Knowledge Engineering Process for a Rapid Prototyping of Inductive Expert System

Nittaya Kerdprasop and Kittisak Kerdprasop

**Data Engineering Research Unit, School of Computer Engineering,
Suranaree University of Technology, 111 University Avenue,
Nakhon Ratchasima 30000 Thailand**

Abstract

The main characteristic of current expert systems is the separation of a knowledge base that may be changed from one application to another from the inference engine that still remains the same across applications. The delay in the development of many expert systems is due to the difficulty in acquiring and eliciting knowledge from the human domain experts. The concept of inductive expert system is thus been devised to overcome such bottleneck by incorporating automatic knowledge acquisition module in the system. According to this new concept, knowledge can now be induced or learned in an automatic way from archived databases that are normally available in most organizations. In this paper, we propose an architecture of the inductive expert system that includes the knowledge engine part to automatically forming expert rules from the stored data. We explain the automatic knowledge creation technique through a simple running example, then followed by a real application. We also provide our Prolog source code in appendices for knowledge engineers to apply our technique as a rapid prototyping of their own expert systems.

Keywords: *Expert Systems, Intelligent Knowledge Base, Machine Learning, Knowledge Engineering.*

1. Introduction

Since the release of DENDRAL in the 1960s from the Stanford Heuristic Programming Project [5] as the first practical knowledge-driven program, expert systems have enormously proliferated and been applied to all areas of computer-based problem solving. The inventors of DENDRAL system have introduced the novel and important concept of knowledge base separation in that the content of knowledge could be added and refined independently from the program module, called the inference engine, that interprets and uses that knowledge. The loosely coupling of a knowledge base and an inference engine is an influential concept to all successor rule-based expert systems such as MYCIN [10], INTERNIST-1 [6], and many others.

Since the 1980s expert systems, also called knowledge-based systems, have shifted from the medical and scientific application domains to various areas. In manufacturing and other engineering applications, rule-based expert systems are commonly applied to solve optimization problems, plan manufacturing scheduling, diagnose equipment failures, and use in almost every stage of the manufacturing process [2]. The increasing popularity of rule-based expert systems is due to the simplicity of the *if-then* rules that are easy to comprehend by humans. Many expert system tools such as Clips and Jess are available as a rule engine to facilitate rule generation for a knowledge base. These tools help facilitating knowledge representation, but knowledge acquisition and elicitation are still the labor-intensive tasks facing most knowledge engineers.

Modern expert system development process has thus moved toward the automating methodology by applying intelligent knowledge extraction techniques. Such intelligent techniques can be acquired through the machine learning and data mining technologies. There have been increasing numbers of research work attempting to apply learning techniques to automatically extract end elicit knowledge [1], [3], [4], [7], [8], [11]. These attempts have pushed the current expert system technology to the next generation of an inductive expert system in the sense that besides the knowledge base and the inference engine, the system now includes the learning component.

The research work presented in this paper takes the same direction as most researchers in an attempt to automate knowledge extraction and elicitation with machine learning and data mining techniques. Our work, however, is different from others in that not only proposing an architecture of the learnable inductive expert system and experimenting with some learning algorithms, but we also design and develop a full complement of the rule-based expert system. The work presented in this paper covers the knowledge mining from existing databases, knowledge transfer as a set of rules to be stored in the knowledge base, and knowledge reasoning through a logic-based inference engine. Program source code for the whole process also provided in appendices.

2. A Framework for Automatic Knowledge Base Creation

We design (in Fig. 1) an architecture of the inductive expert system to include the knowledge engine facility. This part of the system requires a machine learning algorithm and a training dataset. The learning algorithm used in our work is based on the ID3 algorithm [9] because the structure of induced tree is appropriate for generating reasoning and explanation in the expert system shell. The induced knowledge is to be generated in a format of decision rules incorporated with probabilistic values. This value is intended to be used as the degree of potential applicability of each decision rule. The probabilistic values are indeed the coverage values of decision rules and can be computed as a proportion of (*number of instances at leaf nodes*) / (*total data instances in a training dataset*).

The steps graphically shown in Fig.2 are the process to generate decision rules to be stored in the knowledge base. These rules are to be used by the inference engine for giving recommendation to users. Consulting rules are for reasoning and giving explanation when requested by the users.

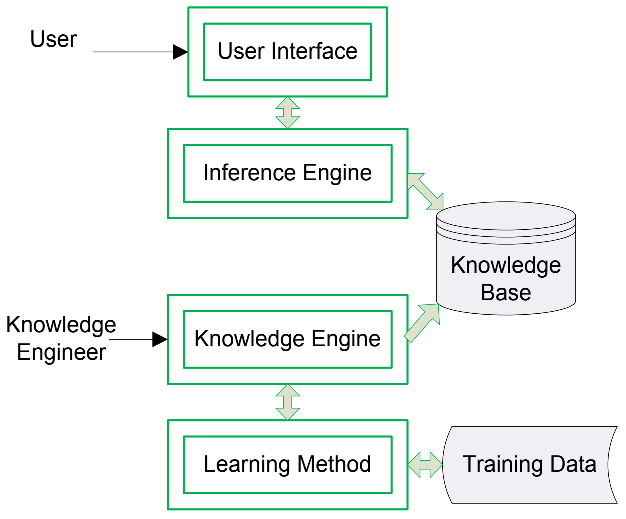


Fig. 1 Architecture of the inductive expert system.

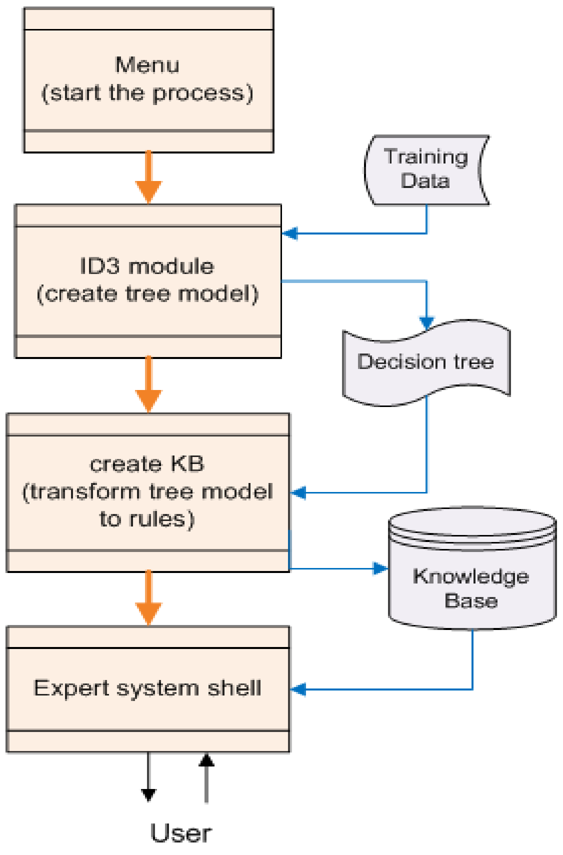


Fig. 2 Automatic knowledge engineering process.

3. Running Example

3.1 Training Data for Building a Tree Model

To explain the idea proposed in the previous section, we provide a running example through a simple training dataset as illustrated in Fig. 3. The given data contain information regarding color and shape of three objects and their classified class as either yes (the right object), or no (the wrong one). Our objective is to learn a decision model from this small dataset and extract a model in a form of a decision tree that to be helpful in identifying objects in the future with unknown class. The first step is converting data format to fit the program. Most data in the databases are represented as table. Appropriate format as required by our Prolog program is the one shown below the table in Fig.3. This converted data has been saved in a file 'shape.pl', and is to be used as a training dataset in the next step.

color	shape	class
red	round	yes
blue	polygon	yes
green	square	no

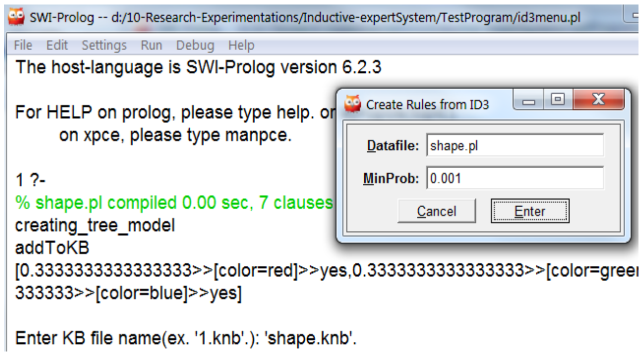
```
attribute(color,  [red, green, blue]).
attribute(shape,  [round, polygon, square]).
attribute(class,  [yes, no]).

instance(1,  class=yes,  [color=red,    shape=round]).
instance(2,  class=yes,  [color=blue,   shape=polygon]).
instance(3,  class=no,   [color=green,  shape=square]).
```

Fig. 3 A sample shape dataset that contains three instances.

3.2 Tree Model and a Transformed Knowledge Base Rule

Once the training dataset has been prepared, the next step is to build a tree model from the data. This can be done through invoking the program ‘id3menu.pl.’ A small dialog box will be popped up (as shown in Fig. 4) to ask the file name of training data. The parameter ‘MinProb’ is for pruning a tree model. The more the value, the shorten the tree model. Default value of this parameter is 0.001, which should be small enough for most moderate size data.

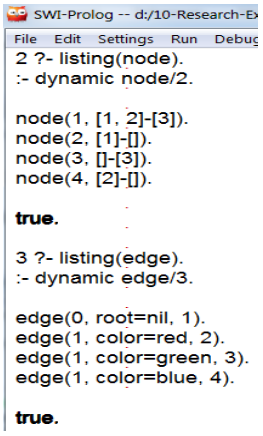
When user clicks the ‘Enter’ button, the dialog box disappears and the program starts building a tree model. This model is actually a data structure of nodes and edges (as illustrated in Fig.5). User will then be asked to input the file name to store the model. In this example, we store a model in the file named ‘shape.knb’. Content of this file (displayed in Fig.6) is automatically created by the ‘id3menu.pl’ program. The program traverses the tree model and converts the structures of nodes and edges into rules. The created file, ‘shape.knb’, is a knowledge base induced from the training data and can be consulted by the inference engine of the expert system shell.


SWI-Prolog -- d:/10-Research-Experimentations/Inductive-expertSystem/TestProgram/id3menu.pl
File Edit Settings Run Debug Help
Create Rules from ID3
Cancel
Enter

```
The host-language is SWI-Prolog version 6.2.3

For HELP on prolog, please type help. or
      on xpce, please type manpce.

1 ?-
% shape.pl compiled 0.00 sec, 7 clauses
creating_tree_model
addToKB
[0.3333333333333333>>[color=red]>>yes,0.3333333333333333>>[color=gree
333333>>[color=blue]>>yes]

Enter KB file name(ex. '1.knb'.): 'shape.knb'.
```

Datafile: shape.pl
MinProb: 0.001

Fig. 4 A snapshot of parameter setting and output of the program id3menu.pl.



```
2 ?- listing(node).
:- dynamic node/2.

node(1, [1, 2]-[3]).
node(2, [1]-[]).
node(3, []-[3]).
node(4, [2]-[]).

true.

3 ?- listing(edge).
:- dynamic edge/3.

edge(0, root=nil, 1).
edge(1, color=red, 2).
edge(1, color=green, 3).
edge(1, color=blue, 4).

true.
```

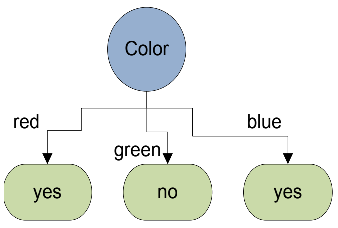
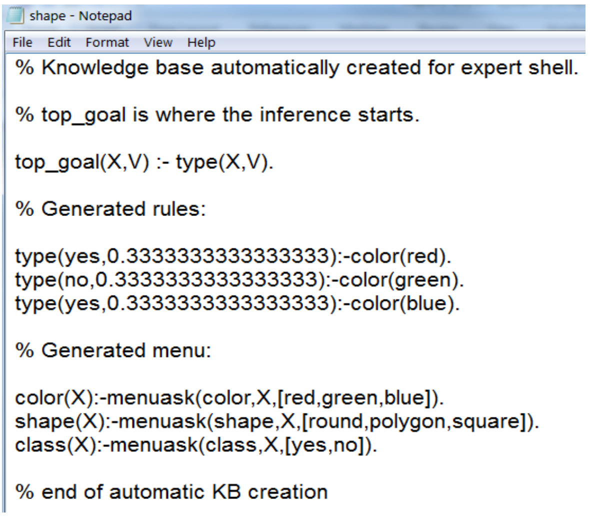


Fig. 5 A tree model in a form of node and edge structures (left) and its interpretation in a graphical form (right).

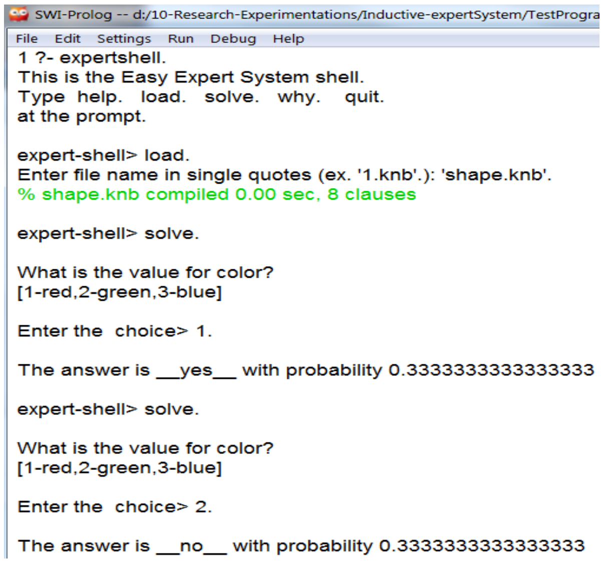


```
% Knowledge base automatically created for expert shell.

% top_goal is where the inference starts.

top_goal(X,V) :- type(X,V).

% Generated rules:

type(yes,0.3333333333333333):-color(red).
type(no,0.3333333333333333):-color(green).
type(yes,0.3333333333333333):-color(blue).

% Generated menu:

color(X):-menuask(color,X,[red,green,blue]).
shape(X):-menuask(shape,X,[round,polygon,square]).
class(X):-menuask(class,X,[yes,no]).

% end of automatic KB creation
```

Fig. 6 A knowledge base ‘shape.knb’ that is automatically generated from a tree model.

3.3 Knowledge Consulting Through the Expert System

To consult a knowledge base, user needs a second program named ‘expertshell.pl’. After running this program (by double-clicking at the file name), the prompt sign ‘1 ?’ will appear on the screen. User can now start commanding the expert system by typing ‘expertshell.’ and press enter. The system will greet with simple advice (as in Fig.7). This expert shell can work with any knowledge base. Therefore, user has to specify the file name of the knowledge base. It is ‘shape.knb’ in this example. Once the knowledge base has been loaded, user may start the consulting process by typing the command ‘solve.’ (Note that every command in Prolog ends with a full-stop.)



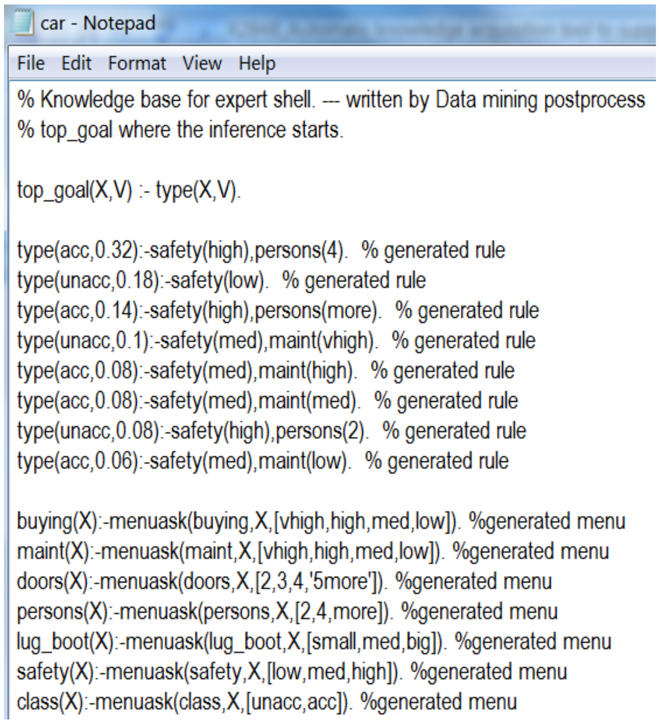
```
SWI-Prolog -- d:/10-Research-Experimentations/Inductive-expertSystem/TestProgra
1 ?- expertshell.
This is the Easy Expert System shell.
Type  help.  load.  solve.  why.   quit.
at the prompt.

expert-shell> load.
Enter file name in single quotes (ex. '1.knb'.): 'shape.knb'.
% shape.knb compiled 0.00 sec, 8 clauses

expert-shell> solve.

What is the value for color?
[1-red,2-green,3-blue]

Enter the  choice> 1.

The answer is __yes__ with probability 0.3333333333333333

expert-shell> solve.

What is the value for color?
[1-red,2-green,3-blue]

Enter the  choice> 2.

The answer is __no__ with probability 0.3333333333333333
```

Fig. 7 An interaction with the expert system shell using a knowledge base 'shape.knb'.

The expert shell starts asking questions as suggested by information stored in the knowledge base. Thus, the order and content of questions can vary according to the knowledge base currently applicable to the expert shell. After the system provides appropriate answer, user may ask for explanation by typing a command 'why.'

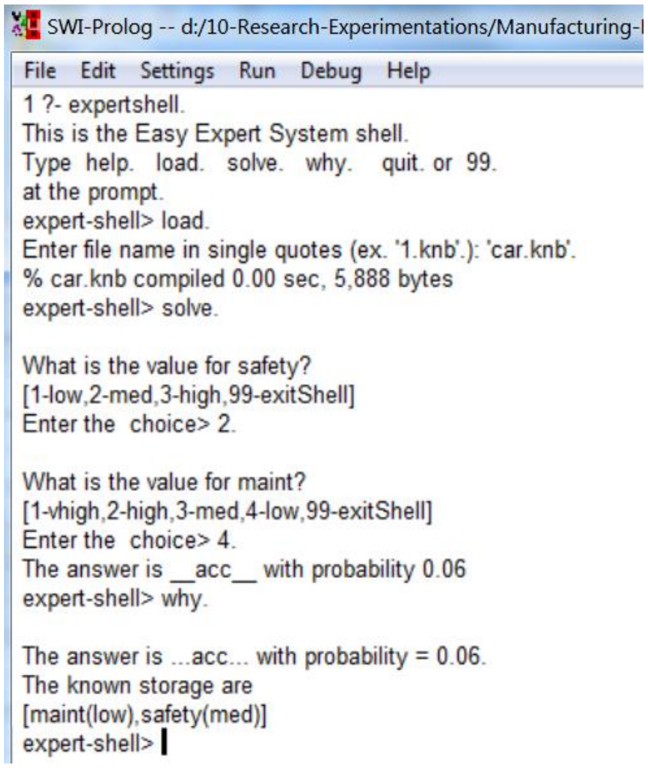
4. Experimentation

The experimentation with real data is to confirm the efficiency of the proposed automatic knowledge base creation method. For the purpose of demonstration, we use a car evaluation data set obtain from the UCI repository (http://archive.ics.uci.edu/ml). In this dataset, each car is to be evaluated as acceptable or unacceptable based on the buying price, price of maintenance, number of doors, capacity in terms of persons to carry, the size of luggage boot, and the estimated safety of the car. The data set has been formatted as Prolog clauses and saved in a file named 'car.pl'. The created knowledge base is illustrated in Fig. 8, and consulting this knowledge base through the expert system shell is shown in Fig.9.



```
car - Notepad
% Knowledge base for expert shell. --- written by Data mining postprocess
% top_goal where the inference starts.

top_goal(X,V) :- type(X,V).

type(acc,0.32):-safety(high),persons(4).  % generated rule
type(unacc,0.18):-safety(low).  % generated rule
type(acc,0.14):-safety(high),persons(more).  % generated rule
type(unacc,0.1):-safety(med),maint(vhigh).  % generated rule
type(acc,0.08):-safety(med),maint(high).  % generated rule
type(acc,0.08):-safety(med),maint(med).  % generated rule
type(unacc,0.08):-safety(high),persons(2).  % generated rule
type(acc,0.06):-safety(med),maint(low).  % generated rule

buying(X):-menuask(buying,X,[vhigh,high,med,low]). %generated menu
maint(X):-menuask(maint,X,[vhigh,high,med,low]). %generated menu
doors(X):-menuask(doors,X,[2,3,4,'5more']). %generated menu
persons(X):-menuask(persons,X,[2,4,more]). %generated menu
lug_boot(X):-menuask(lug_boot,X,[small,med,big]). %generated menu
safety(X):-menuask(safety,X,[low,med,high]). %generated menu
class(X):-menuask(class,X,[unacc,acc]). %generated menu
```

Fig. 8 An automatically created knowledge base 'car.knb'.



```
SWI-Prolog -- d:/10-Research-Experimentations/Manufacturing-I
1 ?- expertshell.
This is the Easy Expert System shell.
Type help. load. solve. why.  quit. or 99.
at the prompt.
expert-shell> load.
Enter file name in single quotes (ex. '1.knb'.): 'car.knb'.
% car.knb compiled 0.00 sec, 5,888 bytes
expert-shell> solve.

What is the value for safety?
[1-low,2-med,3-high,99-exitShell]
Enter the  choice> 2.

What is the value for maint?
[1-vhigh,2-high,3-med,4-low,99-exitShell]
Enter the  choice> 4.
The answer is __acc__ with probability 0.06
expert-shell> why.

The answer is ...acc... with probability = 0.06.
The known storage are
[maint(low),safety(med)]
expert-shell>
```

Fig. 9 Consulting 'car.knb' through the expert system shell.

5. Conclusion

Artificial intelligence, specifically expert systems, has played an important role in solving complex engineering and manufacturing problems. Knowledge base and inference procedures have been employed to solve the problems that require significant human expertise and domain-specific knowledge. The required knowledge has to be elicited by knowledge engineers. It is a labor-intensive task, and thus a bottle neck in building intelligent systems. We propose to apply data mining technique as a major step in a knowledge engine component of the inductive expert system to assist the knowledge elicitation task. The proposed technique is a novel method for automating knowledge acquisition that help supporting intelligent manufacturing systems. Knowledge in our tool can be discovered from the stored data using the decision tree induction algorithm. The learned tree structure is then transformed to a rule set that can be integrated into the knowledge base. The implementation of our knowledge acquisition tool is based on the logic programming scheme that has been proven appropriate for inferring and reasoning answers and recommendations from the existing knowledge base.

Appendix A. Source Code for Automatic Knowledge Base Creation

The source code provided here is for learning a tree model from training data and then transform the model to be a rule set to store in the knowledge base. The given ID3 module is capable of learning model of binary classes such as yes/no, true/false, acceptable/unacceptable. For training data with multiple classes, the module needs some modification. This program should be saved in a single file, named “id3menu.pl”. To run the program, user may double click at the file name in the directory where it has been saved. The knowledge base will be automatically created and stored in the same directory with the file name such as ‘shape.knb’, and this program can now be closed. The created knowledge base will be used later by the expert system shell, which is another Prolog program.

```
/*--------id3menu.pl-------*/
id3menu:-
      new(Dialog,dialog('Create Rules from ID3')),
      send_list(Dialog, append,
           [ new(D1, text_item(datafile,'*.pl')),
            new(Per,text_item(minProb,'0.001')),
            button(cancel, message(Dialog, destroy)),
            button(enter, and(message(@prolog,callId3,
            D1?selection, Per?selection),message(Dialog,destroy) )) ]),
            send(Dialog, open).
callId3(Dfile,Per) :-  term_to_atom(Per1,Per),
                       consult(Dfile), createKB(Per1).
:-id3menu.

% ----------- Create KB rules -----------------------
createKB(Min) :- init(AllAttr,EdgeList), getnode(N),
       create_edge(N,AllAttr,EdgeList), addAllKnowledge,
       selectRule(Min,Res), writeln(Res), nl,
       write('Enter KB file name(ex. ''1.knb''.): '),
       read(F),   tell(F),     writeHeadF,   format('~n%   Generated
rules:~n'),
       maplist(createRule1,Res), nl,
       format('~n% Generated menu:~n'),
       writeTailF,  told,  writeln(endProcess).
writeHeadF :-
     format('%  Knowledge  base  automatically  created  for  expert
shell.'),
     format('~n~n% top_goal is where the inference starts.~n'),
     format('~ntop_goal(X,V) :- type(X,V).~n').
writeTailF :-
    findall(_,(attribute(S,L),
    format('~n~w(X):-menuask(~w,X,~w). ',[S,S,L])),_),
    format('~n~n% end of automatic KB creation').
transform1([X=V],[Res]) :-
       atomic_list_concat([X,'(',V,')'],Res1),
       term_to_atom(Res,Res1),!.
transform1([X=V|T],[Res|T1]) :-
       atomic_list_concat([X,'(',V,')'],Res1),
       term_to_atom(Res,Res1), transform1(T,T1).
createRule1(I) :- I = Z>>X>>Y,
        transform1(X, BodyL),
        format('~ntype(~w,~w):-', [Y,Z]),
        myformat(BodyL) , !.
myformat([X]) :- write(X), write('.'),!.
myformat([H|T]) :- write(H), write(','), myformat(T).
addAllKnowledge :-
       findall([A], pathFromRootToLeaf(A,_), Res),
       retractall(_>>_>>_),  maplist(apply(assert),Res),
       write(addToKB), nl.  % add to knowledge base
selectRule(V,Res) :-
       findall(N>>X>>Class,(X>>Class>>N,N>=V),Res1),
       sort(Res1,Res2), reverse(Res2,Res).
path(A,[H|T],C) :- edge(A,H,B), path(B,T,C).
path(C,[],C) :- !.
pathFromRootToLeaf(V>>Class>>Num, C) :-
       path(1,V,C),   node(C,Value1-Value2),
       (Value1=[] ; Value2=[]),
       (Value1=[] -> length(Value2,Numb) ; length(Value1,Numb)),
        total+Total,    Num is Numb/Total,   hasClass(C1,C2),
         (Value1=[]->Class=C2;Class=C1).

%-------------- ID3 (work only with data with 2 classes) --------------
:- dynamic current_node/1,node/2,edge/3,hasClass/2,type/2.
```

```
init(AllAttr,[root-nil/PB-NB]) :-
        writeln(creating_tree_model), retractall(hasClass(_,_)),
        attribute( class,[ Y1, Y2]),  assert(hasClass(Y1,Y2)),
        retractall(node(_,_)),   retractall(current_node(_)),
        retractall(type(_,_)),   retractall(edge(_,_,_)),
        assert(current_node(0)),   hasClass(C1,C2),
        findall(X,attribute(X,_),AllAttr1),
        delete(AllAttr1,class,AllAttr),
        findall(X2,instance(X2,class=C1,_),PB),
        findall(X3,instance(X3,class=C2,_),NB),
        length(PB,N1), length(NB,N2), N is N1+N2,
        retractall(total+_),  apply(assert,[total+N]).
getnode(X) :- current_node(X), X1 is X+1,
       retractall(current_node(_)),
       assert(current_node(X1)), X1 <4000. % limit at 4000 nodes
create_edge(_,_,[]) :- !.
create_edge(_,[],_) :- !.
create_edge(N,   AllAttr,   EdgeList)   :-     create_nodes(N,   AllAttr,
EdgeList).
create_nodes(N, AllAttr, [H1-H2/PB-NB|T] ) :-
         getnode(N1),
         assert(edge(N,H1=H2,N1)),   assert(node(N1,PB-NB)),
         append(PB, NB, AllInst),
         ( (PB\==[], NB\==[]) -> (cand_node(AllAttr, AllInst, AllSplit),
                                  min_cand(AllSplit, [V, MinAttr, Split]),
                                  delete(AllAttr,MinAttr,Attr2),
                                  create_edge(N1,Attr2,Split))  ;  true ),
         create_nodes(N,AllAttr,T).
create_nodes(_,_,[]) :- !.
create_nodes(_,[],_) :- !.
min_cand([H|T], Min) :- min_cand(T, H, Min).
min_cand([], Min, Min).
min_cand([H|T], Min0, Min) :-  H = [V,_,_], Min0 = [V0,_,_],
            ( V<V0 -> Min1=H ; Min1=Min0),
            min_cand(T, Min1, Min).
cand_node([H|T], CurInstL, [[Val, H, SplitL] | OtherAttr]) :-
            info(H, CurInstL, Val, SplitL),
            cand_node(T, CurInstL, OtherAttr).
cand_node([],_,[]) :- !.
cand_node(_,[],[]).
info(A,CurInstL,R,Split) :-  attribute(A,L),
            maplist(concat3(A,=), L, L1),
            suminfo(L1, CurInstL, R, Split).
concat3(A,B,C,R) :-   atom_concat(A,B,R1), atom_concat(R1,C,R).
suminfo([H|T], CurInstL, R, [Split | ST]) :-
        AllBag = CurInstL,  hasClass(C1,C2),
        term_to_atom(H1,H),
        findall(X1,(instance(X1,_,L1),member(X1,CurInstL),
                    member(H1,L1)), BagGro),
        findall(X2,(instance(X2,class=C1,L2),
                    member(X2,CurInstL), member(H1,L2)), BagPos),
        findall(X3,(instance(X3,class=C2,L3),member(X3,CurInstL),
                    member(H1,L3)), BagNeg),
        (H11=H22) = H1,
        length(AllBag,Nall), length(BagGro,NGro),
        length(BagPos,NPos), length(BagNeg,NNeg),
        Split = H11-H22/BagPos-BagNeg,
        suminfo(T,CurInstL,R1,ST),
        ( NPos is 0 *->L1 = 0; L1 is (log(NPos/NGro)/log(2)) ),
        ( 0 is NNeg *->L2 = 0; L2 is (log(NNeg/NGro)/log(2)) ),
        ( NGro is 0 -> R = 999;
           R is (NGro/Nall)*(-(NPos/NGro)*L1-(NNeg/NGro)*L2)+R1 )  .
suminfo([],_,0,[]).
    % ------------------------------ End of KB Creation Process --------------
```

Appendix B. Expert System Shell in Prolog

```
% -------- expertshell.pl -------------
%  To run this program call 'expertshell.'
%      then call 'load.' and input a file name such as 'file.knb'.
%      Start consulting the expert system with the command 'solve.'
:-dynamic known/1, answer/2.
expertshell :-
        greeting, repeat, nl, write('expert-shell> '), read(X), do(X),
        X == quit,  writeln('>>>>Goodbye, see you later<<<<'), !.
greeting :-
        write('This is the Easy Expert System shell.'), nl,
         native_help.
do(help) :- native_help, !.
do(load) :- load_kb, !.
do(solve) :- solve, !.
do(why) :- why, !.
do(quit).
do(X) :- write(X), write(' is not a legal command.'), nl, fail.
native_help :- write('Type  help.   load.   solve.   why.    quit.'),
        nl, write('at the prompt.'), nl.
load_kb :- write('Enter file name in single quotes (ex. ''1.knb''.): '),
        read(F),  reconsult(F).
solve :-  retractall(known( _) ),retractall(answer(_,_)),
        top_goal(X,V),
        format('The answer is __~w__ with probability ~w',[X,V]),
        assert(answer(X,V)), nl.
solve :-     write('No answer found.'), nl.
menuask(Pred,Value,Menu) :-
         menuask(Pred,Menu),
atomic_list_concat([Pred,'(',Value,')'],X),
         term_to_atom(T,X),  known(T),!.
menuask(Pred,_) :-
         atomic_list_concat([Pred,'(','_',')'],X),
         term_to_atom(T,X),  known(T), !.
menuask(Attribute,Menu):-
        nl, write('What is the value for '), write(Attribute), write('?'),
        nl,  addchoice(Menu,MenuRes), writeln(MenuRes), nl,
        write('Enter the  choice> '), read(C), nl,
        member(C-V,MenuRes),
        atomic_list_concat([Attribute,'(',V,')'],X),
       term_to_atom(T,X),  asserta(known(T)) .

why :- answer(A,V),
       format('~nThe answer is ...~w... with probability =
              ~w.~n',[A,V]),
       findall( X , known(X),Result),
       writeln('The known storage are'), writeln(Result).

addchoice(X,Res) :- length(X,Len),
       numlist(1,Len,NumL), map(NumL,X,Res).

map([],[],[]).
map([H|T], [X|TT], [H-X|T1]) :- map(T, TT, T1).

   % ---------------------- END OF EXPERT SYSTEM SHELL -----------
```

Acknowledgment

This research was supported by the SUT Research and Development Fund, Suranaree University of Technology.

References

[1] F. Alonso, L. Martinez, A. Perez, and J.P. Valente, "Cooperation between expert knowledge and data mining discovered knowledge: Lesson learned," *Expert Systems with Applications*, vol.39, 2012, pp.7524-7535.

[2] A. B. Badiru, *Expert Systems: Applications in Engineering and Manufacturing*, Prentice Hall, 1992.

[3] N. Kerdprasop and K. Kerdprasop, "Higher order programming to mine knowledge for a modern medical expert system," *International Journal of Computer Science Issues*, vol. 8, no. 3, 2011, pp. 64-72.

[4] C. Leon, F. Biscarri, I. Monedero, J. I. Guerrero, J. Biscarri, and R. Millan, "Integrated expert system applied to the analysis of non-technical losses in power utilities," *Expert Systems with Applications*, vol.38, 2011, pp.10274-10285.

[5] R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg, "DENDRAL: A case study of the first expert system for scientific hypothesis formation," *Artificial Intelligence*, vol.61. no.2, 1993, pp.209-261.

[6] R. A. Miller, H. E. Pople, and J. D. Myer, "INTERNIST-1, An experimental computer-based diagnostic consultant for general internal medicine," *New England Journal of Medicine*, vol.307, no.8, 1982, pp.468-476.

[7] A. H. Mohammad and N. A. M. Al Saiyd, "A framework for expert knowledge acquisition," *International Journal of Computer Science and Network Security*, vol.10, no.11, 2010, pp.145-151.

[8] R. A. Perez, L. O. Hall, S. Romaniuk, and J. T. Lilkendey, "Inductive learning for expert systems in manufacturing," *Proceedings of the 25th Hawaii International Conference on System Sciences*, 1992, pp.14-25.

[9] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol.1, no.1, 1986, pp.81-106.

[10] E. H. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, Elsevier, 1976.

[11] T. Witkowski, P. Antczak, and A. Antczak, "Machine learning-based classification in manufacturing system," *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, 2011, pp.580-585.

Nittaya Kerdprasop is an associate professor with the school of computer engineering, Suranaree University of Technology, Thailand. She received her B.S. from Mahidol University, Thailand, in 1985, M.S. in computer science from the Prince of Songkla University, Thailand, in 1991, and Ph.D. in computer science from Nova Southeastern University, U.S.A., in 1999. She is a member of ACM and IEEE Computer Society. Her research of interest includes Knowledge Discovery in Databases, Artificial Intelligence, Logic Programming, Deductive and Active Databases.

Kittisak Kerdprasop is an associate professor and chair of computer engineering school, Suranaree University of Technology, Thailand. He received his bachelor degree in Mathematics from Srinakarinwirot University, Thailand, in 1986, master degree in computer science from the Prince of Songkla University, Thailand, in 1991, and doctoral degree in computer science from Nova Southeastern University, U.S.A., in 1999. His current research includes Data mining, Artificial Intelligence, Functional and Logic Programming, and Computational Statistics.