A Robust Tree Induction Method Based on Heuristics and Cluster Analysis

Nittaya Kerdprasop and Kittisak Kerdprasop

Data Engineering Research Unit, School of Computer Engineering,
Suranaree University of Technology, Nakhon Ratchasima 30000 Thailand
nittaya@sut.ac.th, kittisakThailand@gmail.com

Abstract

Data mining is the process of extracting useful and yet unknown information such as patterns or associations hidden in stored data. Among various existing techniques applied to search for interesting patterns, decision tree is one of the most popular tools used for data mining. Most data mining techniques are data-driven, however, data repositories of interest in data mining applications can be very large and noisy. Noise is a random error in data. Noise in a data set can happen in different forms: misclassification or wrong labeled instances, erroneous or distorted attribute values, contradictory or duplicate instances having different labels. All kinds of noise can more or less affect the learning performance. The most serious effect of noise is that it can confuse the learning algorithms to produce complex and distorted results. The long and complex results are due to the attempt to fit every training data instance, including noisy ones, into the concept descriptions. This is a major cause of overfitting problem. Most learning algorithms are designed with the awareness of overfitting problem due to noisy data. Prepruning and postprocessing are two major techniques applied to avoid growing a decision tree too deep down to cover the noisy training data. These techniques are tightly coupled to the tree induction phase. We, on the contrary, design a loosely coupled approach to deal with noisy data. Our noise-handling feature is in a separate phase from the tree induction. Both corrupted and uncorrupted data are clustered and heuristically selected prior to the application of tree induction engine. We observe from our experimental study that tree models produced from our approach are as accurate as the models generated by conventional decision tree induction approach. Moreover, upon highly corrupted data our approach shows a better performance than the conventional approach.

Keywords: *Robust tree induction, Noise tolerance, Noisy data, Heuristics, Cluster analysis*

1. Introduction

Recent advances in digital information storage and data acquisition technologies have made it possible to acquire and store large volumes of data. Extracting useful knowledge from such large volumes of data needs an automatic data mining approach. Decision tree induction is a popular method for mining knowledge from data by means of decision tree building and then representing the end result as a classifier tree. Popularity of this method is due to the fact that mining result in a form of decision tree is interpretability, which is more concern among casual users than a sophisticated method but lacking of understandability [6]. A decision tree is a hierarchical structure with each internal node containing a decision attribute, each node branch corresponding to a distinct attribute value of the decision node, and the class of decision appears at the leaf node [3]. The

goal of building a decision tree is to partition data with mixing classes down the tree until each leaf node contains data instances with pure class.

When a decision tree is built, many branches may be overly expanded due to noise in the training data set. Noisy data contain incorrect attribute values caused by many possible reasons, for instance, faulty data collected from instruments, human errors at data entry, errors in data transmission [1]. Noise in a data set can happen in different forms: (1) misscalssification or wrong labeled instances (called class noise), (2) erroneous or distorted attribute values (called attribute noise), (3) contradictory or duplicate instances having different labels, and (4) missing attribute values. As an example, the first two data instances in Table 1 [11] contain a class noise such that a class label 'N' is wrongly recorded as 'P.' If noise occurs in the training data, it can lower the performance of the learning algorithm [14]. The serious effect of noise is that it can confuse the learning algorithm to produce too specific model because the algorithm tries to classify all records in the training set including noisy ones. This effect is demonstrated in Figure 1.

Table 1. A Training Data Set with class noise in the first and second Instances

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	~~N~~ P
2	sunny	hot	high	true	~~N~~ P
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

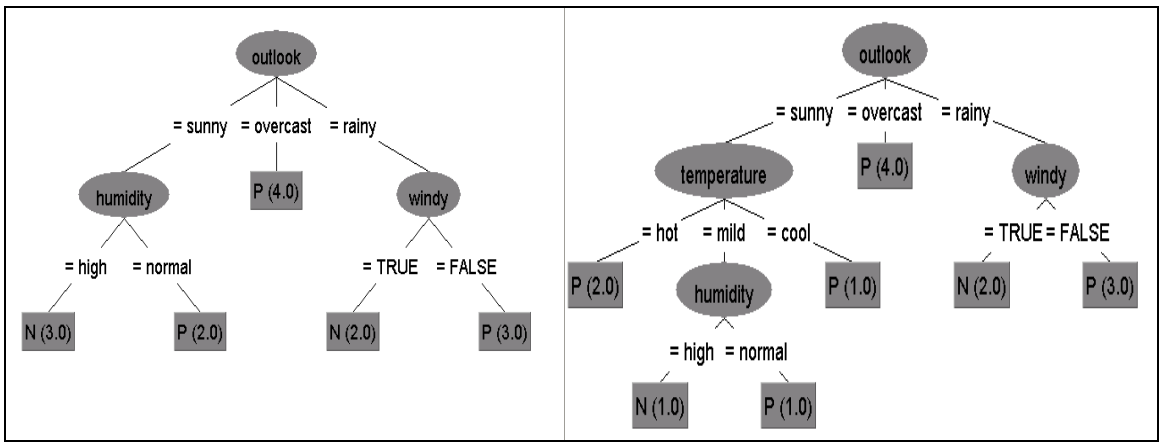


Figure 1. A Decision Tree Built from a Correct Data Set (left) versus a Tree Induced from Noisy Data (right)

Learning from noisy data leads to the problem known as *overfitting* [4, 8, 13]. General solution to this problem is a tree pruning method to remove the least reliable branches, resulting in a simplified tree that can perform faster classification and more accurate prediction about the class of unknown data class labels [4, 8, 10].

Most decision tree learning algorithms are design with the awareness of noisy data. The ID3 algorithm [9] uses the pre-pruning technique to avoid growing a decision tree too deep down to cover the noisy training data. Some algorithms adopt the technique of post-pruning to reduce the complexity of the learning results. Post-pruning techniques include the cost-complexity pruning, reduced error pruning, and pessimistic pruning [7], [11]. Other tree pruning methods also exist in the literature such as the method based on minimum descriptive length principle [12], and dynamic programming based mechanism [2].

A tree pruning operation, either pre-pruning or post-pruning, involves modifying a tree structure during the model building phase. Our proposed method is different from most existing mechanism in that we deal with noisy data prior to the tree induction phase. Its loosely coupled framework is intended to save memory space during the tree building phase and to ease the future extension on dealing with streaming data. We present the research framework and the detail of our methodology in Section 2. The prototype of our implementation based on the logic programming paradigm is illustrated in Section 3. Efficiency of our implementation on noisy data is demonstrated in Section 4. Conclusion and discussion appear as a last section of this paper.

2. Robust Tree Induction Method

Our proposed system has been named *robust-tree induction* to enunciate our intention to design a decision tree induction method with a noise tolerant property. The framework as shown in Figure 2 is composed of the robust-tree component, which is the noise tolerant decision tree induction part, and the testing component responsible for evaluating the accuracy of the decision tree model as well as reporting some statistics such as tree size and running time.

Noise tolerance of our robust-tree induction method can be achieved through the selection of the representative data, instead of learning from each and every training data instance. These selected data are used further in the tree building phase. Training data are first clustered by clustering module to find the mean point of each data group. The data selection module then uses these mean points as a criterion to select the training data representatives. Data around a mean point are considered suitable representatives of a group. This idea is based on the assumption that data along the border line of each group might be the outliers or the noisy instances. It is a set of data representatives that to be used as input of the tree induction phase. Heuristics have to be applied as a threshold in the representative selection step and as a stopping criterion in the tree building phase. The algorithms of a main module as well as the clustering, data selection, and tree induction modules are presented in Figures 3-6, respectively.

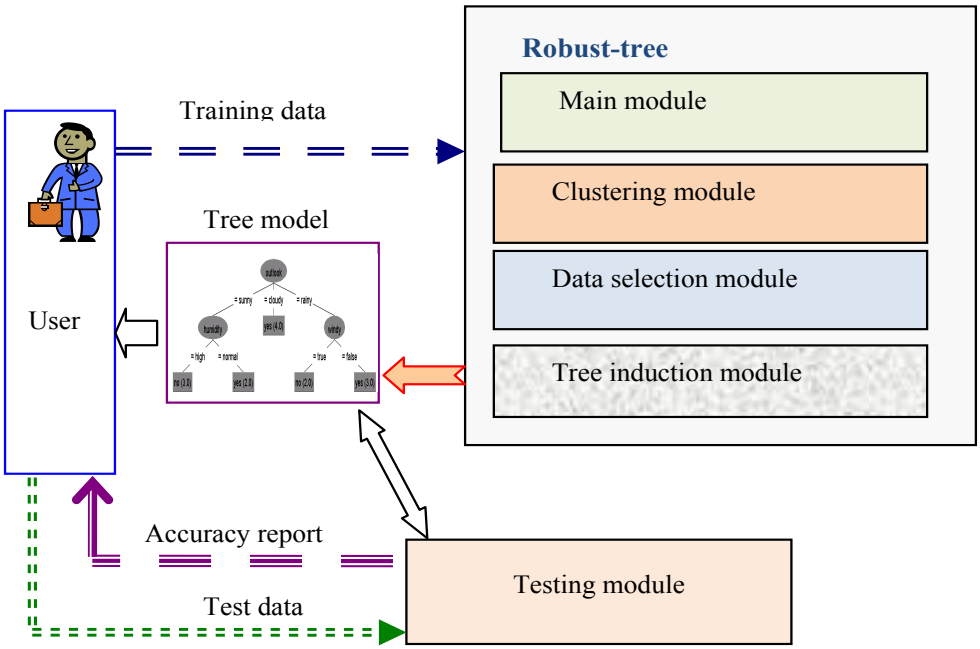


Figure 2. A Framework of the Robust Tree Induction System

Input: Training data set D with class label

Output: A robust tree model M

Steps:

1. Read D and extract class label to check distinctive values K
2. Cluster D to group data into K groups
3. In each group
   - 3.1 Get mean attribute values
   - 3.2. Compute similarity of each member compared to its mean
   - 3.3 Compute average similarity and variance
   - 3.4 Set threshold $T = 2*Variance$
   - 3.5 Select only data with similarity > T
4. Set stopping criteria S for tree building as

$$S = K - log\ [\ (number\ of\ removed\ data + K) / |D|\ ]$$

5. Send selected data and criteria S into tree-induction module
6. Return a tree model

Figure 3. Main Module of the Robust Tree Induction System

Steps:

1. Initialize K means /* Create temporary mean points for all K clusters. */
2. Call find_clusters(K, Instances, Means) /* assign each data to the closest cluster; reference point is the mean of cluster */
3. Call find_means(K, Instances, NewMeans) /* compute new mean of each cluster; this computation is based on current members of each cluster */
4. If Means ≠ NewMeans Then repeat step 2
5. Output mean values and instances in each clusters

Figure 4. Data Clustering Algorithm

Steps:

1. For each data cluster
2. Compute similarity of each member compared to cluster mean
3. Computer average similarity score of a cluster
4. Computer variance on similarity of a cluster
5. *Threshold = 2* variance*
6. Remove member with similarity score < *Threshold*
7. Return K clusters with selected data

Figure 5. Heuristic-based Data Selection Algorithm

Steps:

1. If data set is empty /* Case 1 */
2. Then Assert(node(leaf,[Class/0], ParentNode)
3. Exit /* insert a leaf node in a database, then exit */
4. If number of data instances < MinInstances /* Case 2 */
5. Then Compute distribution of each class
6. Assert(node(leaf, ClassDistribution, ParentNode)
7. If all data instances have the same class label /* Case 3 */
8. Then Assert(node(leaf, ClassDistribution, ParentNode)
9. If data > MinInstances and data have mixing class labels /* Case 4 */
10. Then BuildSubtree
11. If data attributes conflict with the existing attribute values of a tree /* Case 5 */
12. Then stop growing and create a leaf node with mixing class labels
13. Return a decision tree

Figure 6. Tree Induction Algorithm

3. A Logic-based System Implementation

The implementation of a robust tree induction method is based on the logic programming paradigm using SWI-Prolog (www.swi-prolog.org). Data set takes the format of Horn clauses as illustrated in Figure 7.

```
%% Data weather
%
% attribute detail including names and their possible values
%
attribute( outlook,  [sunny, overcast, rainy]).
attribute( temperature, [hot, mild, cool]).
attribute( humidity, [high, normal]).
attribute( windy, [true, false]).
attribute( class, [yes, no]).
%
% data detail
%
instance(1, class=no, [outlook=sunny, temperature=hot, humidity=high, windy=false]).
instance(2, class=no, [outlook=sunny, temperature=hot, humidity=high, windy=true]).
instance(3, class=yes, [outlook=overcast, temperature=hot, humidity=high, windy=false]).
instance(4, class=yes, [outlook=rainy, temperature=mild, humidity=high, windy=false]).
instance(5, class=yes, [outlook=rainy, temperature=cool, humidity=normal, windy=false]).
instance(6, class=no, [outlook=rainy, temperature=cool, humidity=normal, windy=true]).
instance(7, class=yes, [outlook=overcast, temperature=cool, humidity=normal, windy=true]).
instance(8, class=no, [outlook=sunny, temperature=mild, humidity=high, windy=false]).
instance(9, class=yes, [outlook=sunny, temperature=cool, humidity=normal, windy=false]).
instance(10, class=yes, [outlook=rainy, temperature=mild, humidity=normal, windy=false]).
instance(11, class=yes, [outlook=sunny, temperature=mild, humidity=normal, windy=true]).
instance(12, class=yes, [outlook=overcast, temperature=mild, humidity=high, windy=true]).
instance(13, class=yes, [outlook=overcast, temperature=hot, humidity=normal, windy=false]).
instance(14, class=no, [outlook=rainy, temperature=mild, humidity=high, windy=true]).
%
```

Figure 7. A Weather Training Data Set [11] in a Format of Logic-based Programming

Robust-tree induction system provides two level of noise tolerance: 0 and 1. Level 0 corresponds to ordinary ID3 style [9] without additional noise handling mechanism. Level 1 is a robust-tree induction with a heuristic-based mechanism to deal with noisy data. The main module of the system is the module 'rt' which can be displayed as a logic program as follows:

```
%% Main module:  rt
%% ==========
  rt :-
    writeln('Robust tree induction for data classification:'), nl,
    writeln(' There are two level of robustness'),
    writeln('  0 =   = simply ID3 style without noise handling function'),
    writeln('  1 = grouping data then select representatives to build tree'),  nl,
    write('     Please specify level of robustness (and end command with a period): '),
```

```
read(L),
 write('    Training-data file name (e.g. data-sample.) ==> '),
 read(D),              % get data file name as typed by user
 consult(D),           % then open and compile that file; data is also a prolog program
 get_time(StartTime),
                       % retractall:  clear all nodes and node-ID counter in the DB
                       %        node and counter are two global values of this program
 retractall(node(_, _, _)),
 retractall(counter(_)),
                       % findall:  make list Attr of all attribute names
                       %            except attribute class
findall(A, (attribute(A, _), A \= class), Attr),
rtree(L, Attr),        % then call  robust tree building module
get_time(FinishTime),
Time is FinishTime-StartTime,
 nl,write('ROBUST-TREE:: robust level '),write(L),write(',  '),
 write('Model building time = '), write(Time),writeln('  sec.').
```

The main module interacts with user to input the file name and to specify the tolerant level of robust tree. The predicate that deals with robust tree induction is 'rtree.' The Prolog coding of robust tree induction at noise tolerance levels 0 and 1 are presented as the predicates *rtree(0, Attr)* and *rtree(1, Attr)*, respectively. The Prolog codes are as follows:

```
% ------------------------------------
%  start traditional tree-induction with ID3 algorithm

  rtree(0, Attr) :-  !,
                  % make a list Ins = [1,2,...,n] of all instance ID
     findall(N, instance(N, _, _), Ins),
        % create decision tree, start with the root node
        % set MinInstance in leaf nodes = 1
        % then show model as decision tree once finish building phase
     induce_tree(root, Ins, Attr, 1),
     print_tree_model.

%-------------------------------------------
% start clustering before induce a robust tree

  rtree(1, Attr) :-  !,
     attribute(class, ClassList),
     length(ClassList, K),            % K is for specifying number of clusters
     findall(N, instance(N,_,_),Ins),
     clustering(Ins, K, Clusters, Means),     % grouping instances
     select_DataSample(Clusters,K,Means,[],Sample),
               % then select Sample
     removed_Data(Sample, Ins, Removed),
     length(Removed, R),
     length(Ins, I),
     MinInstance is K-log((R+K)/I),            % a heuristic to prune tree
     induce_tree(root,Sample,Attr,MinInstance),
      print_tree_model.
```

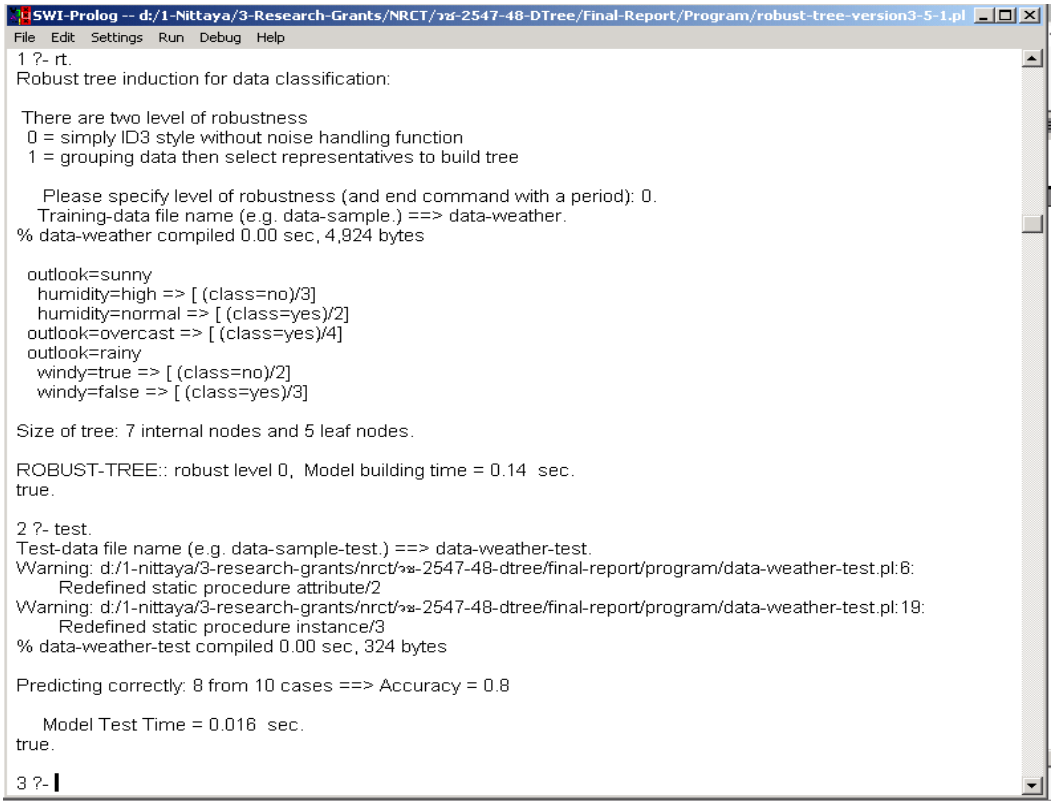


Figure 8. User Interface of the Logic-based Robust-tree Induction System Running with the Noise Tolerance at Level 0

The robust tree induction system is invoked by the predicate 'rt' as shown in Figure 8. The built model can be tested for its predictive accuracy by calling the predicate 'test.' Running result in Figure 8 is the robust tree model with zero noise tolerance. Therefore, it generates the same model as the one we got from the conventional decision tree induction algorithm. At the noise tolerance level 1, the system can induce a more compact tree model as shown in Figure 9. Model evaluation results using the training data are shown in Figure 10.

outlook=sunny **humidity=high** **windy=true => [(class=no)/1]** **windy=false** **temperature=hot => [(class=yes)/1]** **temperature=mild => [(class=no)/1]** **temperature=cool => [(class=yes)/0]** **humidity=normal => [(class=yes)/2]** **outlook=overcast => [(class=yes)/4]** **outlook=rainy** **windy=true => [(class=no)/2]** **windy=false => [(class=yes)/3]** Size of tree: 12 internal nodes and 8 leaf nodes. ROBUST-TREE:: robust level 0, Model building time = 0.125 sec.	**outlook=sunny** **humidity=high => [(class=no)/2, (class=yes)/1]** **humidity=normal => [(class=yes)/2]** **outlook=overcast => [(class=yes)/4]** **outlook=rainy** **windy=true => [(class=no)/2]** **windy=false => [(class=yes)/3]** Size of tree: 7 internal nodes and 5 leaf nodes. ROBUST-TREE:: robust level 1, Model building time = 0.0940001 sec.

Figure 9. Conventional Decision Tree Model (left) versus the Robust Tree Model (right)

?- test. Test-data file name (e.g. data-sample-test.) ==> data-weather. % data-weather compiled 0.00 sec, -512 bytes **Predicting correctly: 13 from 14 cases ==> Accuracy = 0.928571** Model Test Time = 0.016 sec.	?- test. Test-data file name (e.g. data-sample-test.) ==> data-weather. % data-weather compiled 0.00 sec, 0 bytes **Predicting correctly: 14 from 14 cases ==> Accuracy = 1** Model Test Time = 0.0 sec.

Figure 10. Model Evaluation Results of Conventional Decision Tree Model (left) versus the Robust Tree Model (right)

The robust tree with noise tolerance level 1 can induce a compact decision tree model by applying the cluster analysis technique to select data representatives and also using heuristics to stop the tree growing phase. The cluster analysis module and the test module as a Prolog program are given as follows:

```
%%  Module Clustering
%%  ==============
clustering(Ins, K, Clusters, Means) :-
    length(Ins, N),
    initialized_means(N, K, [], MeanPoints),
                          % e.g. MeanPoints = [2/1, 3/2]
                          % get attributes of initial MeansPoints
    findall(MeanAttr/Cluster, (member(P/Cluster, MeanPoints),
                                instance(P, _, MeanAttr)),
            MeansAttrList),
                          % e.g. [(size=small,color=red,shape=circle)/1,
                          %       (size=large,color=blue,shape=circle)/2]
    assign_clusters(MeansAttrList, Ins, K, Clusters,Means).
%
%
assign_clusters(MeansAttr, Ins, K, Clusters,Means) :-
    find_clusters(MeansAttr, Ins, [], InsClusterList),
    find_means(InsClusterList, K, [], TempMeans),
    getRepresentatives(TempMeans,Ins,[],RepList),
    getMeans(RepList,[],NewMeans),
    find_clusters(NewMeans, Ins, [], NewInsClusterList),
    entropy(InsClusterList, K, PreEntropy),
    entropy(NewInsClusterList, K, PostEntropy),
    average(PreEntropy, PreEn),
    average(PostEntropy, PostEn),
    (PostEn >= PreEn, Clusters = InsClusterList, Means = NewMeans, ! ;
       assign_clusters(NewMeans, Ins, K, Clusters,Means),!).

%%
initialized_means(_, 0, Means, Means) :- !.
```

```
initialized_means(N, K, Means, NewMeans) :-
      MeanIns is random(N-1)+1,
      NewK is K-1,
      initialized_means(N, NewK, [MeanIns/K|Means], NewMeans).
%%
find_clusters(_, [], List, List) :- !.
find_clusters(MeanAttrList, [Ins|Rest], CurrentList, NewList) :-
       findall(Cluster/Score, (instance(Ins,_, InsAtt),
                                  member(MAtt/Cluster, MeanAttrList),
                                  similarity(MAtt, InsAtt, 0, Score) ),
                ClusterScoreList),
       maximum(ClusterScoreList, Cluster/_),
       find_clusters(MeanAttrList, Rest, [Ins/Cluster|CurrentList], NewList).
similarity([],[],S,S) :- !.
similarity([A | RestA1], [A | RestA2], Score, NewS) :-
                NewScore is Score + 1, !,
                similarity(RestA1, RestA2, NewScore, NewS).
similarity([A1| RestA1], [A2|RestA2], Score, NewS) :-
                 A1 \= A2,
                 similarity(RestA1, RestA2, Score, NewS).
%%
minimum([ClusterScore], ClusterScore) :- !.
minimum([C/S | Rest], Cluster/Score) :-
          minimum(Rest, Clus/Sc),
          ( Sc > S, Cluster/Score = C/S ;
                    Cluster/Score = Clus/Sc), !.
%%
find_means(_, 0, List, List) :- !.
find_means(InsClusterList, K, CurrentList, NewList) :-
         findall(Ins, member(Ins/K, InsClusterList), InsList),
         findall(Name=Vlist, (attribute(Name,Values),
                                 Name \= class,
                                 findall(V/0, member(V,Values), Vlist)),
                 AttValueList),
         common_attributes(InsList, AttValueList, AttrList),
         NewK is K - 1,
         find_means(InsClusterList, NewK, [AttrList/K | CurrentList], NewList).

%%
common_attributes([], AttValueList, AttList) :- !,
          findall(A=V, (member(A=VList, AttValueList),
                       maximum(VList, V/_)  ),   AttList).

common_attributes([Ins|Rest], AttValueList, AttList) :-
         instance(Ins,_, AttValue),
         count_value(AttValue, AttValueList, NewAttValueList),
         common_attributes(Rest, NewAttValueList, AttList).

%%
count_value([], AVList, AVList) :- !.
count_value([A=V|Rest], AttValueList, NewAttValueList) :-
         member(A=VList, AttValueList),
```

```
        delete(AttValueList, A=VList, TempAttValueList),
        member(V/Count, VList),
        delete(VList, V/Count, TempVList),
        NewCount is Count + 1,
        append([V/NewCount], TempVList, NewVList),
        append([A=NewVList], TempAttValueList, NewAVList),
        count_value(Rest, NewAVList, NewAttValueList).

%%
entropy(_, 0, []) :- !.
entropy(InsCluster, K, Entropy) :-      K>0,
        findall(Ins, member(Ins/K, InsCluster), InsList),
        length(InsList, InsLen),
        (InsLen >0,
             compute_info(InsList, InsLen, Info),
             Entropy = [K/Info | RestEntropy];
        Entropy = [K/1 | RestEntropy]),
        NewK is K-1,
        entropy(InsCluster, NewK, RestEntropy).
sum_list([], 0) :- !.
sum_list([H|T], Value) :-
          sum_list(T, NewValue),
          Value is H + NewValue.
%%
getRepresentatives([],_, List, List) :- !.
getRepresentatives([Mean/Cluster | Rest], InsList, Current, NewList) :-
        findall(Ins/Score, (member(Ins, InsList),
                            instance(Ins,_,InsAtt),
                            similarity(InsAtt, Mean, 0, Score)),   InsScoreList),
        maximum(InsScoreList, Instance/_),
        delete(InsList, Instance, NewIns),
        getRepresentatives(Rest,NewIns, [Instance/Cluster | Current], NewList).
%%
getMeans([], List, List) :- !.
getMeans([Ins/Cluster | Rest], Current, NewMeans) :-
        instance(Ins,_, InsAtt),
        getMeans(Rest, [InsAtt/Cluster | Current], NewMeans).

removed_Data(DataSample, InstList ,RemovedData  ) :-
          findall( D, (member(D, InstList),
                       not(member(D, DataSample))),
                   RemovedData).
%%
select_DataSample(_, 0, _, DataSample, DataSample) :- !.
select_DataSample(Clusters, K, Means, TempData, DataSample) :-
          findall(Ins, member(Ins/K, Clusters), InsKList),
          length(InsKList, Len), Len > 0,
          findall(I/Score, (member(I, InsKList),
                            instance(I, _, InsAtt),
                            member(MAtt/K, Means),
                            similarity(InsAtt, MAtt, 0,Score)),   IScoreList),
        average(IScoreList, Average),
```

```
        variance(IScoreList, Average,Variance),
        Threshold is (2 * Variance),        % a heuristic for stopping tree induction
        findall(Inst, (member(Inst/Sc, IScoreList),
                        Sc >= Threshold), InstList),
        append(InstList, TempData, NewData),
        NewK is K-1,
        select_DataSample(Clusters, NewK, Means, NewData, DataSample).
%%
 average(ValueList, E) :-
        findall(S, member(_/S, ValueList), SList),
        sum_list(SList, SValue),
        length(SList, Len),
        (Len=0, E = 0; E is SValue / Len).
%%
variance(ValueList, Avg, Var) :-
        findall(Diff, (member(_/S, ValueList),
                        Diff is abs(S-Avg)),
                DiffList),
        sum_list(DiffList, DValue),
        length(DiffList, DLen),
        D is DLen-1,
        (D=0, Var = 0; Var is DValue / D).
% ===== End Clustering ==============


    %==== Test TREE =========
    test :-
        write('Test-data file name (e.g. data-sample-test.) ==> '),
        read(D),
        consult(D),
        get_time(Start),          % get all instance ID of test data
        findall(TestIns, instance(TestIns, _, _), TestInsList),
        length(TestInsList, NumTestCase),
                        % send all test cases to test_accuracy module
                        %      with initial correct case = 0
        test_accuracy(TestInsList, 0, Totalcorrect), !,
        Accuracy is Totalcorrect / NumTestCase,
        nl,write('Predicting correctly: '), write(Totalcorrect),
        write(' from '), write(NumTestCase), write(' cases ==> '),
        write('Accuracy = '), writeln(Accuracy),
        get_time(Finish),
        Time is Finish- Start,
         nl, tab(5), write('Model Test Time = '), write(Time), writeln('  sec.').

     %% Module test_accuracy
     %%  get all test cases, and
     %%    start evaluating correctness of prediction one case at a time,
     %%    stop when the lest of test cases is empty,
     %%    then report the total number of cases predicted correctly
     test_accuracy([], C, C) :- !.
    test_accuracy([Case | Rest], Correct, NextCorrect) :-
          instance(Case, Trueclass, AttList), % get current test case
```

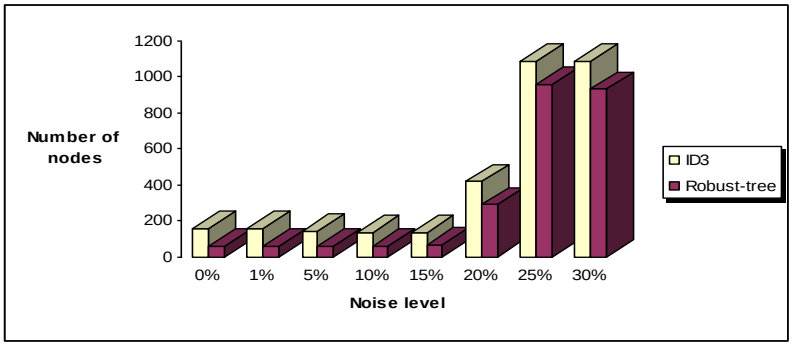
```
                                % search tree for predicted class start from root node
        search_decision(root, AttList, Prediction),
                                % compare Trueclass and PredictedClass
                                % and count correct prediction
        evaluate(Case, Trueclass, Prediction, Correct, NewCorrect),
                                % recursively do the same for other cases
        test_accuracy(Rest, NewCorrect, NextCorrect).

search_decision(StartNode, _, Prediction) :-
        node(leaf, Prediction, StartNode), !.
                % return Prediction once leaf node has been found
search_decision(StartNode, AttList, Prediction) :-
        node(NextNode, TestAtt, StartNode),
        member(TestAtt, AttList), !,
        search_decision(NextNode, AttList, Prediction).

evaluate(_, Trueclass, Prediction, Correct, NewCorrect) :-
                        % Prediction might be a mixture such as
                        %          [(class=positive)/2, (class=negative)/1]
                        % thus, PredictedClass should be the majority class
        maximum(Prediction, PredictedClass/_),
       (Trueclass == PredictedClass,  NewCorrect is Correct +1;
        NewCorrect = Correct).
  %% ======== END Test-Tree======================
```

4. Experimental Results

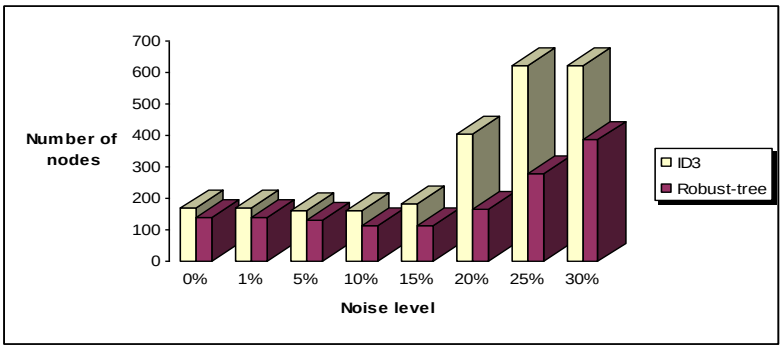
On a series of experimentation, we compare sizes of the tree models as well as predicting accuracy of the robust-tree models with noise tolerance level 0 and 1. To test the accuracy and the noise tolerant efficiency of the proposed robust-tree induction system, we use the standard UCI data repository [5] including the monk, audiology, breast cancer, and vote data sets. Characteristics of these data sets are summarized in Table 2. Each data set is composed of a separate subset of training and test data. We prepare each training data set to contain eight levels of noise, that is, 0%, 1%, 5%, 10%, 15%, 20%, 25%, and 30%. The comparison results of conventional decision tree induction algorithm (ID3 [9]) and the robust-tree induction algorithm in terms of model size and predicting accuracy are shown in Figures 11 and 12, respectively.

Table 2. Summary of Data Sets used in the Robust Tree Experimentation

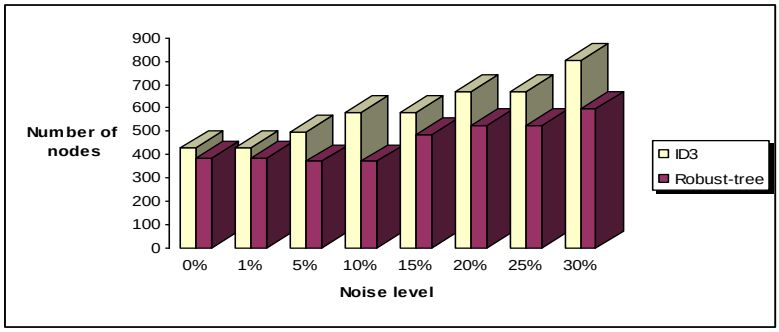
Data set name	# Instances in training data	# Instances in test data	# Predicting attributes	# Classes in goal attribute
Monk	124	432	6	2
Audiology	150	76	69	24
Breast cancer	191	95	9	2
Vote	300	135	16	2



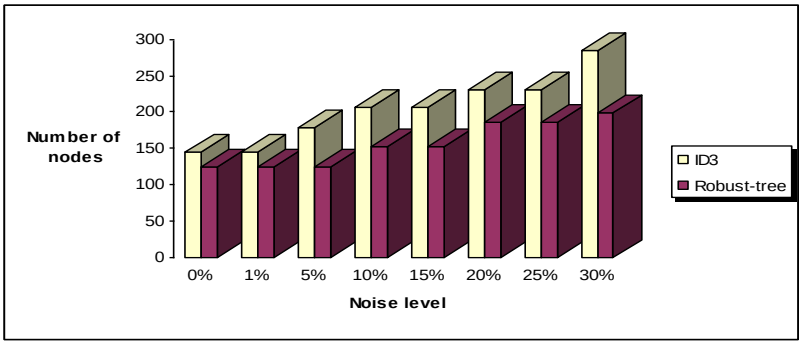
(a) Monk data set



(b) Audiology data set

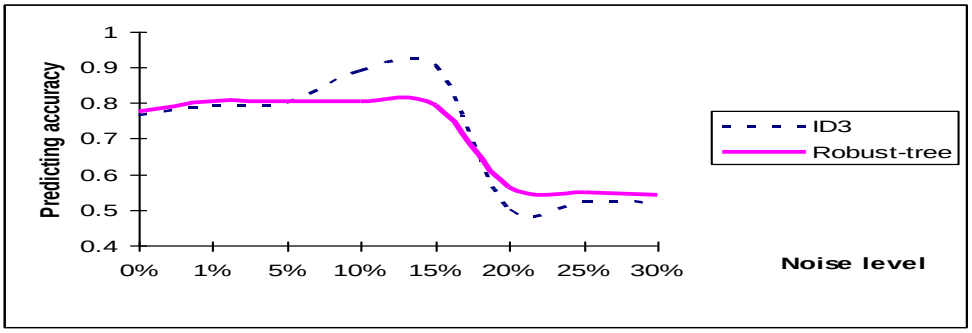


(c) Breast cancer data set

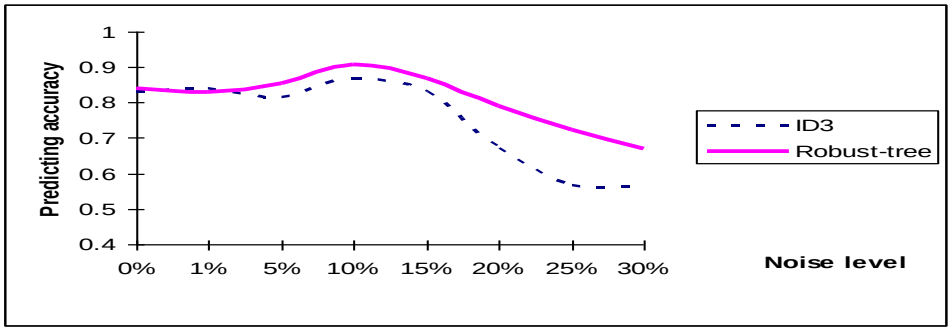


(d) Vote data set

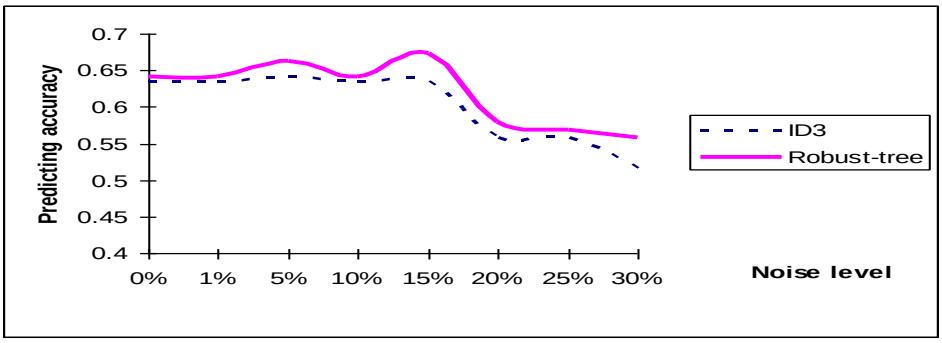
Figure 11. Model Size Comparison of the Robust Tree Induction Against Conventional Decision Tree Induction



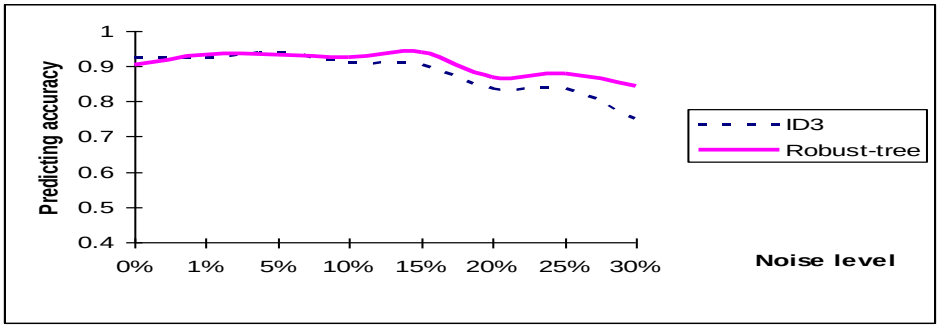
(a) Monk data set



(b) Audiology data set



(c) Breast cancer data set



(d) Vote data set

Figure 12. Predicting Accuracy of the Robust Tree Induction Models Against Conventional Tree Induction Models

It can be seen that robust-tree induction can produce a smaller tree model than conventional decision tree induction algorithm. The predicting accuracy of robust-tree model is higher than conventional decision tree model on most data sets, except the monk data set at the noise level 7-18% that the robust-tree model cannot outperform the conventional decision tree model.

On running time comparison, we record the time to build model in addition to the time for model testing (in seconds) of conventional decision tree induction against the robust-tree induction. Comparison results of the four data sets are reported in Table 3.

Table 3. Model Building and Testing Time (in seconds) of the Four Data Sets

	Noise level	Conventional decision tree induction	Robust-tree induction
Monk	0%	0.359+0.093=0.488	0.312+0.093=0.405
Data set	1%	0.344+0.141=0.485	0.344+0.094=0.438
	5%	0.343+0.110=0.453	0.297+0.063=0.36
	10%	0.360+0.094=0.454	0.375+0.094=0.469
	15%	0.313+0.110=0.423	0.375+0.078=0.453
	20%	0.766+0.250=1.016	0.797+0.25=1.047
	25%	2.609+0.954=3.563	2.593+0.968=3.561
	30%	2.594+0.859=3.453	2.656+0.906=3.562
Audiology	0%	0.414+0.069=0.483	0.398+0.041=0.439
data set	1%	0.409+0.092=0.501	0.387+0.044=0.431
	5%	0.383+0.077=0.460	0.379+0.038=0.417
	10%	0.397+0.095=0.492	0.361+0.046=0.407
	15%	0.471+0.133=0.604	0.375+0.033=0.408
	20%	1.577+0.181=1.758	0.884+0.059=0.943
	25%	1.965+0.911=2.876	1.726+0.656=2.382
	30%	2.018+0.965=2.983	1.998+0.701=2.699
Breast	0%	0.210+0.011=0.221	0.204+0.008=0.212
cancer	1%	0.197+0.015=0.409	0.213+0.015=0.228
data set	5%	0.274+0.048=0.322	0.186+0.011=0.197
	10%	0.211+0.069=0.280	0.197+0.020=0.217
	15%	0.298+0.053=0.351	0.214+0.029=0.243
	20%	0.323+0.079=0.402	0.231+0.037=0.268
	25%	0.465+0.93=1.395	0.240+0.056=0.296
	30%	0.512+0.104=0.616	0.443+0.077=0.520
Vote	0%	0.414+0.069=0.483	0.398+0.041=0.439
data set	1%	0.409+0.092=0.501	0.387+0.044=0.431
	5%	0.383+0.077=0.460	0.379+0.038=0.417
	10%	0.397+0.095=0.492	0.361+0.046=0.407
	15%	0.471+0.133=0.604	0.375+0.033=0.408
	20%	1.577+0.181=1.758	0.884+0.059=0.943
	25%	1.965+0.911=2.876	1.726+0.656=2.382
	30%	2.018+0.965=2.983	1.998+0.701=2.699

5. Conclusion

Noisy data can cause serious problem to many learning algorithms in terms of distorted results and the decrease in predicting performance of the learning results. In this paper, we propose a methodology to deal with noise in a decision tree induction algorithm. Our intuitive idea is to select only potential representatives, rather than applying the whole training data that some values are corrupted, to the tree induction algorithm.

Data selection process starts with clustering to form groups of similar data items in order to obtain the mean point of each data group. For each data group, the selection heuristic $T = 2 * Variance\_of\_cluster\_similarity$ will be used as a threshold to select only data around mean point within this T distance. Data that lie far away from the mean point are considered prone to noise and outliers; we thus remove them.

The removed data still play their role as one factor of a tree building stopping criterion, which can be formulated as $S = K - log[(number\ of\ removed\ data\ instances + K) / D]$, where K is the number of clusters, which has been set to be equal to the number of class labels, and D is the number of training data.

From experimental results, it turns out that our heuristic-based decision tree induction method is robust to data set with a high level of noise. It also produces a compact tree model. With such promising results, we thus plan to improve our methodology to be incremental such that it can learn model from steaming data.

Acknowledgements

This work has been supported by grants from the National Research Council of Thailand (NRCT) and Suranaree University of Technology via the funding of Data Engineering Research Unit.

References

[1] Angluin D and Laird P, "Learning from noisy examples", Machine Learning, vol. 2, **(1988)**, pp. 343-370.
[2] Bohanec M and Bratko I, "Trading accuracy for simplicity in decision trees", Machine Learning, vol. 15, **(1994)**, pp. 223-250.
[3] Breiman L, Freidman J, Olshen R and Stone C, "Classification and Regression Trees", Wadsworth **(1984)**.
[4] Esposito F, Malerba D and Semeraro G, "A comparative analysis of methods for pruning decision trees", IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 19, no. 5, **(1997)**, pp. 476-491.
[5] Frank A and Asuncion A, "UCI Machine Learning Repository", [http://archive.ics.uci.edu/ml]. Irvine, University of California, School of Information and Computer Science, **(2010)**.
[6] Han J and Kamber M, "Data Mining: Concepts and Techniques", 2nd ed. Morgan Kaufmann **(2006).**
[7] Kim H and Koehler G, "An investigation on the conditions of pruning an induced decision tree", European Journal of Operational Research, vol. 77, no. 1, **(1994)**, pp. 82.
[8] Mingers J, "An empirical comparison of pruning methods for decision tree induction", Machine Learning, vol. 4, no. 2, **(1989)**, pp. 227-243.
[9] Quinlan J, "Induction of decision tree", Machine Learning, vol. 1, **(1986)**, pp. 81-106.
[10] Quinlan J, "Simplifying decision tree", In: Gaines, B., Boose, J. (eds.) Knowledge Acquisition for Knowledge Based Systems, vol. 1, Academic Press, **(1989)**.
[11] Quinlan J, "C4.5: Programs for Machine Learning", Morgan Kaufmann **(1992)**.
[12] Quinlan J and Rivest R, "Inferring decision trees using the minimum description length principle", Information and Computation, vol. 80, no. 3, **(1989)**, pp. 227-248.
[13] Schaffer C, "Overfitting avoidance bias", Machine Learning, vol. 10, **(1993)**, pp. 153-178.
[14] Talmon J and McNair P, "The effect of noise and biases on the performance of machine learning algorithms", Int. J. Bio-Medical Computing, vol. 31, no. 1, **(1992)**, pp. 45-57.

Authors

Nittaya Kerdprasop is an associate professor at 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, USA, 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 at 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, USA., in 1999. His current research includes Data mining, Artificial Intelligence, Functional and Logic Programming Languages, Computational Statistics.