Rheinisch Westfälische Technische Hochschule Aachen
Lehrstuhl für Informatik VI
Prof. Dr.-Ing. Hermann Ney

Speech Recognition and Language Processing in SS 2003

Natural Language Database Queries using Classification and Regression Trees

Samuel S. Okae

Matriculation number 23 66 72

July 31, 2003

Supervisors: Klaus Macherey
Oliver Bender

Abstract

The key characteristic of any understanding system is that it represents the meaning of its inputs. For instance, natural language understanding (NLU) aims at extracting all the information from a natural language based input which are relevant for a specific application. For this purpose, concepts are used to represent the associated meaning. Hence, the task of NLU is to convert a sequence of words into a sequence of concepts [6]. A typical application using NLU are automatic dialogue systems, which map a user's utterance to a machine-readable form, so that it can be translated into a database query in a further step. There are several approaches to NLU, and in this paper, I present a method applying decision trees.

Table of Contents

1 Introduction

In this chapter, I discuss the aim of natural language understanding (NLU) and present the most relevant approaches to this task. In section 1.1, the definition and basic use of NLU is presented. In section 1.2, I look at some applications of NLU. In section 1.3, the various approaches to NLU are motivated; I introduce the decision tree approach, which is the concentration of this seminar.

1.1 Definition and Use of Natural Language Understanding

One very important aspect of natural language processing (NLP) are automatic dialogue systems, which require a NLU component. The aim of NLU is the mapping of a user's utterance to a machine readable form for further processing. NLU is therefore defined as *the understanding of human language by a computer* [4]. NLU is a sub-field of artificial intelligence research that aims at making computers understand statements written or spoken in human language. The objective is to map a natural language input into a semantic representation. A specific example to this seminar is the mapping of a sequence of words into a sequence of concepts, in which case a concept is defined as the smallest unit of meaning relevant for a specific task [6]. Fig. 1 is an illustration of this notion.

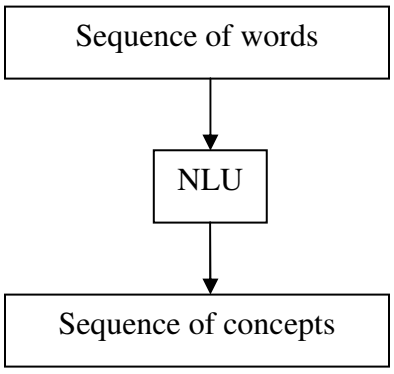

Fig. 1: Natural Language Understanding Task

1.2 Applications of Natural Language Understanding

NLU is needed in many fields like natural language processing, linguistic research and artificial intelligence. Its application can be divided into two major groups, dialogue-based and text-based applications. The following sub-sections deal with details of these applications. However, there could be situations in which the two must be combined.

1.2.1 Text-based Applications

Text-based NLU applications process written text, such as books, newspapers, reports, manuals, e-mail messages, and so on. Some of the current uses of text-based natural language research are:

- Summarizing Texts: where NLU is used to analyze the document to find the salient concepts, facts, and sometimes whole sentences [7], for instance, summarizing a 1000-page report to 2 pages.
- Machine Translation: where NLU is used to analyze the source document in order to be able to translate it from one language into another, a good example is the production of product manuals in different languages.
- Information Extraction: where NLU is used for extracting key facts out of short documents [7], for instance, building a database of all purchases made by a company in a given period.

It must be noted that not all systems that perform such tasks use NLU techniques.

1.2.2 Dialogue-based Applications

Dialogue-based applications on the other hand involve communication between humans and machines. Mostly, this involves speech input, but it may also include written language, for example through the use of keyboards. Spoken dialogue systems are well-known applications of this type, where NLU is used to interpret the input. The aim here is to provide an interface between a user and a computer-based application. For instance in air travel information systems or telephone directory assistance.

1.3 Approaches to Natural Language Understanding

For both text-based applications and dialogue-based applications, there are several approaches that have been successfully applied. Some of them make use of:

- **Rule-based approaches**
- **Hidden Markov Models (HMM)**
- **Machine Translation approach**
- **Decision Trees**

In the subsequent sections, I look briefly at their definition and the basic ways in which they are applied.

1.3.1 Rule-Based Approaches

In the design of a NLU system, a NLU component will be faced with the issue of ambiguity of sentences, for instance “sorry” in English could mean “leid” or “traurig” in German. If there is an input sentence containing such words, a NLU system has to analyze all possibilities of this sentence in order to be able to arrive at the right output and this could be problematic. Rule-based approaches help to solve this problem since they have set down rules for the grammar. We will discuss two types of grammars, context free grammars (CFG) and Probabilistic context free grammars (PCFG).

A CFG is a set of recursive rewriting rules (or productions) used to generate patterns of strings. A production is a rule in a grammar that determines how a string of symbols may be rewritten as another. A CFG is given by $G = (V, \sum, S, P)$,

where:

- V is a finite set of variables or non-terminal symbols.
- $\sum$ is a finite set of symbols or terminal symbols.
- S is a *start symbol*, which is a specific non-terminal symbol ($S \epsilon V$) that appears in the initial string.

- P is a set of *productions*, which are rules for replacing non-terminal symbols (the left side of the production) in a string with other non-terminal or terminal symbols (the right side of the production).

Given an input string, a string of terminal symbols is generated by a CFG; the process is started by applying one of the productions or rules to the given string. If no production states that a symbol currently in the string can be replaced, the process is stop, since there is no possibility to replace any string. If it is possible to replace any of the strings, the string is rewritten by replacing one of its symbols with a sequence that matches some production. This process is repeated until all non-terminals have been replaced by terminal symbols. There are several ways to generate the set of words produced by a grammar; I present a technique based on the number of productions:

Given the CFG with $G = (V, \sum, S, P)$:

$V = \{S, a, b, A, B\}$

$S = \{S\}$

$\sum = \{a, b\}$

$P = \{S \rightarrow aB, S \rightarrow bA, A \rightarrow a, A \rightarrow aS, A \rightarrow bAA, B \rightarrow b, B \rightarrow bS, B \rightarrow aBB\}$

To find the words generated by this CFG, we follow the steps below:

1. Applying at most zero productions, will not generate any word.
2. Applying at most one production and starting with the start symbol (S) we can generate {aB}. And since it does not consist entirely of terminal symbols, no words can be generated here also.
3. Applying at most two productions, we can generate all the strings that are possible with one production, plus any additional strings we can generate with an additional production. {ab, abS, aaBB}. Only one of these strings is made up of only terminal symbols, hence, the word we can generate using at most two productions is {ab}.

4. Applying at most three productions, we can generate {ab, abaB, abbA, aabb, aabSbS, aaaBBaBB}. The set of words we can generate with at most three productions is therefore {ab, aabb}.
5. Applying at most four productions, we can generate {ab, abab, ababS, abaaBB, abba, abbaS, abbbAA, aabb, aabaBbaB, aabbAbbA, aaabbabb, aaabSbSabSbS, aaaaBBaBBaaBBaBB}. The set of words we can generate with at most four productions is therefore {ab, abab, abba, aabb, aaabbabb}.

We can repeat this process as many as N times, to find all words the grammar can generate by applying N productions.

Probabilistic context free grammars on the other hand are probabilistic extensions of CFGs. The goal is to define a probability for each symbol string by assigning a probability to each rule of a given grammar [6]. Just like CFG Productions, each PCFG Production specifies a non-terminal symbol that can be expanded to a sequence of terminals and non-terminal symbols.

1.3.2 HMM-based Approach

A Hidden Markov Model is a Markov chain with a finite set of states, where each state generates an observation and each observation is associated with a probability distribution [9]. Transitions from one state to another are governed by a set of probabilities called *transition probabilities.* For each state of the model, an outcome or observation is generated according to the associated emission probability. There are five main ingredients of a HMM [9]:

1. the number of states N, individual states are given by S: = $\{S_{1,}...,S_N\}$, the state at time t is given by q_t
2. the number of distinct observation symbols per state M, we denote the individual symbols as V = $\{V_1, V_{2,} \ldots, V_M\}$
3. the state transition probability distribution A

$A = a_{ij} = p[q_{t+1} = S_j \mid q_t = S_i], \quad 1 \le i \le N, \quad 1 \le j \le N$

in other words, a_{ij} is the probability that the next state is q_j given that the current state is q_i.

4. the observation symbol probability distribution B in state j

   $B = \{b_j(k) = p[V_k \text{ at } t \mid q_t = S_j], \quad 1 \le j \le N, \quad 1 \le k \le M$

5. the initial state distribution π

   $\pi_i = p[q_1 = S_i], \quad 1 \le i \le N$

Given appropriate values of N, M, A, B and π the HMM can be used as a generator for observation sequences. For additional details on HMM, refer to [9].

1.3.3 Machine Translation Approach

In simple terms, machine translation (MT) is the automatic translation from one human language to another. In order to understand the fundamental principles of MT, I will explain some basic principles of translation.

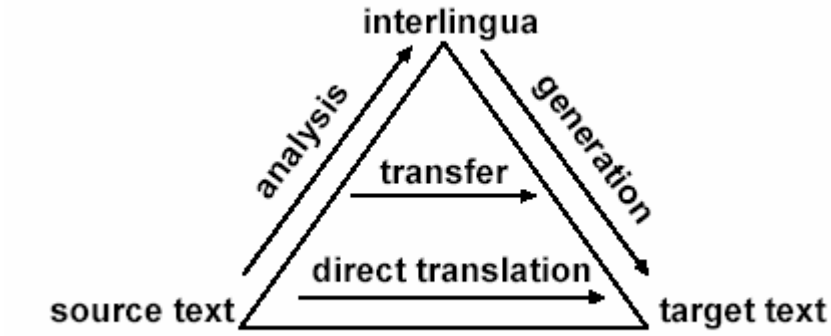

Fig. 2: Translation model

Fig. 2 above is a representation of the three translation strategies, which are direct, interlingua and transfer. The direct strategy is the most basic; the source text is translated word-by-word directly. The Interlingua strategy uses a language independent representation between the analysis of the source text and the generation of the target text. The final translation strategy is the transfer strategy. This strategy has three phases; the source text goes through lexical, syntactic and semantic phases before the final target language text is generated.

Now to a more specific definition of MT. MT is the process by which a natural source sentence $f_1^J := f_1 \ldots f_j \ldots f_J$ is translated into a formal target language sentence $e_1^I = e_1 \ldots e_i \ldots e_I$ among all possible strings, that maximizes $pr(e_1^I | f_1^J)$ [10]. the result therefore will be:

$$\underset{e_1^I}{\operatorname{argmax}} Pr(e_1^I | f_1^J) = \underset{e_1^I}{\operatorname{argmax}} \{ Pr(e_1^I) \cdot Pr(f_1^J | e_1^I) \}$$

Two independent probability distributions arise, the first is $Pr(f_1^J | e_1^I)$, which is the translation model linked to the input string and the second is $Pr(e_1^I)$, which is the language model and this indicates how well the output string is formed. For additional details on machine translation, refer to [6].

1.3.4 Decision Tree Approach

A decision tree works like a flow chart and looks like an upside down tree. It is constructed from training data made up of a set of attributes that are to be tested in order to predict an output. One major advantage decision trees offer for NLU is the ability to use training data to train the tree during the construction process and this makes future prediction of word sequences from the tree less error prone. This is because decision trees have set rules for making decisions at each node as you go down a particular decision path. The next chapter is dedicated to decision trees and how they are constructed.

2 Basics of Decision Trees

In chapter 1, we discussed natural language understanding and the various approaches that can be used. In this chapter, I will take a detailed look at decision trees as a build up to chapter 3 where I will discuss how the decision tree approach can be applied to the task of NLU.

2.1 Definition and Use of Decision Trees

In most cases of NLU applications, it becomes necessary to make decisions based on a sequence of questions. The nature of the answer to a particular question then leads to the next question, and this continues until the final decision is reached. A decision tree is an important structure that can be trained to handle such situations.

A decision tree is defined as a classifier of a set of attributes in the form of a tree structure where each node is either a *leaf node* or a *decision node*. Leaf nodes indicate the value of the target attribute, for example, word sequence in natural language processing. The decision node on the other hand specifies some test to be carried out on attribute-values [8]. The tree is made up of a set of nodes starting at the root node and ending at a leaf node. A node is called a decision node if it splits into other nodes. There could be zero or more internal nodes, and one or more leaf nodes. All internal nodes have one or more child nodes and they contain splits at each decision node. The root node is displayed at the top of the tree and connected by branches to other nodes. The branches of the tree represent attribute-values with one branch and sub-tree for each possible outcome of the test at the node. Fig. 3 is a typical example of a real life situation which can be used to determine whether someone is happy of not. Depending on the decision taken at the root, one could be a male or female. If the female branch is followed, a female who is un employed but married will be happy.

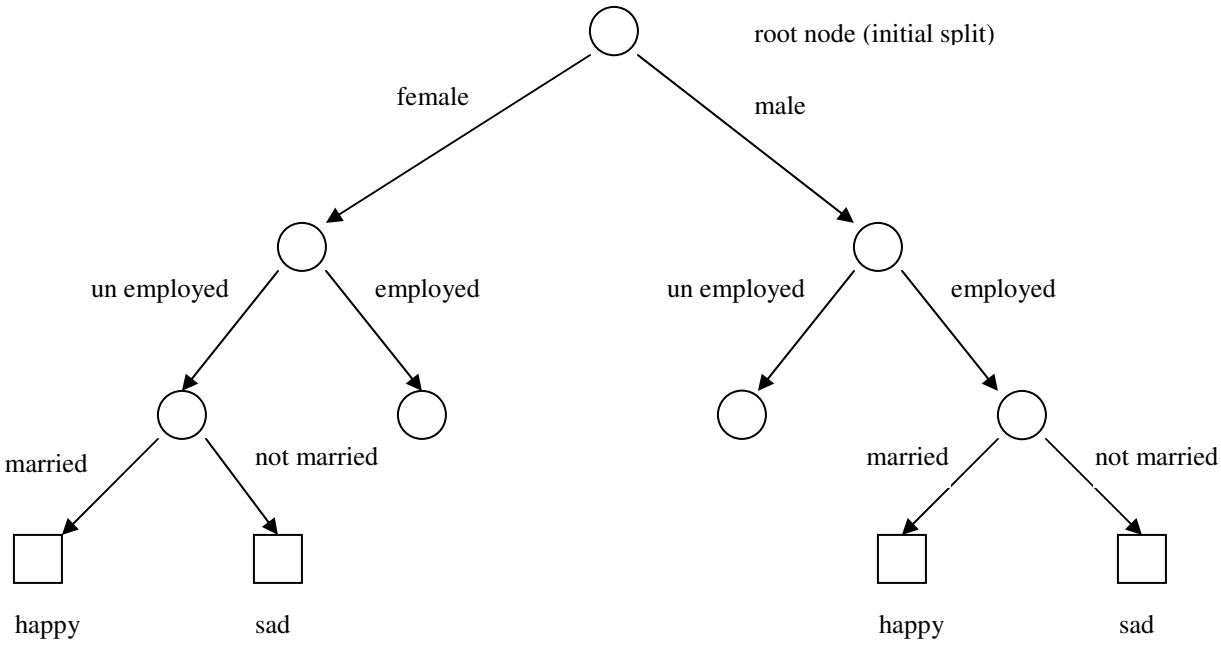

Fig. 3: Decision Tree Example

A decision tree can be used largely for prediction purposes by starting at the root node and moving down the tree until a leaf node is reached. Some of the major uses include medical diagnosis, risk analysis and calendar scheduling.

2.2 Building of Decision Trees

In section 2.1, I mentioned some of the uses of decision trees, in this section I discuss how they can be constructed in order to make its use for such purposes feasible.

A decision tree is built from a training set, which consists of objects. In NLU, the objects are word sequences and each object is completely described by a set of attribute-values. The building of the tree starts at the root node where the value of a particular attribute is requested. There are links from the root node that correspond to different possible values and based on the answer reached at the node, a link is followed to the appropriate child. The process is then repeated at each child node until a leaf node is reached where there are no further questions to be asked and this node is referred to as a leaf node.

2.2.1 Decision Tree Construction Algorithm

Algorithm 1 can be followed to build a decision tree:

Algorithm 1:

1. *create root node*
2. *select the best attribute for the current node*
3. *generate child nodes, one for each possible value of the selected attribute*
4. *test property at the current node and create branches:*
    - *if no question declare as leaf node*
    - *false decisions on the left branch*
    - *true decisions on the right branch*
5. *if each child node is a leaf node, stop*
6. *else set child to current node and go to 2*

When applying the decision tree construction algorithm, we have to bare in mind six major questions [3]:

1. Should the properties be restricted to a single value or allowed to be multi valued? In other words, how many splits will there be at a node?
2. Which property should be tested at a node?
3. When should a node be declared as a leaf node?
4. If a tree becomes too large how can it be made smaller or how can it be pruned?
5. If a leaf node is impure, how should the category label be assigned?
6. How should missing data be handled?

Answers to the above questions are presented in the subsequent sections.

2.2.2 Number of Splits

In general, the number of splits to be generated at each node depends on the property being tested at the node. For instance, some properties can only yield a ‘yes’ or ‘no’ answer whilst others have the possibility of more than two splits for example in circuit systems where we could yield an ‘OR’ , ‘AND’ or a ‘XOR’. The number of splits at a node however depends largely on the designer and this may vary throughout the tree.

2.2.3 Property to be Tested and Node Impurity

As mentioned above, the look of the tree or the final decision to be made depends on the property that will be tested at each node. It might become necessary to test two or more properties at a node, but in deciding on this property we bare in mind the fact that the simpler the tree the better. The aim is to make the tree simple and compact with few nodes, and this is well explained by *Occam's razor, which states that the simplest model that explains data is the one to be preferred* [3]. In order to make the tree as simple as possible we need to make sure that descendant nodes are as pure as possible. In view of this, we introduce a property I(T) which is the Impurity of a node T. There are various types of impurity; variance impurity, entropy impurity, misclassification impurity. Refer to [3] for details. I look at the Gini impurity, which is given by:

$$i(T) = -\sum_{I \neq j} (p(w_i)\, p(w_j))$$

$p(w_j)$ is the fraction of patterns at node T that are in category w_j. the smaller the impurity at a node the better. In fact, our desire is to have a zero impurity at each. Since this is not possible most of the time, the path with the lowest impurity is selected. Hence, the need arises to calculate the drop in impurity, which is defined by [3]:

$$\Delta\, i(T) := i(T) - p_L\, i(T_L) - (1-p_L)\, i(T_R)$$

Where T_L and T_R are the impurities of the left and right descendant nodes respectively, and p_L is the probability of generating a particular node.

2.2.4 When To Stop Splitting

The idea of creating splits at a node could continue as long as the impurity of the child is lower than that of the parent and hence there is the tendency of going too far down the tree before stopping. There is also the possibility of stopping too early if child nodes with a lower impurity are not generated. So it becomes very necessary to determine when to stop splitting and there are two methods that can be used.

The first is the validation and cross validation method, where the tree is trained with a subset of the data and the remaining is kept as a validation set. For instance, one could decide to use about 90% data for validation and the remaining 10% as the validation set. In this case, splitting is continued until the error on the validation set is minimized.

In the second method, we set a small threshold during the reduction of the impurity and stop splitting as soon as the impurity at a node is reduced below this threshold. The advantage of this method over the first is that the tree is trained using the entire set of data.

2.3 Pruning of Decision Trees

Splitting may be stopped prematurely due to the designer's inability to look ahead. Such a situation is referred to as the 'horizon effect', which is a situation whereby you feel you have reached your destination whilst in fact you are not there. In a situation where splitting is stopped prematurely, a node could be declared as a leaf node thereby cutting off the possibility of having beneficial child nodes and this could be misleading since one may end up having a biased tree which may not reflect the result that the training data should have yielded.

Pruning is the alternative approach used to check such behavior in decision tree growing. Pruning is considered as the inverse of splitting since in pruning a tree is grown fully until leaf nodes have minimum impurity. After the whole tree has been generated to the highest level, all pairs of neighboring leaf nodes are considered for elimination. The idea is to find pairs of nodes whose elimination yields a little increase in impurity for elimination and declare the common parent as a leaf node. It must be noted that it is not mandatory to start pruning from the leaf nodes only. Another way of achieving pruning is to set rules for all leaf nodes and based on these rules, pruning is achieved. This method is called rule pruning.

2.4 Strengths and Weaknesses of Decision Trees

I have been looking at how to construct decision trees. I now mention some of their strengths and weaknesses.

2.4.1 The Strengths

The strengths of decision trees vary depending on where they are being applied. We look at a few:

- Decision trees are able to generate understandable rules. They structure the decision processes hence problems can be approached in a sequential manner.
- Decision trees are powerful and popular tools for classification and prediction.
- Decision trees are fast and easy to interpret, even for unseen data.
- Decision trees provide a clear indication of which fields are most important for prediction.

2.4.2 The Weaknesses

There can never be a perfect situation, every thing has some disadvantages, and decision trees are no exception. Among some of the weaknesses are:

- Decision trees are not good for estimation tasks where the goal is to predict the value of a continuous attribute.
- Though Decision trees are simple, they can be very restrictive, since they restrict you to which property to test.
- Decision trees are prone to errors in problems with relatively small number of training examples.
- Most decision tree algorithms only examine single information at a time and this is a great limitation.
- They might become too large if a proper algorithm is not developed to determine leaf nodes.

3 Natural Language Understanding Using Decision Trees

This chapter deals with semantic classification trees and how they can be applied to natural language understanding. In section 3.1, a few examples of decision tree applications are mentioned. In section 3.2, I discuss the two main types of semantic classification trees. Section 3.3 is a presentation of how semantic classification trees are constructed and finally, 3.4 looks at the ATIS evaluation of spoken dialogue systems.

3.1 Application of Decision Trees to Natural Language

Decision trees have been applied largely in many fields; I only mention a few practical situations. IBM used decision trees for probabilistic language modeling by estimating the probability p(w) that a word w will occur given the history provided by previous words [1]. Researchers at the University of Western Ontario have also trained decision trees to guess the past participle of verbs given the present participle. Decision trees were applied to the task of partitioning newspaper articles into classes, by making sure the questions in the trees ask whether a particular word occurs or not [1].

3.2 Semantic Classification Trees

A semantic classification tree (SCT) is a specialized decision tree that learns semantic rules from training data for semantic classification of word sequences [1]. The nodes of the tree are associated with binary questions, each of which admits a “Yes” or ”No” answer that corresponds to the two branches attached to the node. One major characteristic of SCTs is that the generation and selection of these questions is completely automatic, and each question involves regular expressions made up of string symbols, which could be words or gaps of words. There are two main types of SCTs, single-symbol and set-membership SCTs.

3.2.1 Single-Symbol Semantic Classification Trees

Single-symbol SCTs are SCTs that process single words or gaps of words. In this method of growing SCTs, the first thing to do is the creation of the root node by calling a collection of strings, which must have a length of at least one. Each string has its own

class label and the strings must be made up of words that are listed in a vocabulary**.** The next thing to do is to associate each node of the growing tree with a regular expression called the “Known Structure”, which consists of symbols, words and gaps of words. The Symbols '<' and '>' mark the beginning and end of a regular expression, ‘+' indicates a gap of at least one word and the expression M(w) matches one or more occurrences of the word w. For instance, the known structure < +*fare*+ *flights.* > takes care of all the input sentences for which fare is not the first word of the sentence, one or several words follow fare, before flights, just after which a period occurs. After creating the root of the tree which has the known structure <+>, rules that encourage the growth of a complete tree are used to generate the full tree as described in the previous chapter. A SCT node is declared a leaf node when it is no longer possible to perform splits, and this could be because all training items in the node belong to the same category, or there are no gaps (+) in the known structure and thus no further questions to be asked. It is also possible that none of the possible questions gives a split that reduces the Gini impurity.

Fig. 3 shows an example extracted from the Air Traffic Information System (ATIS) [1]. Given the input sentence “show me fare flights to Aachen”, the root node is generated with <+fare+> as explained above, and the process is continued down the tree by trying to find words that match the above request. From Fig. 4, the input matches the pattern <+fare+> at the root, and does not match <+fare code+>, but matches the pattern <+fare flights+> and hence yields NO.

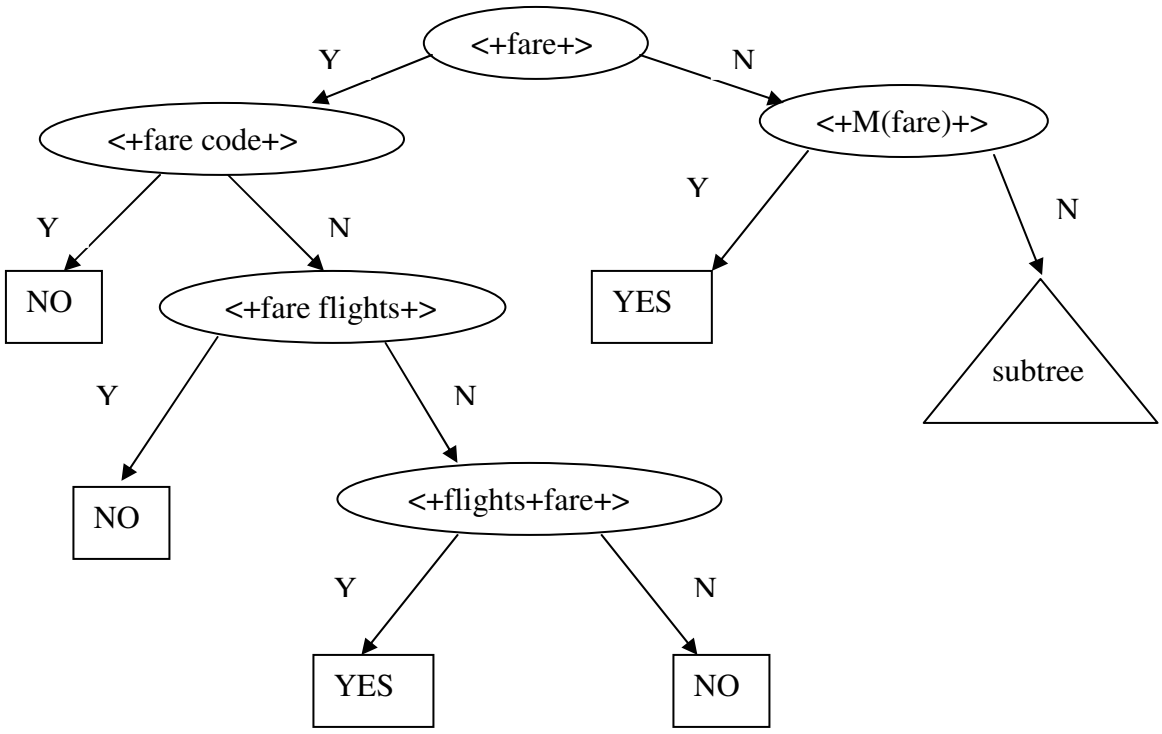

Fig. 4: A Single Symbol SCT for the input "show me fare flights to Aachen "

3.2.2 Set-Membership Semantic Classification Trees

Set-membership SCTs are similar to single-symbol SCTs, but instead of processing single words they ask for the presence or absence of any member of a set of words at each node. Single-symbol SCTs tend to have more No's, because sentences that only differ in synonym go to different sub trees allowing system-defined synonyms. In set-membership SCT, word sets instead of single-symbols are inserted into the gaps in the known structure.

3.3 Building Semantic Classification Trees

There are three key elements that need to be supplied in order to help SCTs learn the semantic rules needed to grow classification trees [1]. These elements are:

1. a set of possible Yes or No questions that can be applied to data items
2. a strategy for selecting the best question at any node depending on the training data (e.g. Gini impurity).

3. a good method for pruning the trees in order to prevent over fitting

In the subsequent sections, I will look the question generation and the Gini criterion.

3.3.1 Question Generation

Given a lexical item w_i, there are four expressions that can be generated from a given gap (+) in the known structure (KS):

I. The expression obtained by replacing + in KS by w_i
II. The expression obtained by replacing + in KS by w_i+
III. The expression obtained by replacing + in KS by +w_i
IV. The expression obtained by replacing + in KS by +w_i+

Hence starting with <+> as the known structure at the root, the expressions <w_i >, <+ w_i > ,< w_i +>,<+ w_i +> can be generated. Each of these expressions is a potential question to be asked. The question that achieves the best split of the training data is selected.

3.3.2 Gini Criterion

The best question for a node T is selected using the Gini impurity. That is, the best question for T is the one which yields the greatest drop in impurity when going from T to its children . The change in impurity at node T is given by:

$$\Delta I(T) = i(T) - p_Y * i(YES) - p_N * i(NO)$$

where:

- i(T) is the Gini impurity at node T
- YES and NO are the child nodes of T
- p_Y is probability of yielding a yes child node
- p_N is probability of yielding a NO child node

The Gini impurity at any node T is defined as

$$i(T) = -\sum_{i \neq j}(p(w_i)\, p(w_j))$$

$p(w_i)$ is the fraction of patterns at node T that are in category w_i.

3.3.3 Semantic Classification Tree Growing Algorithm

The following basic steps could be followed to build a semantic classification tree:

Algorithm 2:

1. *create root node and set known structure (ks) to* <+>
2. *select best question for node using Gini impurity*
3. *test node to see if there are further splits*
4. *if not set to leaf node, else generate YES and NO child nodes*
5. *set YES node to current and go to 2*
6. *set NO node to current and go to 2*

Steps 5 and 6 each become independent paths to follow.

Fig. 5 below gives a version of the algorithm for growing a single-symbol SCT that assumes words are not repeated in a sentence [1]. Expressions of the form K: = B [u←v] in Fig. 5 means the string K is a copy of the string B where the substring u is replaced with the substring v. The SCT is grown by calling grow_SCT() on a collection of strings in which each string is associated with a class label [1]. The procedure expand_node() is called after the root node has been created in grow_SCT(), to create and expand the YES and NO child nodes. At each node, the function find_question() is used to select the best question from the four expressions that can be generated for a given gap in the known structure.

```
function grow_SCT (D: datasettype): nodeptr;
var root: nodeptr;
begin
  create (root);
  root.KS := <+>;
  root.data := D;
  expand_node (root);
  return (root);
end; {end of grow_SCT()}

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

Function stop_cond (node: nodeptr): Boolean;
begin
  if (all items in node data the same
    or have same label) then
      return (TRUE);
    else return (FALSE);
end; {end of stop_cond()}

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

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

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

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

function Δ Gini (Q: question, D:datatype): real
begin
  return decrease in gini impurity when
    split D with question Q
end; {end Δ Gini()}
```

Fig. 5: Growing single-symbol classification trees [1]

3.3.4 Computational Complexity

Given that D is the total number of sentences in a training set, L is the upper limit of the number of words in a sentence, and V is the size of the vocabulary. Then for at most L words in a sentence, there are at most 4V meaningful questions that can be asked along a particular path since it is possible to choose any of the four possible expressions in the known structure. A rough upper bound for the depth of a path will be 4*L*V. Time complexity results for both single symbol and set membership SCTs depend on whether D<4*L*V or D>4*L*V. For D<4*L*V the time complexity for a single-symbol is $O(D^3*L^2*V)$ and that of a set-membership is $O(D^3*L^2*V^2)$. For D>4*L*V the time complexity for a single-symbol is $O(D^2*L^3*V^2)$ and that of a set-membership is $O(D^2*L^3*V^3)$. Further details can be found from [1].

3.3.5 Classification of Substrings

While growing either a single-symbol or a set membership SCT, it becomes necessary to classify multiple occurrences of substrings. The strategy is to submit the same sentence as many times as the number of substrings to be classified. For each submission, the current substring is marked with a "*" symbol. After the first substring has been classified, the next is considered until all substrings have been classified. For instance, given an input question like "show me flights from Aachen, no sorry, from Bonn to Düsseldorf stopping over in Köln?", which after local parsing will be changed to "show me flights from CIT, no sorry, from CIT to CIT stopping over in CIT", we submit "show me flights from *CIT, no sorry, from CIT to CIT stopping over in CIT" to the SCT To classify the first CIT. To classify the second CIT we submit "show me flights from CIT, no sorry, from *CIT to CIT stopping over in CIT to the SCT. This process is continued until the last CIT is classified. Take note of the position of "*" during each classification.

3.4 ATIS Evaluation of Spoken Dialogue Systems

A typical scenario for the presented methods is the Air Traffic Information System (ATIS), which is a DARPA-sponsored project for speech recognition and understanding. ATIS deals with spoken questions about air travel that have been collected and translated

into a database language. There are three ATIS benchmarks

I. ATIS 1: SPREC (speech recognition performance)
II. ATIS 2: NL (natural language understanding of written text)
III. ATIS 3: SLS (spoken language understanding)

In the ATIS environment, a Class ‘A‘ sentence is a sentence that does not depend on previous questions and a Class ‘D‘ sentence is one that depends on previous questions. There were two ATIS 2 (NL) evaluations, one in November 1992, and the other in December 1993. In the next section, I look at CHANEL, which was a participating system of ATIS.

3.4.1 CHANEL

CHANEL is an acronym for “CRIM Hybrid Analyzer for Natural Language”. The system was built at the Centre de recherche informatique de Montreal (CRIM), and partially supported by the Institute for Robotics and Intelligent Systems in Canada. It uses a SCT-based linguistic analyzer for NLU with the aim of generating database queries for natural language based inputs. CHANEL takes spoken language as input and translates it into an efficient SQL query. For example if a question like “show me 10 am flights from Berlin to Aachen and how much they cost ” is asked, it is processed by a local chart parser which looks for words or phrases carrying information that should be incorporated into the SQL query. These words or phrases are replaced by three letter symbols carrying the associated meaning. And the sentence is then passed on to a robust matcher, which has two parts, one part that displays a list of attributes using a forest of SCTs contained in CHANEL. These SCTs return either a zero or one indicating whether the attribute should be displayed or not. The other part of the robust matcher uses role assignment SCTs to assign roles to constraints. for instance, CIT is the role assignment SCT for city which could have possible values like origin, destination, stopover etc. The information generated by the robust matcher is passed on to a SQL module, which in turn generates the SQL query. The above description can be found diagrammatically in Fig. 6 below.

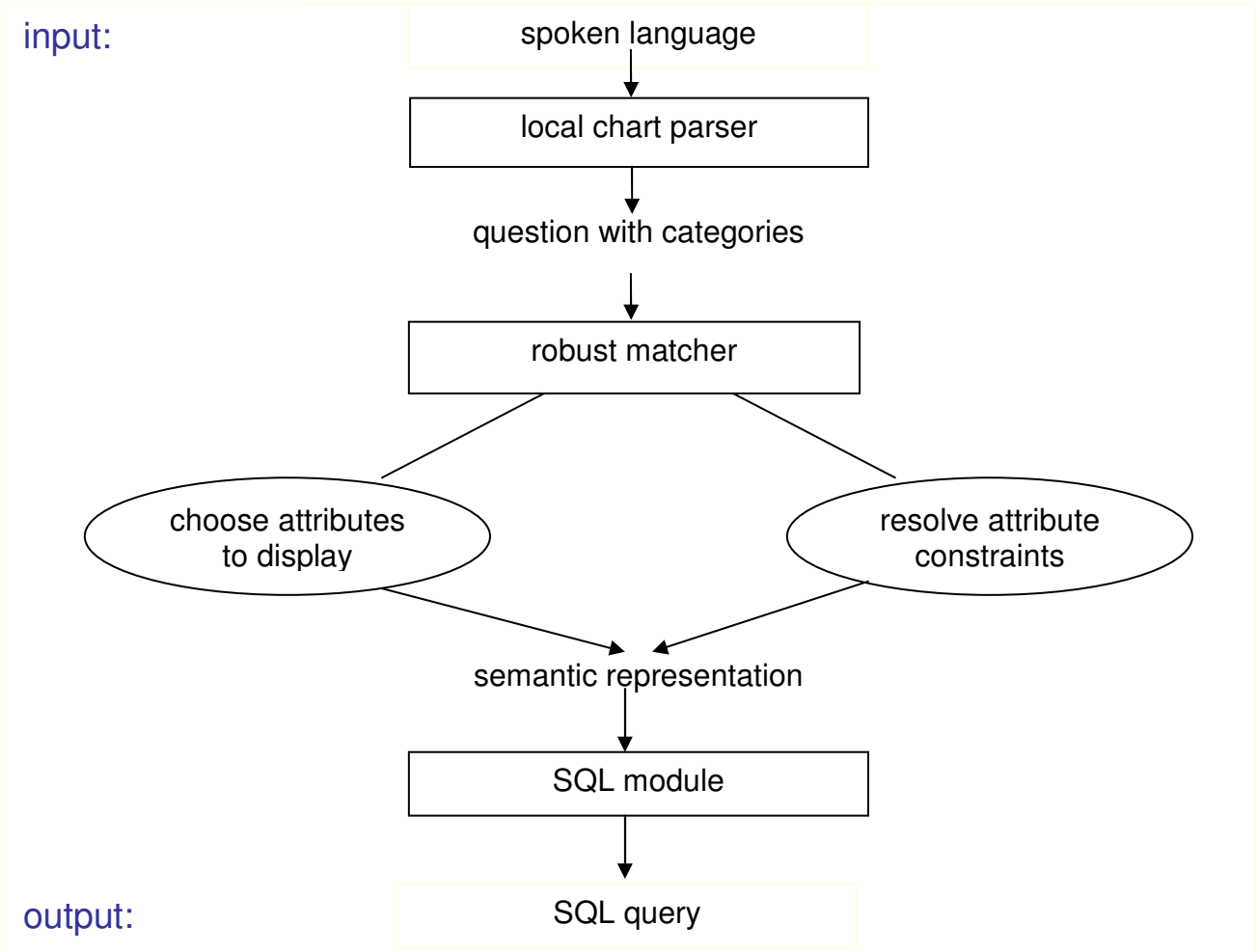

Fig. 6: The CHANEL Architecture

3.4.2 Results

In November 1992 and December 1993, there were two different evaluations of the ATIS systems. In the November 1992 evaluation, the official criterion was weighted error while in December 1993 the official criterion was un weighted error. The CHANEL approach also contained some hand coded rules that return "NO ANSWER" when local constraint roles clash [1]. Weighted error penalizes a false answer twice as a "NO ANSWER" whilst un weighted error penalizes a false answer and "NO ANSWER" equally. The results in table I and II show ATIS class 'A' NL un weighted and weighted error as presented by [1]. The two systems compared to the CHANEL method were AT & T's Chronus, which made use of HMMs, and the context free grammar based HUM approach developed by BBN. After the test, it was realized that some simple bugs in the non-SCT

portions of CHANEL had affected its performance. Therefore, the official version of CHANEL (CHANEL (o)) was debugged (CHANEL (d)) and the test was re conducted. The tables below show a comparison of the above mentioned approaches.

Table I: November 1992 NL 'A' Benchmarks

systems	method used	NL 'A' un weighted error [%]	NL 'A' weighted error [%]
ATT	HMM	19.5	34.7
BBN	CFG	11.1	15.7
CHANEL	SCT	30.2	40.5

Table II: December 1993 NL 'A' Benchmarks

system	method used	NL 'A' un weighted error [%]	NL 'A' weighted error [%]
ATT	HMM	7.4	14.7
BBN1	CFG	9.6	19.0
BBN2	CFG	16.1	31.9
CHANEL (o)	SCT	21.7	42.2
CHANEL (d)	SCT	12.3	23.9

4 Conclusion

In this seminar, I have presented natural language database queries using classification and regression trees. I gave a brief introduction to decision trees and presented Semantic classification trees (SCT) which are specialized decision trees. SCTs were used as a natural language understanding component needed to achieve the aim of this seminar. Two types of SCTs were presented in section 3.2. The ATIS speech understanding task was used as a test bed. I also looked at the architecture of CHANEL, which was a participating system in ATIS. As discussed in chapter 3, the building blocks of CHANEL are SCTs which are specialized decision trees. The objective of SCTs is to learn semantic rules from training data for semantic classification of word sequences. There were two evaluations of the ATIS, in November 1992 and December 1993 and the results obtained from CHANEL were compared to results obtained from other approaches.

References

[1] R. Kuhn, R. De Mori: "The Application of Semantic Classification Trees to natural Language Understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 17, No. 7, pp. 449-460, 1995.

[2] R. Kuhn, R. De Mori: "Sentence Interpretation," in *Spoken Dialogues with Computers*, R. De Mori (ed.), Chapter 14, pp. 485-522, Academic Press, San Diego, CA, 1998.

[3] R.O. Duda, P.E. Hart, D.G. Stork: *Pattern Classification*, John Wiley and Sons, New York, NY, 2001, pp. 395-411.

[4] James Allen. 1995. *Natural Language Understanding*. 2nd Edition. Addison-Wesley.

[5] An Introduction to IBM Natural Language Understanding an IBM White Paper 1993-2001.

[6] Prof. Dr.-Ing. Hermann Ney, *Statistical Natural Language Processing*, RWTH Aachen, Computer Science Department
Summer Semester 2003

[7] Roger Evans, *Natural Language Understanding*, ITRI, University of Brighton

[8] H.Hamilton. E. Gurak, L. Findlater W. Olive, *Overview of Decision Trees*.

[9] L. Rabiner, *A tutorial on Hidden Markov Models and selected applications in speech recognition*, 1989, Proc. IEEE 77(2):257—286

[10] Klaus Macherey, Franz Josef Och, Hermann Ney. "*Natural Language Understanding Using Statistical Machine Translation*". "EUROSPEECH 2001 - 7th European Conference on Speech Communication and Technology", pp. 2205-2208, Aalborg, Denmark, September 2001.