Building and Querying a Decision Tree Model with Constraint Logic Programming

Nittaya Kerdprasop, Fonthip Koongaew and Kittisak Kerdprasop
Data Engineering Research Unit,
School of Computer Engineering,
Suranaree University of Technology, Thailand
nittaya@sut.ac.th

Abstract
Decision tree induction has gained its popularity as an effective automated method for data classification mainly because of its simple, easy-to-understand, and noise-tolerant characteristics. The induced tree reveals the most informative attributes that can best characterize training data and accurately predict classes of unseen data. Despite its predictive power, the tree structure can be overly expanded or deeply grown when the training data do not show explicit patterns. Such bushy and deep trees are difficult to comprehend and interpret by humans. We thus propose a logic-based method to query over a complicate tree structure to extract only parts of the tree model that are really relevant to users’ interest. The implementation using ECLiPSe constraint language to perform constrained search over a decision tree model is given in this paper. The illustrative examples on medical domains support our hypothesis regarding simplicity of constrained tree-based patterns.

Keywords: Constraint data mining, Pattern induction, Querying, Classification tree, Decision tree induction, Constraint logic programming

1. Introduction
A decision tree is a hierarchical structure comprising of nodes and edges. Each internal node, including the root of a tree, represents a decision choice. All possible decision choices are represented as branches from a node. The terminal decision outcome appears at the leaf node [13, 20, 24]. Machine learning and data mining communities consider the automatic process of building a decision tree from the training data with labeled decision outcomes as a classification problem (if the decision outcomes are continuous values rather than the nominal ones, it is referred to as a regression problem).

Given a training data set with decision attributes and the labeled outcome, the classification process aims at constructing an optimal classifier (or a tree) with minimum classification error. The tree-based classification process is thus composing of the tree-building phase and the pruning phase [12].

Building a decision tree from a set of training data is to partition data with mixing decision classes down the tree until each leaf node contains data instances with pure class. For a large data set with many attributes, constructed tree may contain branches that reflect chance occurrences, instead of the true relationship underlying the data subset. Many tree induction algorithms [5, 16, 18, 19] apply pruning strategies as
subsequent steps following the tree-building phase. A tree pruning operation, either pre-
pruning or post-pruning, involves modifying a tree structure to be more simplified.

The built tree is considered corresponding to a collection of decision rules when
traversing from the root node down to its leaves. A tree, or a set of decision rules, is
normally applied as a classifier to help identifying appropriate decision on the future
event or predicting class of unseen object. A classifier also serves as a generalized
model of the training data set. Due to its simplicity and efficiency, decision tree
induction has been applied in many research and application areas ranging from grid
computing [4], finance [15], engineering [22], health care industry [1, 14], medicine
[11], to bioinformatics [23].

Even with a tree pruning operation, a final tree structure can become a complex
model when applying to the real world data with so many instances and attributes.
General users and decision-makers normally prefer less complex decision trees. Many
researchers solve this problem by simplifying tree structure with the trade off in
classification accuracy [3], or applying some constraints during the tree-building phase
[8, 9]. Our proposed method is different from most existing mechanism in that we deal
with complexity problem after the tree induction phase.

We propose to construct a complete decision tree with the top-down induction
approach [17]. Then we suggest that the users can manipulate the structure to be less
complicate and truly reflect their interest by posing querying on this tree structure.
Querying the tree model also appears in the literature [2, 6] but with quite a different
purpose. Previous work on querying tree aims at extracting meta-data and statistical
information from the model. Our work, on the other hand, focuses on serving users to
extract only parts of the tree model that are of their interest.

We present the method and the detail of our implementation in Section 2. The
prototype of our implementation based on the logic programming paradigm is also
illustrated. Tree induction is normally implemented with SQL language [10, 21]; we
demonstrate in this paper that it can be more effective to implement with constraint
logic programming using ECLiPSe. Section 3 shows querying techniques. Efficiency of
our implementation on medical data [7] is demonstrated in Section 4. Conclusion
appears as the last section of this paper.

2. Building a Decision Tree Model with Logic Programming

We implement the decision tree induction method based on the ID3 algorithm [17]
using logic programming paradigm and run with the ECLiPSe constraint programming
system (http://www.eclipseclp.org). Program and data set are in the same format: that
is, Horn clauses. Example of breast cancer data set [7] is shown in Figure 1. Format of a
data set is \texttt{data([data instances]+[attribute set])}.

Program coding is given in Appendix. To run the program, users may simply call the
predicate “run” as shown in Figure 2. The output of the program is a tree model that has
been displayed as a textual format. Each branch of the tree has also been transformed as
a decision rule.
data(["age="30-39", menopause-premeno, tumor_size="30-34", inv_nodes="0-2", node_caps-no, deg_malig-3, breast-left, breast_quad-left_low, irradiat-no, class-no_recurrence_events"],
...+
["age="40-49", menopause-premeno, tumor_size="20-24", inv_nodes="0-2", node_caps-no, deg_malig-2, breast-right, breast_quad-right_up, irradiat-no, class-no_recurrence_events"],
["class-no_recurrence_events, class-recurrence_events"],
[menopause-lt40, menopause-ge40, menopause-premeno],
[inv_nodes="0-2", inv_nodes="3-5", inv_nodes="6-8", inv_nodes="9-11", inv_nodes="12-14", inv_nodes="15-17", inv_nodes="18-20", inv_nodes="21-23", inv_nodes="24-26", inv_nodes="27-29", inv_nodes="30-32", inv_nodes="33-35", inv_nodes="36-39"],
[node_caps-yes, node_caps-no],
[deg_malig-1, deg_malig-2, deg_malig-3],
[breast-left, breast-right],
[breast_quad-left_up, breast_quad-left_low, breast_quad-right_up, breast_quad-right_low, breast_quad-central],
[irradiat-yes, irradiat-no]
}).

Figure 1. Breast Cancer Data Set in a Horn Clause Format

Figure 2. A Screenshot of Program Running on the Breast Cancer Data Set
3. Querying a Tree Model

Once a decision tree model has been created as shown in Figure 2, user can then query the model with 7 different styles of constraints:

\( \text{findRule}([ ]) \): to display all rules extracted from a decision tree model.

\( \text{findRule}([X]) \): to display only rules that are relevant to the attribute \( X \) (such as \( \text{irradiat} \) for rules that contain the attribute “irradiat” or \( \text{irradiat – yes} \) as a query to show all rules with “irradiat with value yes”); number of attributes is not limited.

\( \text{findRule}([\neg X]) \): to display all rules except the ones with attribute \( X \) (\( \neg \) means “not”).

\( \text{findRule}([X1,X2]) \): to display all rules that satisfy the condition \( X1 \text{ AND } X2 \) (negation \( \neg \) can also be applied to the attributes \( X1, X2 \)).

\( \text{findRuleOr}([X1],[X2]) \): to display all rules satisfied either the attribute \( X1, \) or \( X2 \) (negation \( \neg \) can also be applied to the attributes \( X1, X2 \)).

\( \text{findRuleOr}([\neg X1],[\neg X2]) \): to display all rules extracted from a decision tree model

\( \text{findRuleOr}([X1,\neg X2],[\neg X3,X4]) \): to display all rules that satisfy the compound operations “\( (X1 \text{ AND } (\neg X2)) \text{ OR } ((\neg X3) \text{ AND } X4) \)”.

We show the result of \( \text{findRule}([ ]) \) querying over a tree model induced from the breast cancer data set in Figure 3. Then constraining the result with the query \( \text{findRule} ([\text{class - recurrence_events}]) \). The query result is shown in Figure 4.

```plaintext
if breast_quad - left_low inv_nodes - 15-17 age - 40-49 then class -
if breast_quad - left_low inv_nodes - 15-17 age - 50-59 then class -
if breast_quad - left_low inv_nodes - 24-28 then class - recurrence_ev
if breast_quad - right_up tumor_size - 0-4 then class - no_reccurrence_
if breast_quad - right_up tumor_size - 5-9 then class - no_reccurrence_
if breast_quad - right_up tumor_size - 10-14 then class - no_reccurrence
if breast_quad - right_up tumor_size - 15-19 then class - no_reccurrence
if breast_quad - right_up tumor_size - 20-34 inv_nodes - 0-2 then class
if breast_quad - right_up tumor_size - 20-34 inv_nodes - 3-5 then class
if breast_quad - right_up tumor_size - 25-29 deg_malign - 1 then class
if breast_quad - right_up tumor_size - 25-29 deg_malign - 2 age - 40-
if breast_quad - right_up tumor_size - 25-29 deg_malign - 2 age - 50-
if breast_quad - right_up tumor_size - 25-29 deg_malign - 2 age - 60-
if breast_quad - right_up tumor_size - 25-29 deg_malign - 3 age - 30-
if breast_quad - right_up tumor_size - 25-29 deg_malign - 3 age - 40-
if breast_quad - right_up tumor_size - 25-29 deg_malign - 3 age - 50-
if breast_quad - right_up tumor_size - 25-29 deg_malign - 3 age - 60-
if breast_quad - right_up tumor_size - 30-34 deg_malign - 1 then class
if breast_quad - right_up tumor_size - 30-34 deg_malign - 2 node_caps
if breast_quad - right_up tumor_size - 30-34 deg_malign - 2 node_caps
if breast_quad - right_up tumor_size - 30-34 deg_malign - 2 node_caps
if breast_quad - right_up tumor_size - 30-34 deg_malign - 2 node_caps
```

Figure 3. A Set of Rules Obtained from the Query \( \text{findRule}([ ]) \)
Figure 4. Result of Constrained Search with the Query \texttt{findRule ([class - recurrence\_events])}

4. Experimentation and Results

To test the performance of the proposed method to query over a discrete tree model, we use the standard UCI data repository [7] including the hepatitis data set (155 instances and 15 attributes), breast cancer data set (286 instances, 10 attributes), and thyroid disease data set (2800 instances and 16 attributes). Query result over a hepatitis data set with the query is shown in Figure 5. For each data set we test the system with seven different kinds of queries as summarized in the followings.

Hepatitis data set

\begin{verbatim}
findRule([])
findRule([bilirubin - "0.1-1.0"])
findRule([+bilirubin - "0.1-1.0", class - live])
findRule([+bilirubin - "0.1-1.0", class - live])
findRuleOr([[ascites - yes], [bilirubin - "0.1-1.0", age - "31-40"]]) % Version 1
findRuleOr([[ascites - yes], [bilirubin - "0.1-1.0", age - "31-40"]]) % Version 2
\end{verbatim}

Breast cancer data set

\begin{verbatim}
findRule([])
findRule([breast\_quad - right\_up])
findRule([+breast\_quad - right\_up])
findRule([breast\_quad - right\_up, class - no\_recurrence\_events])
findRule([+breast\_quad - right\_up, +class - no\_recurrence\_events])
findRuleOr([[breast\_quad - left\_low, inv\_nodes - "6-8"] , [breast\_quad - left\_low, inv\_nodes - "6-8", tumor\_size - "15-19"]]) % Version 1
\end{verbatim}
findRuleOr([[breast_quad - left_low, inv_nodes - "6-8"], [breast_quad - left_low, inv_nodes - "6-8", tumor_size - "15-19"]]) % Version 2

Thyroid disease dataset

findRule()
findRule([pregnant - false])
findRule([+pregnant - false])
findRule([query_hyperthyroid - true, query_hyperthyroid - false, sex - male, on_antithyroid_medication - false])
findRule([+query_on_thyroxine - false, +on_antithyroid_medication, +class - sick])
findRuleOr([query_hyperthyroid - true, query_hyperthyroid - false], [+query_hyperthyroid - true], [+class - negative]) % Version 1
findRuleOr([query_hyperthyroid - true, query_hyperthyroid - false], [+query_hyperthyroid - true], [+class - negative]) % Version 2

Figure 5. Running Result of Querying Hepatitis Data Model with the Query findRule([])
As a demonstration of querying the model, we show only one example. For the query “findRuleOr( [ +referral_source- svc, + query_hypothyroid - false, class - sick] ), its result is as follows:

Rule 1: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - female  on_thyroxine - false  query_hyperthyroid - false  psych - false  query_on_thyroxine - false  on_antithyroid_medication - false  pregnant - false  thyroid_surgery - false  lithium - false  goitre - false  hypopituitary - false then class - negative

Rule 2: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - female  on_thyroxine - false  query_hyperthyroid - false  psych - true then class - negative

Rule 3: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - female  on_thyroxine - false  query_hyperthyroid - true then class - negative

Rule 4: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - female  on_thyroxine - true then class - negative

Rule 5: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - male  query_hyperthyroid - false  on_thyroxine - false  query_on_thyroxine - false  on_antithyroid_medication - false  pregnant - false  thyroid_surgery - false  lithium - false  goitre - false  hypopituitary - false  psych - false then class - negative

Rule 6: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - male  query_hyperthyroid - false  on_thyroxine - true then class - negative

Rule 7: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - male  query_hyperthyroid - false  on_antithyroid_medication - false then class - sick

Rule 8: if referral_source - other  sick - false  tumor - false  query_hypothyroid - true  sex - male  query_hyperthyroid - true  on_antithyroid_medication - true then class - negative

Rule 9: if referral_source - other  sick - false  tumor - true then class - negative

Rule 10: if referral_source - other  sick - true  sex - female  query_hyperthyroid - false query_hypothyroid - true then class - negative

Rule 11: if referral_source - other  sick - true  sex - female  query_hypothyroid - true then class - negative

Rule 12: if referral_source - other  sick - true  sex - male then class - negative

Rule 13: if referral_source - stmw then class - negative
Rule 14: if referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - false sex - female psych - false then class - negative

Rule 15: if referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - false sex - female psych - true query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false then class - sick

Rule 16: if referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - false sex - male then class - negative

Rule 17: if referral_source - svhc query_hypothyroid - true on_thyroxine - false sick - true then class - sick

Rule 18: if referral_source - svhc query_hypothyroid - true on_thyroxine - true then class - sick

Rule 19: if referral_source - svhd query_hypothyroid - true sex - female then class - negative

Rule 20: if referral_source - svhd query_hypothyroid - true sex - male then class - sick

Rule 21: if referral_source - svi query_hypothyroid - true sex - female sick - false on_thyroxine - false query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false psych - false then class - negative

Rule 22: if referral_source - svi query_hypothyroid - true sex - female sick - false on_thyroxine - true query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false psych - false then class - sick

Rule 23: if referral_source - svi query_hypothyroid - true sex - female sick - true then class - sick

Rule 24: if referral_source - svi query_hypothyroid - true sex - male sick - false on_thyroxine - false query_on_thyroxine - false on_antithyroid_medication - false pregnant - false thyroid_surgery - false query_hyperthyroid - false lithium - false goitre - false tumor - false hypopituitary - false psych - false then class - negative

Rule 25: if referral_source - svi query_hypothyroid - true sex - male sick - true then class - sick

Performance of running results in terms of rule reduction, that is the simplification of a tree model, can be graphically shown in Figures 6-8.
Figure 6. Performance in Terms of Model Reduction of Hepatitis Data Set

Figure 7. Performance in Terms of Model Reduction of Breast Cancer Data Set

Figure 8. Performance in Terms of Model Reduction of Thyroid Disease Data Set
5. Conclusion

Classification is a data mining task that aims to induce general concept from training data. The induced concept not only explains major characteristics of the underlying data, but also acts as a classification model to predict classes of unseen data. Several learning algorithms have been proposed to induce classification concept, but the most applicable algorithm is decision tree induction. The main reason for its popularity is a simple and understandable form of a tree that has been used to represent the induced concept.

Despite its simplicity and efficiency, it could be a problem when communicating sophisticated concept as a large tree model to general users who are not an expert in decision science or computer technology. Large tree model is difficult to comprehend at a glance. Therefore, simplifying tree structure is necessary for conveying concept model to novice users. Many researchers propose a constraint-based approach during the tree-building phase to make a tree structure more simplified. We, however, consider tree simplification as a post-process of decision tree induction. We propose to grow a full decision tree. Then, apply users’ preferences as a constrained search over a tree model. Only branches of a tree model that correspond to the user-specified constraints are displayed in a simple form of decision rules to the users.

The implemented prototype is expected to ease users in searching for useful knowledge from the tree model. The usability test with users who are practitioners in the field is nevertheless essential to confirm our assumption. In our future work, we also intend to consider other aspects of pushing constraints in the tree induction process.

Acknowledgements

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

Appendix

A source code of decision tree with findRule predicates to query a tree model, implemented with constraint logic programming language ECLiPSe, is given as follows.

```
%% Program Discrete-Tree Induction with findRule queries
:- lib(listut).
:- lib(sd).
:-dynamic rule_me/1.
:-dynamic allrule/1.
append_me([H|T],L,[H|RT]) :- append_me(T,L,RT).
append_me([],L,L).
findRuleOr([H|T]) :- ( findRule(H) -> true ; ! ),
    findRuleOr(T).
findRuleOr([]).
findRule(X) :- allrule(L),
    findRule(X,L).
findRule(_,[]).
findRule(X,[H | T]) :- (findQuery(X,H) -> split_rule(H) ; true),
    findRule(X,T).
findQuery([X|TX],H) :- (findAtt(X,H) -> false ; (findAttLabel(X,H) -> false ; true )),
    findQuery(TX,H).
```
findQuery([X|TX],H) :- (findAtt(X,H) -> true ;  (findAttLabel(X,H) -> true ;  false ) ),
    findQuery(TX,H).
findQuery([],_).
findAtt(X,[X-_|_]) :- findAtt(X,T).
findAtt(_,[]) :- false.
findAttLabel(X,[X|_]) :- findAttLabel(X,T).
findAttLabel(_,[]) :- false.

run :- retractall(rule_me(_)),
    retractall(allrule(_)),
    compile("//C/Users/ASUS/Desktop/id3_ok/data-sickthyroid_OK.txt"),
    data(Data+Attrs),
    main(Data,Attrs,OldAttr),
    retractall(rule_me([_])),
    findall(X,rule_me(X),R),
    assert(allrule(R)).

main([],_,_).
main(_,[],_).
main(Data, Attrs,OldAttr) :- all_info(Data+Attrs, R1),
    \+hasOneClass(R1) -> (maplist(avg_info,R1,Out),
        chooseMin(Out, nil/11,OO),
        O=O/2,
        writeln(Choose ':OO),
        [listut]:delete(Attrs,O,NewAttrs),
        (foreach(X,O),param(Data,NewAttrs,OldAttr) do
            filterData([X],Data,NewData),
            write(' At':X),
            append_me(OldAttr,[X],OAL),
            main(NewData,NewAttrs,OAL)) ) ;
    writeln(''),
    getlast_goal(Data,Att-Ans),
    write(' Ans: '), writeln(Att-Ans),
    append_me(OldAttr,[Att-Ans],NOAL),
    write('rule :'),
    split_rule(NOAL),
    assert(rule_me(NOAL)).

split_rule(NOAL) :- mem_last(NOAL,L,RL),write('if'),
    (foreach(RLL,RL) do write(' '),write(RLL),write(' ')),
    write('then'),write(' '),writeln(L).

%last member in list
%mем_last([1,2,3,5,6],L,RL).
mem_last([H|T],L,[]) :- mem_last(T,L,RL).
main(Data+Attrs,all_info(Data+Attrs, R),
    \+mem_last([1,2,3,5,6],L,RL),
    all_info(Data+P|Attr,R):-maplist(info(Data,P), Attr, R).

%mем(Data+_), info(Data,[p-y,p-n],[o-s,o-c,o-r], R).
%mем([2,3],[4,0],[3,2], info(Data, P, O, O-R) :- maplist(info1(Data,P),O,R).
info1(Data, P, O, R) :- maplist(mcount(Data,O),P,R).

mcount(Data, O, P, Sum) :- foreach(L,Data),fromto(0,I,R,Sun),param(O,P) do
    (member(O,L), member(P,L)->R is I+1; R is I).

%avg_info(o-[2,3],[4,0],[3,2],R).
avg_info(O-LL,O/Info) :- flatten(LL,L), sumlist(L, Sum),
    (foreach(X,LL), fromto(0,I,N,Info),param(Sum) do
        sumlist(X,SumX),
(Sum>0 -> Ratio is SumX/Sum, ! ; Ratio is 0),
logInfo(X,InfoSub),
N is 1+[Ratio*InfoSub)
)).

sumlist([L],[L]),!.
sumlist([],[],[]),!.
sumlist([H1|T1],[H2|T2],[HR|TR]) :- HR is H1+H2,sumlist(T1,T2,TR).

sumlists,[],R,R):-!.
sumlists([H|T],PR,NR) :- sumlist(H,PR,R), sumlists(T,R,NR).

hasOneClass([]):-!.\nhasOneClass([-VL|T]) :- sumlists(VL,[],NR), hasOneClass(T,NR).

hasOneClass([],[]):!.
hasOneClass([-VL|T],PR) :- sumlists(VL,PR,NR), hasOneClass(T,NR).
hasOneClass([],[H|T]) :- (H=0 -> hasOneClass([],T) ; find(T)).

find([]):-!.
find([H|T]) :- (H=0 -> find(T) ; false).

% get the Last data in Fist of list in list
getlast_goal([H],R):- getlast(H,R).
getlast([H],H).
getlast([-|T],R) :- getlast(T,R).

%filterData
filterData(_,[],[]).
filterData(L,[H|Data],[H|R]) :- msubset(L,H), !, filterData(L,Data,R).
filterData(L,[H|Data],R) :- \msubset(L,H), !, filterData(L,Data,R).

%--------
msubset([S1,S2]) :- foreach(X,S1), param(S2) do member(X,S2), !.
allmem([H],[L]) :- member(H,L), !.
allmem([H|L],[]) :- member(H,L), allmem([],L).
logInfo(XL, R) :- sumlist(XL,Sum), Sum==0, R=99, !.
logInfo(XL, R) :- sumlist(XL,Sum),
foreach(X,XL), fromto(0,S,N,R), param(Sum) do
( Ratio is X/Sum,
( Ratio>0 -> iso:[log(Ratio,Log) ; Log is 1], %log(0) is undefined
[iso]:log(2,Base2),
N is S-(Ratio*(Log/Base2) ) )).

chooseMin([],O/Tmp,O/Tmp).
chooseMin([A/H|T],O/Tmp,Min) :- (H<Tmp -> NextMin = A/H ; NextMin = O/Tmp),
chooseMin(T,NextMin,Min).

% ===== End of Program =============

References


Authors

Nittaya Kerdprasop is an associate professor at the School of Computer Engineering, Suranaree University of Technology, Thailand. She received her bachelor degree in Radiation Techniques from Mahidol University, Thailand, in 1985, 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. Her research interest includes Knowledge Discovery in Databases, Artificial Intelligence, and Logic Programming.
**Fonthip Koongaew** is a computer engineer and research assistant. She received her bachelor degree in Computer Engineering from Suranaree University of Technology (SUT), Thailand, in 2010, and master degree in Computer Engineering from SUT in 2012. Her current research includes Constraint Data Mining, Classification Mining, and Logic Programming Languages.

**Kittisak Kerdprasop** is an associate professor and chair of the School of Computer Engineering, 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 Programming Language, and Computational Statistics.