DT: A Classification Problem Solver with Tabular-Knowledge Acquisition

Yoshiyuki Koseki, Yoichiro Nakakuki, Midori Tanaka
C&C Systems Research Laboratories
NEC Corporation
Kawasaki, 216 JAPAN

Abstract

This paper describes an approach to integrating a classification problem solving function and an interactive knowledge acquisition function on a tabular knowledge representation. A tabular knowledge base presents concept functions in DNF (Disjunctive Normal Form), which handles disjunctive concepts with multiple-value attributes. Because of the peculiar knowledge base construction, which represents concept functions with positive and negative disjuncts, the knowledge refinement procedures are kept simple, and changes in the knowledge base are comprehensible to users. Based on the approach, a system called DT was developed. When the user finds an incorrect answer, he can immediately tell the system to correct it through a simple question and answer routine. The system corrects the knowledge base by generalizing, specializing and differentiating the concepts.

1 Introduction

Since the advent of expert system technology in the 1970's, the technology has been applied to many application fields. Most of the tasks solved by the technology are categorized as heuristic classification problems [2]. Construction of a heuristic classification knowledge base for a target problem field has been recognized to be a challenging issue as knowledge acquisition bottleneck.

In recent years, a number of attempts have been made to realize knowledge acquisition functions for expert systems, in order to overcome the bottleneck. Among them, one class of works takes an induction approach, which generates general descriptions from a number of given examples. The systems in this category include ID3 [16], INDUCE [9] and others. A disadvantage of those systems is that they require a significant number of examples to construct a knowledge base.

Another approach is the knowledge refinement or the knowledge apprenticeship approach, which incrementally refines the knowledge base by watching an expert. Most of them use background knowledge to deduce classification knowledge, as formalized in the methods, such as EBG [11] and EBL [3]. There have been some systems developed, based on this approach, including those reported by [18], [7] and others. They are effective methods for some applications whose background knowledge construction is straightforward. However, background knowledge construction is sometimes as difficult as classification knowledge construction.

This paper describes a novel approach to integrating a classification problem solving function and an interactive knowledge acquisition function. In contrast to those previously reported methods, which construct knowledge bases for predefined set of concepts and attributes, the presented approach acquires new concepts and new attributes through interaction with users. Moreover, it does not require many examples nor background knowledge. Unlike rule-based methods, the presented approach utilizes a tabular knowledge representation. A tabular knowledge base presents concept functions in DNF (Disjunctive Normal Form), which handles disjunctive concepts with multiple-value attributes. Because of the peculiar knowledge base construction, which represents concept functions with positive and negative disjuncts, the knowledge acquisition procedures are kept simple, and changes in the knowledge base are comprehensible to users.

Based on the approach, a system called DT was developed. The aim of the system is to lead expert users, who are not familiar with computer technology, to construct knowledge bases, while they use the problem solving functions. When a user finds an incorrect answer, he can immediately tell the system to correct it through a simple question and answer
routine. The system corrects the knowledge base by generalizing, specializing and differentiating the concepts in the knowledge base to produce a correct answer. The system also learns the frequency of successful cases. In order to minimize the total number of attributes asked, it chooses an attribute with maximum entropy gain per question cost, according to the frequency learned. The system has been field-tested for a practical machining tool selection problem with a considerable size. The interactive knowledge acquisition function has been shown to be effective in allowing a domain expert who is not familiar with computer operations to construct the knowledge base.

2 Problem definition

This section presents the classification problem which the authors deal with. Consider a set of concepts $X = \{x_1, x_2, \ldots, x_m \}$ and a set of attributes $A = \{a_1, a_2, \ldots, a_n \}$. The domain for each attribute $a_j$ is a finite value set $Dom(a_j)$. The classification knowledge for concepts is given as a set of logical functions of attributes for concepts. In a diagnostic problem, for example, $X$ corresponds to a set of possible failures and $A$ is a set of observable symptoms or test results. In a machining tool selection problem, $X$ is a set of available tools and $A$ is a set of criteria for a selection. The problem is to find a subset of $X$ which satisfies a given set of attribute values.

Consider a simple example for a grinder selection problem in a hypothetical factory. The problem is to select an appropriate grinder for a product, according to product attributes, such as material feature and required quality. In the factory, two kinds of tools, Grinder-H and Grinder-L, are available. Expensive Grinder-H can grind both hard and soft materials and it produces high quality products. In contrast, inexpensive Grinder-L can only be used for soft materials and it produces lower quality products. To minimize product cost, the factory uses the expensive grinder only when it is necessary. Therefore, for hard materials, Grinder-H must always be used. When the required quality is high, Grinder-H must also be used even for soft materials. Grinder-L can only be used for products which are made of soft materials and which do not require high quality. The following concept functions represent this grinder selection problem.

Example grinder selection problem
Concepts:
$X = \{\text{Grinder-H, Grinder-L}\}$
Attributes:
$A = \{\text{Material, Quality}\}$

2.1 Attribute Domains:
$Dom(\text{Material}) = \{\text{Hard, Soft}\}$
$Dom(\text{Quality}) = \{\text{High, Low}\}$

2.2 Concept Functions:
Grinder-H = Material(Hard) \lor (Material(\text{Soft}) \land \text{Quality(High)})
Grinder-L = Material(\text{Soft}) \land \text{Quality(Low)}$

Here, two characteristics are considered regarding the availability of attribute values. First, not all of the attribute values are available in advance. Therefore, the problem solver should actively seek relevant attribute values. Thus, it would be best not to seek irrelevant attributes in order to avoid useless effort. Suppose in the example, the system was told that required quality is low, but was not told about its material. Because knowing that the quality is low is not sufficient to distinguish between Grinder-H and Grinder-L, the system should ask the user about its material. If the system was told that the quality is high, this information is sufficient to decide that the answer is Grinder-H. Second, even if the problem solver asks for an attribute, there are some cases where the value is not available at all. Even in those cases, the system should give the best answer according to the available information. For example, when the required quality is low but material is unknown, the system should give both of the grinders as an ambiguous answer.

3 System configuration

Figure 1 shows an overview of the described system. The system consists of a problem solver which is integrated with an interactive knowledge acquisition function, a knowledge base editor, and a tabular knowledge base. Users have multiple interfaces to the knowledge base. He may use textual rule expression or textual tabular expression, which may be used for interfacing with other knowledge bases.

Using the knowledge base editor, he can edit the knowledge base utilizing its symbolic editing function. The example system display of Figure 2 shows how he can accomplish this task. Knowledge base editor has a couple of functions to help constructing knowledge bases, namely minimization and simulation. Minimization function shrinks the size of the knowledge base while keeping the concept function logics. Logic minimization on a tabular representation corresponds to two-level logic minimization. Classical two-level minimization algorithms, which seek for minimum solutions, suffer from combinatorial explosion [12] and the maximum size of the problem was limited to around 20 attributes. However, recent heuris-
Symbolic editing
Minimization
Problem Solver
Candidate elimination
Question selection
Knowledge Acquisition

Figure 1: System Configuration

The simulation function of the knowledge base editor allows users to verify a knowledge base under development. The simulator is bidirectional, that is, users can investigate resulting concepts for a given combination of attributes and also can examine combination of attributes for a given set of concepts.

Problem solver accomplishes a classification process by executing candidate discrimination cycle. It dynamically chooses an appropriate attribute to be acquired, according to its information gain per attribute cost. Problem solver can call multiple tables in a hierarchical manner. This enables dividing a large knowledge base into multiple tables. If a table is divided into small tables all of which consist of single attribute, the entire knowledge base becomes equivalent to a decision tree.

4 Tabular knowledge representation

The technique to represent knowledge in a tabular form has been known as Decision Tables [5]. This construct is also found in VLSI design, as PLA (Programmable Logic Array), which implements a hardware logic in a two level AND-OR logic array [6].

The tabular knowledge base represents a set of concept functions, which are expressed in DNF (Disjunctive Normal Form). This construct enables handling disjunctive concepts with multiple-value attributes. A function consists of a number of disjuncts, called cubes. A disjunction of all the cubes for a concept forms a concept function, and it is called a cover for the concept. A cube consists of parts, which designate a set of values for an attribute.

The example problem knowledge can be represented in a tabular form as shown in Table 1. Here, each row forms a cube, and a set of cubes for the same concept forms a cover for the concept. For example, the concept cover for Grinder-H consists of two cubes, and each cube consists of two parts. In a cube, the values specified by the circle (0) in a part are to be ORed and those parts are ANDed to form a cube. Unknown or Don't Care attributes are designated as a part with all (0)'s. A null cube has at least one part which has no 0's at all.

<table>
<thead>
<tr>
<th>Material</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hard</td>
</tr>
<tr>
<td>Grinder-H</td>
<td>☐</td>
</tr>
<tr>
<td>Grinder-H</td>
<td>☑</td>
</tr>
<tr>
<td>Grinder-L</td>
<td>☑</td>
</tr>
</tbody>
</table>

Table 1: Tabular Representation Example

Advantages of the tabular representation can be summarized as follows. First, the content is comprehensible to humans, because tables provide multiple views of a knowledge base. For example, when a concept is given, a user can easily find which combination of attributes gives the concept as an answer. When a set of attributes is given, we can also see which concepts are selected as an answer. Comparing multiple concept functions is also an easy task on this representation. Second, knowledge refinement functions are easily implemented, because generalizations and specializations on the concepts can be accomplished by simply adding and deleting cubes in a table. Third,
it can provide an answer even if some of query attributes are unknown. Fourth, tabular representation explicitly separates control knowledge from logical knowledge. In contrast, rule-based representation gives some orderings for attributes. Ideally, inference mechanisms should not be influenced by those orderings. However, in most of available rule-based inference systems, the order of the attributes to be asked is affected by the knowledge structure.

A disadvantage of the tabular representation is that it may become sparse and it may take up too much space. In the worst case, it requires \(2^n-1\) cubes for the parity function of \(f = x_1 \oplus x_2 \oplus \cdots \oplus x_n\). This situation can be avoided by using hierarchically divided tables. For instance, this function can be represented as \(2n-1\) serially connected tables, each of which realizes two-input exclusive-or function.

The system allows to express concept functions in different ways by using special covers called negative covers. To distinguish normal covers from negative covers, we call them positive covers. Using a positive cover \(PC\) and a negative cover \(NC\), a concept \(C\) is expressed as follows.

\[ C = PC \land NC \]

Cubes which consist of a negative cover are called negative cubes. In the example problem, the condition to select Grinder-H can be differently expressed as; “always use Grinder-H, except when the material is soft and the required quality is low.” Table 2 shows an alternative expression for the example knowledge, based on this interpretation. The sharp operator (\#) declares that this cube is a negative cube.

<table>
<thead>
<tr>
<th>Material</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard</td>
<td>Soft</td>
</tr>
<tr>
<td>Grinder-H</td>
<td>(\square)</td>
</tr>
<tr>
<td>#Grinder-H</td>
<td>(\times)</td>
</tr>
<tr>
<td>Grinder-L</td>
<td>(\times)</td>
</tr>
</tbody>
</table>

Table 2: Negative Cube

Like this example, by using negative cubes, it may be able to express concept functions in a more intelligible way. In addition, it may be possible to reduce the number of cubes which are necessary for expressing a concept function. Moreover, this makes the concept specialization procedure a matter of only a simple cube addition, as described in a later section. This has an additional advantage that the refinement becomes transparent to the users.

5 Problem solver

The problem solver starts with all of the concepts in a knowledge base as candidates for the answer. As it acquires attribute values incrementally, cubes which have become disjoint from the set of the attributes are eliminated from the candidate set. This is done by checking the covering between query attribute values and individual candidate cubes. Then, it selects an effective attribute to be asked about next. This cycle continues, until it satisfies a terminating condition.

5.1 Candidate calculation

This subsection describes a procedure used to calculate a set of candidates for a given set of attribute values. Let \(PC\) denote a positive cover for a concept \(C\) and let \(NC\) denote a negative cover for the concept. \(QC\) is the given query cover. The concept \(C\) is a candidate for the answer when the following condition is true.

\[ (PC \land QC) \# NC \neq \phi \]  

(1)

Here, AND of two covers, \(A \land B\) is operated by taking OR of every AND cube operation of consisting cubes. The OR of two cubes, \(C_1\) and \(C_2\), is a concatenation of the two cubes. The AND of two cubes, \(C_1\) and \(C_2\), is a cube formed by the bit-by-bit AND of the two cubes. The sharp operation \(A \# B\) is defined as \(A \land \overline{B}\). A procedure for this operation is well known and described in [8]. AND operation and sharp operation of two cubes are accomplished in time proportional to the number of parts in those cubes. Therefore, the time complexity for calculating the condition (1) is \(O(np)\), where \(n\) is the total number of cubes in \(PC\) and \(NC\) for the concept \(C\), and \(p\) is the number of parts in those cubes.

When a cube \(PC_i\) in \(PC\) and \(QC\) are mutually disjoint, that is \(PC_i \land QC = \phi\), \(PC_i\) does not contribute to calculating the condition (1). Similarly, when a cube \(NC_i\) in \(NC\) and \(QC\) are mutually disjoint, that is \(NC_i \land QC = \phi\), \(NC_i\) does not contribute to calculating the condition (1). Because of this, it is possible to eliminate those disjoint cubes from the candidates in the early stage of the problem solving cycle to achieve efficient computation.

5.2 Question selection function

The problem solver controls an order of questions to minimize the number of questions, while maintaining a natural flow of logic for the user.

To maintain natural conversation flow, the system utilizes some constraints, specified in the knowledge base. Those constraints are question dependency,
question prohibiting concepts, question grouping, and concept class hierarchy. At first, the system selects a set of attributes which do not violate those constraints. Then, among selected attributes, it chooses an attribute with the highest information-theoretic effectiveness, divided by an attribute-acquiring cost.

To compute information-theoretic attribute effectiveness, the system uses entropy gain function which is calculated from occurrence probabilities for candidate concepts. The mechanism is basically the same as the one found in a model-based diagnostic system [4], called a minimum entropy technique. This function is also similar to the gain function of Quinlan's decision tree learning algorithm ID3 [16].

Here, an entropy $E(D)$ is defined for a set of candidate concepts $D = \{d_1, d_2, \ldots, d_n\}$. Let $p_1, p_2, \ldots, p_n (\sum p_i = 1, p_i > 0)$ be occurrence probabilities for candidates $d_1, d_2, \ldots, d_n$. Then an entropy $E(D)$ is defined as

$$E(D) = -\sum_{i=1}^{n} p_i \log p_i.$$

Let sets of candidates, after asking about an attribute $a$, be denoted by $D_1, D_2, \ldots, D_k$, when $a$ has multiple possible values, $v_1, v_2, \ldots, v_k$. Let the occurrence probability for a value $v_j$ be expressed by $q_j$. Then, the effectiveness of the attribute $a$ is defined as

$$\text{gain}(a) = \sum_{j=1}^{k} q_j (E(D) - E(D_j)).$$

Here, the system assumes that the probability distribution after attribute acquisition remains the same, and that occurrence probabilities for attribute values are all the same.

To presume concept occurrence probabilities, each concept in a table has a counter, which counts the successful usage frequency. This frequency is used for presuming probabilities by the Bayesian method in the current implementation. For situations where the amount of observed data is small, a novel mechanism to induce a probability distribution is being carried out by the authors [14].

6 Interactive knowledge acquisition

6.1 Sample session

The following sample sessions show how the implemented system DT interacts with the users for the example grinder selection knowledge. In the sessions, the question asked by the system is typed in italics with the menu items parenthesized. The user’s reaction is typed in bold face. Actual interfacing is all accomplished in a menu-driven manner in a multi-window environment.

**Example Session 1 (Generalization and specialization)**

Suppose that the factory has improved the Grinder-L to be able to produce high quality products. Therefore, a domain expert would like to modify the knowledge base, making it to answer "Grinder-L" instead of "Grinder-H" for the situation that the material is soft and the required quality is high.

Is the material hard or soft? (Hard, Soft, Unknown, Postpone) . . . Soft.

Is the required quality high or low? (High, Low, Unknown, Postpone) . . . High.

Please use "Grinder-H".

By asking two questions, in the order of importance, the system gave a solution according to the old knowledge base. Using the interactive knowledge refinement function, the user can refine the knowledge base as follows.

Is there any incorrect extra answer? (Yes, No) . . . Yes.

Is there any missing answer? (Yes, No) . . . Yes.

What is the correct answer? (Grinder-H, Grinder-L, Other) . . . Grinder-L.

Is it appropriate to correct the knowledge base, so as not to answer "Grinder-H"? (Yes, No) . . . Yes.

Is it appropriate to correct the knowledge base, so as to answer "Grinder-L"? (Yes, No) . . . Yes.

The knowledge base is refined. Thank you.

From now on, the system will answer "Grinder-L" instead of "Grinder-H" for the same question.

**Example Session 2 (Differentiation)**

Suppose that the factory has developed a new low-cost grinder called Grinder-LL which processes only small soft materials producing lower quality products. The difference between the former Grinder-L and the new Grinder-LL is that the Grinder-LL is only usable for small materials. Therefore, a domain expert would like to modify the knowledge base, so that the system answers "Grinder-LL" instead of "Grinder-L" for the situation that the material is soft, the required quality is low, and the material is small.

The user is shown a whole list of available attributes, which he may modify. In this case, it is "material is soft and quality is high".

160
Is the material hard or soft? (Hard, Soft, Unknown, Postpone) ... Soft.
Is the required quality high or low? (High, Low, Unknown, Postpone) ... Low.
Please use "Grinder-L".
Is there any incorrect extra answer? (Yes, No) ... Yes.
Is there any missing answer? (Yes, No) ... Yes.
What is the correct answer? (Grinder-H, Grinder-L, Other) ... Other.
Please type in the correct answer. ... Grinder-LL.
Is it appropriate to correct the knowledge base, so as not to answer "Grinder-L"? (Yes, No) ... No.
Is it appropriate to correct the knowledge base, so as to answer "Grinder-LL"? (Yes, No) ... Yes.
Can you distinguish "Grinder-L" from "Grinder-LL" by adding a new attribute? (Yes, No) ... Yes.
Please type in the attribute. ... Size.
Please type in a value related to "Grinder-LL" but not to "Grinder-L". ... Small.
Please type in a value related to "Grinder-L" but not to "Grinder-LL". ... Large.
The knowledge base is refined. Thank you.

From now on, the system will ask "Material Size", and it will successfully distinguish "Grinder-L" from "Grinder-LL".

6.2 Refinement method

The refinement is carried out by three procedures: generalization, specialization and differentiation. Generalization and specialization are accomplished by adding cubes to the tabular knowledge base. Differentiation is accomplished by adding new attributes.

After the system gives an answer to the user, it asks for the correctness of the answer. If the user thinks the answer is correct, the system counts up the correct answer frequency. If the user finds an incorrect answer, he can specify which concept in the answer is correct and which one is not. If a concept is mistakenly included in the answer, the system specializes the concept cover. When a correct concept is not found in the answer, the system generalizes the concept cover. When multiple concepts are given as an answer and the user would like to differentiate them, he can give a new attribute by the differentiation procedure.

Because of the tabular representation, the refinement procedures are straightforward. They are accomplished by simply adding rows and columns to the table and by modifying the symbols in the table. The following subsections describe procedures for generalization, specialization and differentiation.

6.3 Refinement functions

6.3.1 Generalization

Generalization is implemented by adding a cube and subtracting the same cube from the negative cover for the concept. A cube to be added is constructed according to the acquired attribute values during the problem solving session. When the adding concept is an entirely new one, the system asks the user to type a new keyword for the concept, and the system simply adds a cube.

In Example Session 1, the concept function for Grinder-L is generalized to include the condition that material is soft and quality is high. To accomplish this refinement, a new cube, shown in Table 3, is added to the example table.

<table>
<thead>
<tr>
<th>Material</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard</td>
<td>Soft</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 3: Added Cube for Generalization

6.3.2 Specialization

Specialization simply adds a negative cube. When an incorrect concept is included in the answer, the system adds a negative cube to the knowledge base. As shown in Example Session 1, when the user indicates that Grinder-H is not an answer for the specific condition, the system adds a negative cube, shown in Table 4.

<table>
<thead>
<tr>
<th>Material</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard</td>
<td>Soft</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>Low</td>
</tr>
</tbody>
</table>

Table 4: Added Negative Cube for Specialization

The system simply merges Table 3 and Table 4 with Table 1 to form a merged table shown in Table 5. Although this table is logically correct, it is redundant. The logic minimization function produces a final optimized table, shown in Table 6.

6.3.3 Differentiation

After acquiring new knowledge, the system checks whether multiple concepts are obtained as answer for the given situation. Then, the user is asked whether he can think of a new attribute to distinguish between
them, and the system asks him to input the attribute. This situation was found in the Example Session 2.

In the example, after acquiring new knowledge for a new Grinder-LL, the system finds that it is not possible to distinguish Grinder-LL from Grinder-L with given attributes. This situation is shown in Table 7.

Table 7: Before Differentiation

<table>
<thead>
<tr>
<th>Material</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard</td>
<td>Soft</td>
</tr>
<tr>
<td>Grinder-H</td>
<td>☒</td>
</tr>
<tr>
<td>Grinder-H</td>
<td>☒</td>
</tr>
<tr>
<td>Grinder-LL</td>
<td>☒</td>
</tr>
<tr>
<td>Grinder-L</td>
<td>☒</td>
</tr>
</tbody>
</table>

Table 8: Differentiated Table

<table>
<thead>
<tr>
<th>Material</th>
<th>Quality</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard</td>
<td>Soft</td>
<td>High</td>
</tr>
<tr>
<td>Grinder-H</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Grinder-H</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Grinder-L</td>
<td>☒</td>
<td>☒</td>
</tr>
<tr>
<td>Grinder-LL</td>
<td>☒</td>
<td>☒</td>
</tr>
</tbody>
</table>

Table 5: Merged Redundant Table

Table 6: Merged and Minimized Table

7 Implementation

The system is implemented on a UNIX\(^2\) based workstation and was written entirely in C language.

8 Discussion

This work is closely related to other concept formation systems, such as UNIMEM [8] and EITHER [13]. Those systems also produce concept functions from

\(^2\)UNIX is a trade mark of AT&T
observations incrementally, but in hierarchical concept representations. The method described in this paper produces a tabular concept representation which has many advantages over hierarchical representations. First, the knowledge base is understandable for humans, by providing multiple views. Second, knowledge refinement procedures can be implemented in a straightforward way, so that the changes in the knowledge base is transparent to humans. Third, problem solver can deal with incomplete queries. Fourth, the knowledge base excludes control knowledge.

Some users prefer hierarchical representations to tabular ones. It is possible to convert a tabular representation into a hierarchical expression, by keeping the trace of classification execution for all of attribute combinations. It is also a straightforward work to convert hierarchical representations into a table, by simply expanding the leafs of the tree.

Some other expert systems have a function to handle uncertainty. One widely used method is to use a value called CF (Certainty Factor), which was introduced in the MYCIN system [17]. The described system basically does not handle uncertainty. However, it can implement a similar function, by providing answers with different certainty values for the same concept.

As future work for the system, some enhancements are planned. Unlike other induction based knowledge acquisition systems, this system does not accomplish any inductive inference. It is possible to incorporate induction procedures into the system, which induce more general rules out of the table. In this case, the cubes in the table are treated as examples for the induction procedure.

9 Conclusion

This paper has described an approach to integrating classification problem solving functions and knowledge acquisition functions on a tabular representation. Because of the simple knowledge base construction, knowledge refinement procedures are implemented in a straightforward manner.

Acknowledgment

The authors express their appreciation to Takeshi Yoshimura of NEC Corporation for giving them the opportunity to pursue this research.

References