP 92*, pp. I-1 - I-4, Mar. 1992.

[Sen92] "A Relaxation Method for Understanding Spontaneous Speech Utterances", S. Seneff [DAR92].

[Sen91] "Interactive Problem Solving and Dialogue in the ATIS Domain", S. Seneff, L. Hirschman, and V. Zue [DAR91 pp. 354-359].

[SenWL] "A Joint Synchrony/Mean-Rate Model of Auditory Speech Processing", S. Seneff [WL90 pp. 101-111].

[Shi82] "Polynomial Time Inference of Extended Regular Pattern Languages", T. Shinohara, *Proc. of RIMS Symposia on Software Science and Engineering*, ed. by G. Goos and J. Hartmanis, Kyoto 1982, pp. 115-127.

[Soo90] "A Tree-Trellis Based Fast Search for Finding the N-Best Sentence Hypotheses in Continuous Speech Recognition", F. Soong and E.-F. Huang [DAR90 pp. 12-19].

[Sta92] "Fragment Processing in the DELPHI System", D. Stallard and R. Bobrow [DAR92].

[Sun91] "Third Message Understanding Evaluation and Conference (MUC-3)", B. Sundheim [DAR91 pp. 301-305].

[Swi] "Gulliver's Travels", Jonathan Swift, *The Writings of Jonathan Swift*, ed. by R. Greenberg and W. Piper, pg. 156, 1973 edition, W. W. Norton Co.

[Tho86] "Dynamic Programming Inference of Markov Networks from Finite Sets of Sample Strings", M. Thomason and E. Granum, *IEEE Trans. PAMI*, July 1986, V. 8, no. 4, pp. 491-501.

[Val84] "A Theory of the Learnable", L. Valiant, *Communications of the ACM*, Nov. 1984, V. 27, no. 11, pp. 1134-1142.

[Van78] "On the Inference of Stochastic Regular Grammars", A. Van der Mude and A. Walker, *Information and Control*, 1978, V. 38, pp. 310-329.

[WL90] "Readings in Speech Recognition", ed. by A. Waibel and K.-F. Lee, Morgan Kaufmann Inc., 1990.

[WaiWL] "Introduction", A. Waibel and K.-F. Lee [WL90 pp. 1-5].

[War92] "Speech Understanding in Open Tasks", W. Ward, S. Issar, *et al* [DAR92].

[War91] "Evaluation of the CMU ATIS System", W. Ward [DAR91 pp. 101-105].

[War90] "The CMU Air Travel Information Service: Understanding Spontaneous Speech", W. Ward [DAR90 pp. 127-129].

[Wei92] "A New Approach to Text Understanding", R. Weischedel, D. Ayuso, *et al* [DAR92].

[Win86] "Understanding Computers and Cognition", T. Winograd and F. Flores, Addison-Wesley Co., 1986.

[You91] "Using Semantics to Correct Parser Output for ATIS Utterances", S. Young [DAR91 pp. 106-111].

[Zue92] "The MIT System ATIS System: February 1992 Progress Report", V. Zue, J. Glass, *et al* [DAR92].

[Zue91] "Integration of Speech Recognition and Natural Language Processing in the MIT VOYAGER System", V. Zue, J. Glass, *et al*, *Proc. ICASSP 91*, Toronto, Ontario, May 1991.

[ZueWL] "The Use of Speech Knowledge in Automatic Speech Recognition", V. Zue [WL90 pp. 200-213].