Knowledge Discovery through SysFor - a Systematically Developed Forest of Multiple Decision Trees

Md Zahidul Islam[1] **Helen Giggins**[2]

[1] Center for Research in Complex Systems (CRiCS),
School of Computing and Mathematics,
Charles Sturt University,
Boorooma Street, NSW 2678, Australia.
Email: zislam@csu.edu.au

[2] School of Architecture and Built Environment,
Newcastle University,
Callaghan, NSW 2308, Australia.
Email: helen.giggins@newcastle.edu.au

Abstract

Decision tree based classification algorithms like C4.5 and Explore build a single tree from a data set. The two main purposes of building a decision tree are to extract various patterns/logic-rules existing in a data set, and to predict the class attribute value of an unlabeled record. Sometimes a set of decision trees, rather than just a single tree, is also generated from a data set. A set of multiple trees, when used wisely, typically have better prediction accuracy on unlabeled records. Existing multiple tree techniques are catered for high dimensional data sets and therefore unable to build many trees from low dimensional data sets. In this paper we present a novel technique called *SysFor* that can build many trees even from a low dimensional data set. Another strength of the technique is that instead of building multiple trees using any attribute (good or bad) it uses only those attributes that have high classification capabilities. We also present two novel voting techniques in order to predict the class value of an unlabeled record through the collective use of multiple trees. Experimental results demonstrate that *SysFor* is suitable for multiple pattern extraction and knowledge discovery from both low dimensional and high dimensional data sets by building a number of good quality decision trees. Moreover, it also has prediction accuracy higher than the accuracy of several existing techniques that have previously been shown as having high performance.

Keywords: Data Mining, Classification Algorithm, Multiple Decision Tree, Prediction Accuracy.

1 Introduction

Huge amount of data are being collected these days in almost every sector of life. Collected data are generally processed and stored as data sets so that various data mining techniques can be applied on them. In this study we consider a data set as a two dimensional table where rows are records and columns are the attributes. We also consider that a data set can have two types of attributes; numerical (such as Price and Temperature) and categorical (such as Employer's Name, and Country of Origin). Numerical attribute values have a natural ordering among them whereas categorical values do not exhibit any natural ordering.

Various data mining techniques are applied to these data sets to extract hidden information. For example, a decision tree algorithm is applied on a data set to discover the logic rules (patterns) that represent a relationship between various classifier (non-class) attributes and the class attribute [18, 19, 10]. A class attribute, which is often also known as the label of a record, is a categorical attribute such as "Diagnosis" in a data set having patient records. Each row/record of such a data set stores values of various classifier attributes for instance Age, Blood Pressure and Blood Sugar Level of a patient, and the class attribute (Diagnosis) of the record. Note that in this study we do not consider multi-label scenario where each record contains more than one class values. An example of a multi-label case can be a patient record having multiple diagnosis such as "Fever" and "Diabetes" at the same time. On the contrary, in this study we only consider that each record has a single class value associated with it.

A decision tree has nodes and leaves as shown in Figure 1. The rectangles are nodes and the ovals are leaves which are numbered from 1 to 3 in the figure. An attribute is tested at each node. If the attribute tested (i.e. the test attribute) at a node is numerical (such as the attribute A348) then typically there are two edges from the node. One of the edges is labeled "$> c$" while the other edge is labeled "$<= c$", where c is a constant which is an element of the domain of the test attribute. The value "c" is called the splitting point for the attribute. If the attribute is categorical then there are typically as many edges from the node as the domain size of the attribute, each labeled by a categorical value drawn from the attribute domain. The edges protruding from a node divide the data set into mutually exclusive partitions.

Each leaf of a tree has information on the number of records (of the leaf) belonging to a class value, for all class values. For example, in Leaf 1 (Figure 1) there are eleven records having Class 1 and one record

The work was supported by the 1st Author's CRiCS Seed Grant 2010.

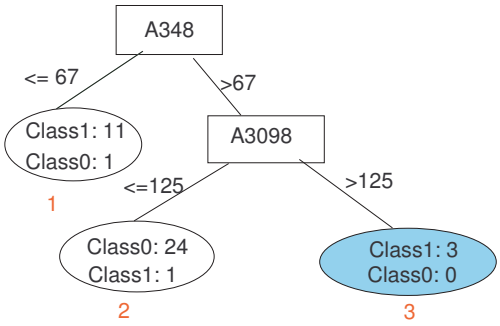

Figure 1: A decision tree obtained from the Central Nervous System data set

having Class 0. The class value of the maximum number of records of a leaf is called the majority class and other class values are called minority classes. If all records belonging to a leaf have the same class value then the leaf is called homogeneous, otherwise the leaf is called heterogeneous [13, 5]. In Figure 1, Leaf 3 is a homogeneous leaf and the other two leaves are heterogeneous. Each leaf has a logic rule that describes the test attributes and the split points (values) of the test attributes for the leaf. For example, the logic rule for Leaf 1 of Figure 1 is "A348 $\leq$ 67 $\Rightarrow$ Class1 (11) & Class0 (1)".

A decision tree is typically used for knowledge discovery as it extracts patterns (logic rules) from a data set. It can also be used to predict the class of an unlabeled record. For example, if an unlabeled record has attribute "A348" = 80 and "A3098" = 130 then from the tree shown in Figure 1 we can conclude that the record falls in Leaf 3 and therefore we can predict that the class value of the record is Class 1. The extraction of hidden pattern and the ability to make good predictions can be very useful in various decision making processes.

In a natural data set usually there are additional logic rules that are almost as good as the logic rules discovered by a single tree. Therefore, it is not unlikely to be able to build another good quality decision tree, which is different from the single tree generated by an algorithm such as C4.5 [18, 19] and Explore [10]. This is why we get different decision trees from a data set when we apply different algorithms.

The existence of multiple patterns is also evident from the experiments on privacy preserving data mining through noise addition [9, 12]. Controlled noise was added to a data set in such a way so that the gain ratios of all attributes tested at various nodes (of the single tree obtained from a data set) remain the same at the node levels even after noise addition. By gain ratio at the "node level" we mean the gain ratio, of the test attribute, for the horizontal data segment represented by the node. The tree shown in Figure 1 is built from a data set. Noise is added to the data set in such a way so that the gain ratio of the root attribute $A348$ remains unchanged for the entire data set, since a root node represents the whole data set. However, the gain ratio of the test attribute $A3098$ (tested by the node at Level 1) remains unchanged for the horizontal segment, represented by the node, where all records have $A348 > 67$ (see Figure 1).

It was expected to get a decision tree (from the perturbed data set) exactly the same as the decision tree from the original data set. However, the experiments frequently built trees that were different from the original tree. A possible explanation was that while adding a little amount of noise, although the gain ratio of the test attributes were preserved, gain ratios of some of the non test attributes were undeliberately improved to the level where they became even better than the original test attributes. Therefore, trees obtained from the perturbed data sets had different test attributes from the test attributes of the tree obtained from the original data set [9, 11]. Since only a very little amount of noise was added in the experiments, it is evident from the analysis that there are some attributes which are not tested in the original tree but still have almost as good gain ratios as the gain ratios of the attributes tested by the tree.

It was also noticed in the experimental results that the prediction accuracies of the trees obtained from the perturbed data sets were almost as good as the accuracy of the tree obtained from the original data set. Therefore, we argue that a data set can have multiple patterns, that can be extracted and represented by multiple trees. By multiple trees we mean a set of trees each having a set of logic rules as shown in Figure 1.

One obvious way to build multiple trees from a data set would be to apply a number of different single tree building algorithms on the data set. However, in that case a data miner needs to have access to a number of algorithms/softwares. He/she can only build as many trees as the number of algorithms he/she has access to. Therefore, with this approach a data miner can only build a limited number of trees. Moreover, the trees are not built systematically in order to extract multiple alternative patterns. Therefore, many of the trees can be very similar to each other. The existence of multiple patterns in a data set encourages us to systematically build multiple trees by deliberately choosing different test attributes for different trees. Multiple trees can be used collectively to achieve a better prediction accuracy and discover more logic rules/patterns.

In this paper we present a novel technique to systematically build a forest of decision trees using only those attributes having a high ability to classify a record. We apply our novel technique to build multiple trees on natural data sets. We also present two novel voting techniques in order to use the multiple trees collectively to predict the class values of unlabeled records. We implement several well known existing techniques and compare their prediction accuracy against SysFor. In Section 2 we discuss several existing techniques. Section 3 describes SysFor in more detail. Experimental results are presented in Section 4, and Section 5 presents concluding remarks.

2 Background Study

A technique called "Cascading and Sharing Trees (CS4)" [14, 15, 16] takes the number of trees to be generated as a user input. CS4 then orders the attributes by their gain ratios. It considers the ith best attribute as the root attribute of the ith tree. After the selection of the root attribute the data set is divided into mutually exclusive horizontal segments based on the values of the root attribute. For example, if the root attribute is numerical the data set is divided into two segments using the best split point of the root attribute, where all records in one segment have the values (of the root attribute) greater than the split point and all records in the other seg-

ment have values less than or equal to the split point. If the root attribute is categorical the data set is divided into segments by the values of the root attribute where all records in a segment have the same value for the root attribute, and records belonging to any two segments have different values.

A single tree building algorithm such as C4.5 [18, 19] is then applied on each segment to build the tree. A tree built on a segment is considered as a subtree of the ith tree. The root node of each subtree is therefore considered as a child node of the root node (of the ith tree) and thus all subtrees are combined to build the ith tree. CS4 builds the user defined number of trees only when the total number of non-class attributes of a data set is greater than or equal to the user defined number of trees. Otherwise, it builds as many trees as the number of non-class attributes in the data set. Each tree has a unique root attribute.

In order to predict the class value of an unlabeled record, CS4 uses all of the trees through a voting system [14, 15, 16], which we call *CS4 voting*. CS4 voting first calculates "coverage" of the leaves of each tree. Coverage of a leaf is the proportion of the records, in the leaf, having the majority class to the total number of records in the whole data set having the same class. For example, if a leaf has 20 records having the majority class say C_1 and the whole data set has 100 records having the class C_1 then the coverage of the leaf is 20%.

An unlabeled record falls in one and only one leaf of each tree. The majority class belonging to the leaf of a tree is the predicted class value of the unlabeled record according to the tree. Different trees may predict different class values. CS4 voting groups the trees such that trees belonging to the same group give the same prediction for the record, and trees belonging to different groups give different predictions. For an unlabeled record CS4 voting then sums up the coverage values of the leaves (belonging to the trees of a group) that the record falls in. The sum of the coverage values is considered as the total vote in favour of the class value that the group predicts. Similarly, sum of the coverage values are calculated for each group. The group having the highest sum wins. Therefore, the class predicted by the winning group is considered as the predicted class value for the record.

We argue that CS4 has several limitations. For example, it can only build (at most) as many trees as the number of non-class attributes of a data set. Therefore, for a low dimensional data set it is unable to build many trees. Another limitation is that it may build some poor quality trees since it uses the ith best attribute for the ith tree, regardless of the classification ability of the attribute in terms of gain ratio. If the ith best attribute has very low gain ratio we may get a poor quality tree having low accuracy and misleading logic rules.

In 2009 Abellan and Masegosa proposed a multiple tree building technique called "An Ensemble method using credal decision trees (Credal)" [3] which is very similar to CS4 except that it uses Imprecise Info-Gain (IIG) [1, 2] based ranking of the attributes as opposed to the Information Gain based ranking. It also has a slightly different voting system to predict the class value of a new record. From each tree it calculates a vote for each class value. That is for each tree it first identifies the leaf where the record falls in and calculates the proportion of records supporting a class value over the total number of records in the leaf. This proportion is considered as the vote for the class value from the tree. Similarly, vote for the class value is calculated from each tree and finally all votes for the class value are added to get the final vote. Final votes for all class values are also calculated. The class value having the highest final vote is considered as the predicted class for the record.

Another multiple tree building technique called "Maximally Diversified Multiple Decision Tree Algorithm (MDMT)" [7, 8] attempts to build multiple trees where an attribute is tested at most in one tree. Each tree tests a completely different set of attributes than the set of attributes tested in any other tree. MDMT builds the first decision tree using a traditional algorithm like C4.5 on a data set. All non-class attributes that have been tested in the first tree are then removed from the data set and C4.5 is again applied on the modified data set to build the second decision tree. The process continues until either the user defined number of trees are generated or all non-class attributes of the data set are removed. We argue that for a low dimensional data set MDMT may not be able to build sufficient number of trees. For a high dimensional dataset the technique may be able to build many decision trees. However, the quality of the later trees may not be high due to the removal of good attributes from the data set every time a tree is generated. A later tree can be generated from a data set having only attributes with poor gain ratios. Therefore, the technique may suffer from poor prediction accuracy as a result of insufficient number trees generated from a low dimensional data set and/or some poor quality trees generated.

In order to predict the class value of an unlabeled record, MDMT uses all generated trees through its voting system [7, 8], which we call the *MDMT voting*. The accuracies of the trees that have the same predicted class value for an unlabeled record are added and the result is considered as the vote in favour of the class value. Similarly, vote for each class value is calculated. The class value having the highest vote is considered as the final prediction for the record. MDMT uses the overall prediction accuracy of a tree in order to take into account the possible low quality of the later trees. We argue that a tree can have a high overall accuracy but a record can still fall in a leaf having high inaccuracy (i.e. having big proportion of minority records) resulting in a weak prediction for the record. Similarly, a tree can have a low overall accuracy but a record can still fall in a big leaf having high accuracy meaning a good confidence and support for the rule. Therefore, taking the accuracy of a tree (instead of the accuracy of a leaf) as an indicator of the prediction quality may not be a good idea. However, experimental results [7, 6] on several data sets indicate a superiority of CS4 and MDMT over several existing techniques including Random Forest, AdaBoostC4.5, SVM method, BaggingC4.5 and C4.5.

3 Our Technique

We now describe our novel technique to build *SysFor*: A Systematically Developed Forest of Multiple Decision Trees from a data set. We use the following steps to build a SysFor.

Step 1: Find a set of "good attributes" and corresponding split points based on a user defined "goodness" threshold and "separation" threshold. For a numerical attribute, the same attribute can be chosen more than once in the set of good attributes if the numerical attribute has high gain ratios with more than one different split points that are well separated and not adjacent to each other.
Step 2: If the size of the set of good attributes is less than a user defined number of trees then choose

each good attribute one by one as the root attribute (at Level 0) of a tree, and thereby build as many trees as the number of good attributes. Else, build user defined number of trees by considering as many good attributes as the number of trees.
Step 3: If the number of trees built in Step 2 is less than the user defined number of trees, then build more trees (until the user defined number of trees are built or the maximum number of possible trees are built) by using alternative good attributes at Level 1 of the trees built in Step 2. The alternative good attributes are chosen from the set of good attributes for the nodes at Level 1 of the trees built in Step 2.
Step 4: Return all trees built in Step 2 and Step 3 as a SysFor.

The whole process of building SysFor is introduced in Algorithm 1, and a couple of supplementary algorithms Algorithm 2 and Algorithm 3. Algorithm 1 takes (as user input) a data set D_f, user defined number of trees N, "goodness" threshold θ, "separation" threshold α, minimum gain ratio R, pruning confidence factor C_F and minimum number of records in each leaf N_L. The last three inputs are used by C4.5 [18, 19] for building a single tree. We now explain the steps as follows.

The data set D_f has a number of non-class attributes and a class attribute. In Step 1 we first calculate the gain ratio [19, 10], for the whole data set, of each non-class attribute one by one. The non-class attributes can be categorical or numerical. If a non-class attribute is categorical we then add the attribute and its gain ratio in two sets A^s and G^s, respectively. Since (unlike a numerical attribute) a categorical attribute does not have a numerical split point we add a negative number (say -100) for the categorical attributes in a set of split points P^s. On the other hand, for a numerical non-class attribute A_i we first calculate the best gain ratio and store it in a set of chosen gain ratios g^c. We also store the corresponding split point in a set p^c. After adding a split point in p^c, we next explore all available split points for the attribute such that any available split point p_l maintains $\frac{abs(p_l - p^c_m)}{|A_i|} > \alpha, \forall p^c_m \in p^c$. Among all available split points, the split point having the highest gain ratio is again added in the set p^c. Also the highest gain ratio is added in the set g^c (see Algorithm 2).

We continue the process of adding gain ratios and corresponding split points in g^c and p^c, and calculating available split points recursively until we have at least one available split point. Note that as the size of p^c grows the size of the set of available split points shrinks. The gain ratios in g^c and corresponding split points in p^c are then added in the set G^s and P^s. We also add the numerical attribute A_i in the set A^s as many times as the size of p^c. We continue the process of adding values in A^s, G^s and P^s for all non-class attributes. After the values are added the set G^s is sorted in a descending order of the gain ratios, and A^s and P^s are rearranged accordingly. All the gain ratios in G^s that have differences from the best gain ratio (i.e. the gain ratio stored in the first element $G^s[0]$) greater than a user defined goodness threshold θ are removed from G^s. A^s and P^s are also reorganised accordingly by removing the entries for which corresponding values in G^s have been removed. A^s is therefore the set of good attributes, G^s and P^s are the sets of gain ratios and split points of the good attributes. Note that with different split points a numerical attribute can appear more than once in the set of good attributes. However, the split points have to be different enough to satisfy the separation threshold requirement as explained before.

In Step 2, we pick the attributes and corresponding split points one by one from the set of good attributes and set of split points. Based on the gain ratios we first choose the best attribute, then the second best and so on. If the attribute picked is categorical then the whole data set is divided into as many mutually exclusive horizontal segments as the domain size, where all records in a segment have the same value for the attribute and records belonging to different segments have different values. If the attribute is numerical the data set is divided into two horizontal segments based on the split point. A gain ratio based decision tree algorithm such as C4.5 is then applied on each horizontal segment to build a tree. The root node of each of these trees is then joined as a child with the attribute picked from the set of good attributes as the parent. Thus we build a tree having a good attribute as the root (see Algorithm 3). If the number of good attributes is less than the user defined number of trees then choose each good attribute one by one as the root attribute (at Level 0) of a tree, and thereby build as many trees as the number of good attributes. Otherwise, build user defined number of trees by considering the necessary number of best attributes (from the set of good attributes) one by one as the root attributes.

If the number of trees built in Step 2 is less than the user defined number of trees, then SysFor algorithm builds more trees in Step 3 until the user defined number of trees are built or the maximum number of possible trees are built (see Algorithm 1). The algorithm first uses the root attribute of the first tree built in Step 2 in order to divide the whole data set in horizontal segments. It then prepares a set of good attributes, a set of corresponding split points and a set of gain ratios for each horizontal segment. Based on the numbers of good attributes of all segments a number of possible trees t_p is calculated using $t_p = \frac{\sum_{j=1}^{|\overline{D_f}|} |A^g_j| \times |D_j|}{\sum_{j=1}^{|\overline{D_f}|} |D_j|}$, where $|\overline{D_f}|$ is the number of data segments, $|D_j|$ is the number of records in the jth segment, and A^g_j is the set of good attributes for the jth segment. Note that t_p is the weighted average of the number of good attributes of the segments. Therefore, some segments may have number of good attributes less than or equal to t_p while other segments having number of good attributes greater than t_p.

If the number of good attributes of a segment is greater than or equal to t_p, then t_p number of trees are built from the segment by using the alternative good attributes for the segment. However, if number of good attributes is less than t_p then as many trees as the number of good attributes are first built using alternative good attributes. All remaining trees are then built using the best attribute from the list of good attributes. A tree is built at a time from a segment, for all segments and the trees are joined by connecting their roots (as children) to the root of the first tree built in Step 2. Therefore, the algorithm finally builds t_p number of trees having the root attribute same as the root attribute of the 1st tree built in Step 2. The algorithm then uses the root attribute of the 2nd tree built in Step 2 in order to build more trees using the good attributes at Level 1 of the 2nd tree. The process of building more trees continues until the requested number of trees are built or all trees built in Step 2 are used up.

Once SysFor is built we can use any voting system such as CS4 and MDMT voting (as explained in

Section 2) in order to predict the class values of unlabeled records. In this study we also present two novel voting systems for SysFor called SysFor Voting-1 and SysFor Voting-2. We now explain them one by one as follows.

For each new record SysFor Voting-1 first finds (from all trees) the list of leaves (logic rules) L that the record belongs to. There will be only one such leaf from each tree. We find the set of leaves $L^{100} \subseteq L$ where a leaf $L_i \in L^{100}$ has the following properties. The leaf L_i has 100% accuracy i.e. all records belonging to the leaf have the same class value. Moreover, the leaf has support greater than a user defined "Support threshold" or $\frac{|D_f|}{|N|} \times RulePercentage$, where $|D_f|$ = total number of records in the data set, $|N|$ = number of leaves of the tree. The Support threshold and Rule Percentage are user defined values. Default values of Support threshold and Rule Percentage can be chosen as 5 and 0.10, respectively. For each class value we take a voting among L^{100}. If the class attribute has a domain size $|C|$ then for a class value $C_i \in C$ we calculate the vote $Vote(C_i) = \sum_{j=1}^{|L^{100}|} \frac{|R_j^{C_i}|}{|R_j|}$, where $|R_j^{C_i}|$ is the number of records having C_i class value in the jth rule, and $|R_j|$ is the total number of records in the jth rule. If the $|L^{100}| = 0$ then for each class value we take a voting among all leaves L, instead of L^{100}, as $Vote(C_i) = \sum_{j=1}^{|L|} \frac{|R_j^{C_i}|}{|R_j|}$. The class value having the highest vote is the final prediction for the record.

In SysFor Voting-2 we take an aggressive approach where in order to predict the class value of a new record we first find all the leaves L (from all trees) that the record falls in. We then determine the leaf $L_i^{max} \in L$ that has the highest accuracy. Finally, the majority class value of L_i^{max} is considered as the predicted class value of the new record. If there are more than one leaves having the highest accuracy then the leaf having the highest support (number of records) among the leaves with the highest accuracy is considered to be the winner.

4 Experimental Results

We first apply our algorithm on a data set called Contraceptive Method Choice (CMC) available from UCI machine learning repository [17]. CMC data set has 1473 records, each record having 7 categorical non-class attributes, 2 numerical non-class attributes and one categorical class attribute. The class attribute "Contraceptive Method Used" has three categorical values, 1, 2 and 3. In this experiment we consider that the user defined number of trees is 60. We also use Goodness threshold = 0.3, and Separation threshold = 0.3. Moreover, for C4.5 (while used within our algorithm and individually) we assign Confidence factor = 25%, Minimum Gain Ratio = 0.01 and Minimum number of records in each leaf = 10. For SysFor Voting-1 we use Support threshold = 5 and Rule Percentage = 0.10. We apply CS4 [14, 15, 16] and MDMT [7, 8] on the data set with user defined number of trees equal to 60. We also build a decision tree from the data set by using C4.5 algorithm [18, 19].

The main purpose of multiple tree generation by SysFor is to perform a better knowledge discovery through the extraction of high quality multiple patterns. They are likely to give a better insight into the data set and therefore be useful in various decision making processes. In order to evaluate the quality of

Input: D_f, N, θ, α, R, C_F, N_L
Output: a set of trees T

initialize a set of decision trees T to null;
initialize a set of good attributes A^g to null;
initialize a set of split points P^g to null;
set A^g, and P^g by calling
GetGoodAttriubtes(D_f, θ, α);
initialise i to 1;
while $|T| < N$ *and* $|T| < |A^g|$ **do**
  T_i = **BuildTree**(D_f, $A_i \in A^g$, $P_i \in P^g$, R, C_F, N_L);
  $T = T \cup T_i$;
  $i = i + 1$;
end
$i = 1$;
initialise K to $|T|$;
while $|T| < N$ *and* $i \leq K$ **do**
  `/* divide the data set` D_f. `*/`
  if A_i *is categorical* **then**
    $D_f = \{D_1, D_2, \ldots D_{|A_i|}\}$;
  end
  if A_i *is numerical* **then**
    $D_f = \{D_1, D_2\}$;
  end
  `/*` $|\overline{D_f}|$ `= number of data segments in` D_f, $|D_j|$ `= number of records in the` j`th segment */`
  for $j = 1$ *to* $|\overline{D_f}|$ **do**
    initialise A_j^g, and P_j^g to null;
    A_j^g, and P_j^g =
    GetGoodAttributes(D_j, θ, α);
  end
  initialise number of possible trees, t_p to null;
  calculate, $t_p = \frac{\sum_{j=1}^{|\overline{D_f}|} |A_j^g| \times |D_j|}{\sum_{j=1}^{|\overline{D_f}|} |D_j|}$;
  initialise x to 1;
  while $|T| < N$ *and* $x \leq t_p$ **do**
    for $j = 1$ *to* $|\overline{D_f}|$ **do**
      if $|A_j^g| > x$ **then**
        t_j = **BuildTree**(D_j, $a_{x+1} \in A_j^g$, $p_{x+1} \in P_j^g$, R, C_F, N_L);
      end
      if $|A_j^g| \leq x$ **then**
        t_j = **BuildTree**(D_j, $a_1 \in A_j^g$, $p_1 \in P_j^g$, R, C_F, N_L);
      end
    end
    build a tree T_{new} by joining the root node of each t_j $(1 \leq j \leq |\overline{D_f}|)$ as a child node with the root node of T_i;
    $T = T \cup T_{new}$;
    $x = x + 1$;
  end
  $i = i + 1$;
end
return T;

Algorithm 1: Building a SysFor.

Input: D_f, θ, α
Output: A^s, P^s
GetGoodAttributes(D_f, θ, α)

initialise a set of attributes A^s to null;
initialise a set of split points P^s to null;
initialise a set of gain ratios G^s to null;
for *each attribute* A_i *in* A **do**
  /* A is the set of all non-class attributes */
  if A_i *is categorical* **then**
    calculate gain ratio of A_i, G_{A_i};
    add A_i into A^s;
    add -100 into P^s;
    add G_{A_i} into G^s;
  end
  if A_i *in numerical* **then**
    initialise a set of chosen split points p^c to null;
    initialise a set of chosen gain ratios g^c to null;
    create a set of available split points p^a of A_i;
    initialise a set of gain ratios g^a to null;
    calculate the gain ratio for each split point in p^a;
    add the gain ratios in g^a;
    while p^a *is not null* **do**
      find the highest gain ratio and corresponding split point from g^a and p^a, respectively;
      add the highest gain ratio and corresponding split point into g^c and p^c, respectively;
      recalculate available split points such that each available split point p_l maintains $\frac{abs(p_l - p^c_m)}{|A_i|} > \alpha$,$\forall p^c_m \in p^c$;
      reset p^a with the new set of available split points;
      reset g^a accordingly;
    end
    while p^c *is not null* **do**
      add A_i into A^s;
      add the best gain ratio of g^c and corresponding split point into G^s and P^s, respectively;
      remove the best gain ratio of g^c and corresponding split point from g^c and p^c, respectively;
    end
  end
end
sort G^s in the descending order of gain ratios in G^s, and also rearrange P^s and A^s accordingly;
remove the elements in G^s, and corresponding elements in A^s and P^s where the difference between a gain ratio and the best gain ratio in G^s is greater than θ;
return A^s, and P^s;

Algorithm 2: Finding a set of good attributes, and a corresponding set of split points.

Input: d, a, p, r, c_f, n_l
Output: a decision tree T
BuildTree(d, a, p, r, c_f, n_l)

if a *is categorical* **then**
  /* divide the data set into $|a|$ number of segments such that in each segment all records have only one value of a and records of any two segments have different values of a */
  $d = \{d_1, d_2, \ldots d_{|a|}\}$;
end
if a *is numerical* **then**
  /* divide the data set into two segments such that in one segment all records have value of a greater than the split point p and in the other segment all records have values for a less than or equal to p */
  $d = \{d_1, d_2\}$;
end
initialise i to 1;
initialise a set of trees t to null;
while $i \leq |d|$ **do**
  t_i = call C4.5(d_i, r, c_f, n_l);
  $t = t \cup t_i$;
  $i = i + 1$;
end
build a tree T by joining the root attribute of each $t_i \in t$ as a child node to the parent node that tests a as the root attribute of T;
return T;

Algorithm 3: Building a single tree given a data set and a particular root attribute.

the trees generated by different algorithms we compare the prediction accuracies of the trees on a testing data set. We use our voting techniques for trees generated by SysFor. Similarly, we use MDMT voting for MDMT trees and CS4 voting for CS4 trees. For C4.5 we use the conventional prediction system where the majority class value of the leaf that a record belongs to is predicted as the class value of the record. We use 3 fold cross validation for prediction purpose, i.e. we divide a data set into three equal sized mutually exclusive horizontal segments and build decision trees from two segments while test prediction accuracy on the third segment. Each segment is used once in turn as a testing set for prediction accuracy test. Prediction accuracies are presented in Table 1.

Data Set	Technique	Prediction accuracy			Avg.
		Cross Val. 1	Cross Val. 2	Cross Val. 3	
CMC	SysFor (Voting-1)	51.53	55.19	52.75	53.16
	SysFor (Voting-2)	52.34	50.31	50.92	51.19
	MDMT	47.45	46.23	48.27	47.32
	CS4	49.69	49.94	50.10	49.91
	C4.5	48.88	52.34	53.56	51.59

Table 1: Prediction Accuracy: 3 Fold Cross Validation on Contraceptive Method Choice (CMC) Data set.

It is clear from Table 1 that SysFor with its Voting-1 performs better than all other techniques in terms of prediction accuracy. SysFor Voting-2 also performs better than MDMT and CS4. Moreover, an important advantage of SysFor over CS4 and MDMT is its ability to build many trees even from a low dimensional data set such as CMC. For example, SysFor builds 60 trees from CMC data set since in this experiment the user defined number of trees is 60. However, CS4 and MDMT produce only 9 and 3 trees, respectively. CS4 produces 9 trees because there are altogether 9 non-class attributes in the data set. It uses each of the nine attributes one by one as a root attribute regardless of the classification ability (gain ratio) of the attribute. MDMT builds 3 trees because it runs out of attributes. All non-class attributes are used up by the three trees.

In order to explore the trend of accuracy change for different user defined number of trees, we build sets of 60, 50, 40, 30, 20 and 10 trees and estimate the prediction accuracy of each set. In Figure 2 we present the average prediction accuracies, using three fold cross validation, for different sets of trees. For SysFor we use its Voting-1 during this test. The prediction accuracies of MDMT and CS4 techniques remain unchanged over different number of trees because the techniques fail to produce more trees even if the user defined number of trees is high. Since SysFor can build as many trees as required by a user we get a variation in accuracies. However, it appears that higher number of trees do not always result in higher accuracy. We find the maximum accuracy (53.7%) of SysFor for 20 trees. Fifty trees produced better accuracy (53.49%) than 10, 30, 40 and 60 trees. The trend of accuracy change for different number of trees deserves a more thorough investigation. However, for any number of trees (10, 20, 30 etc.) SysFor results in higher accuracy than the accuracies of other techniques.

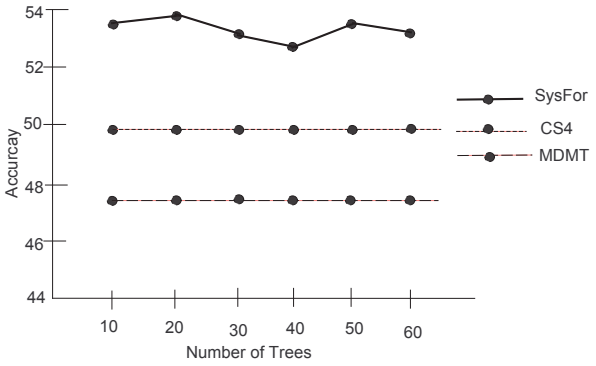

Figure 2: Prediction Accuracy VS User Defined Number of Trees

To evaluate the effectiveness of our voting technique we use our Voting-1 with trees generated by MDMT and CS4. We find an accuracy improvement in MDMT and CS4 when they use SysFor Voting-1. However, the accuracies are still lower than the accuracy of SysFor with its Voting-1 (see Table 1 and Table 2). Moreover, SysFor suffers from an accuracy drop when it is used along with CS4 and MDMT voting. The result is shown in Table 2.

Data Set	Technique	Prediction accuracy			Avg.
		Cross Val.1	Cross Val.2	Cross Val.3	
CMC	SysFor + CS4 Voting	50.92	52.75	51.12	51.60
	SysFor + MDMT Voting	51.73	54.18	51.93	52.61
	MDMT + SysFor Voting-1	46.44	52.34	49.69	49.49
	CS4 + SysFor Voting-1	52.14	53.56	51.53	52.41

Table 2: Effectiveness of Our Voting Technique

We also explore the quality of each individual tree generated by SysFor, CS4 and MDMT through the individual prediction accuracy of a single tree on the testing data set. In Figure 3 we present the individual prediction accuracy of the first 25 trees generated by SysFor, all 9 trees generated by CS4 and all 3 trees generated by MDMT, in cross validation number 2. SysFor trees maintain better accuracy than CS4 and MDMT trees. The 22nd SysFor tree has the highest accuracy of 54.38% among all 25 trees. This indicates that even after the generation of many trees SysFor is still capable of building high quality trees. Out of the 25 SysFor trees 7 have better accuracy than any tree generated by CS4 and MDMT. Both MDMT and CS4 trees suffer from quality drop after first few trees. Specially the quality of MDMT trees drop very sharp. The average accuracy of 25 SysFor trees, 9 CS4 trees and 3 MDMT trees are 50.40, 49.47 and 46.03, respectively. We also note that the combined accuracy (55.19%) of all SysFor trees is better than the accuracy of the best tree (the 22nd tree) indicating the

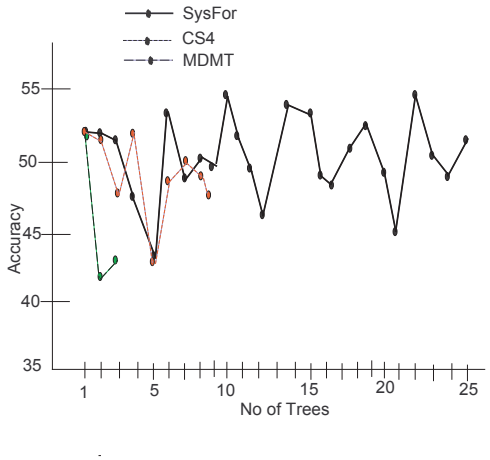

Figure 3: Prediction Accuracy of an Individual Tree

usefulness of SysFor Voting-1 in taking advantage of multiple trees.

We next apply SysFor, MDMT, CS4 and C4.5 algorithms on two high dimensional data sets, namely Lung Cancer (University of Michigan) data set and Central Nervous System data set available from Kent Ridge Biomedical Datasets [4]. The Lung Cancer data set has 96 records, each having 7129 non-class numerical attributes and a categorical class attribute. The class attribute has two values, cancer and normal. Out of 96 records, 86 are labeled as cancer and 10 as normal.The Central Nervous System data set has 60 records each having 7129 non-class numerical attributes and a categorical class attribute with class values Class 0 and Class 1. Out of 60 records 21 are for survivors and 39 are for failures. We build 20 trees from each data set using Goodness threshold = 0.2, and Separation threshold = 0.3. Moreover, for C4.5 we assign Confidence factor = 25%, Minimum Gain Ratio = 0.01 and Minimum number of records in each leaf = 2. The prediction accuracies of the techniques are presented in Table 3 and Table 4.

Data Set	Technique	Prediction accuracy			Avg.
		Cross Val.1	Cross Val.2	Cross Val.3	
Lung Cancer	SysFor (Voting-1)	100	100	100	100
	SysFor (Voting-2)	87.5	93.75	93.75	91.67
	MDMT	100	100	100	100
	CS4	100	100	100	100
	C4.5	100	96.88	100	98.96

Table 3: Prediction Accuracy on High Dimensional Lung Cancer Michigan Data set

For the Lung Cancer data set the accuracy of C4.5 single tree is very high and therefore, the multiple tree techniques do not have much room for further improvement. However, all of them improved the accuracy of a single tree from 98.96% to 100%. SysFor with its Voting-1 also achieves 100% accuracy. Voting-2 achieves 91.67% accuracy. Although Voting-2 does not perform very well on CMC and Lung Cancer data set it does very well on Central Nervous System data set. SysFor with Voting-2 achieves 65% accuracy whereas MDMT, CS4 and C4.5 achieve

Data Set	Technique	Prediction accuracy			Avg.
		Cross Val.1	Cross Val.2	Cross Val.3	
Cent. Nerv. Sys.	SysFor (Voting-1)	80	30	45	51.67
	SysFor (Voting-2)	70	70	55	65
	MDMT	60	65	45	56.67
	CS4	65	55	50	56.67
	C4.5	50	30	40	40

Table 4: Prediction Accuracy on High Dimensional Central Nervous System Data set

56.67%, 56.67% and 40% accuracy, respectively. SysFor with its two voting systems performs the best on all three data sets. However, finding an appropriate voting system appears to be an interesting problem. A possible solution to the problem can be to use an n Fold Cross Validation on a training data set in order to determine which voting system works better for the data set, and use the voting system for predicting the class value of a new record. We also plan to work on the voting system more deeply and present a better voting technique in future.

5 Conclusion

In this study we have presented a novel technique called SysFor for systematically building a forest of multiple trees. We have also presented two novel voting systems. A unique characteristic of the SysFor algorithm is that unlike existing techniques like CS4 and MDMT it only uses good attributes for building the trees. If it runs out of good attributes at a root node then unlike existing techniques it uses the good attributes available at level 1 of the best tree. Moreover, if it runs out of good attributes even at level 1 of the best tree it moves to the second best tree, third best tree and so on to use the good attributes at level 1 of those trees. Therefore, unlike the existing techniques, it does not require to build a tree that uses an attribute having poor gain ratio. Hence, the quality of logic rules and prediction accuracy of the trees should be better than the quality of logic rules and the prediction accuracy of the trees that are built using poor attributes. The algorithm also provides a user with the flexibility to adjust the extent of required "goodness" level of an attribute through an appropriate selection of the "goodness" threshold value that helps to evaluate whether the attribute qualifies as a good attribute.

Additionally, unlike existing techniques SysFor allows a numerical attribute to be chosen more than once if the split points are well separated and the gain ratios are high enough. We argue that well separated split points can actually extract different patterns/logic rules. For example, let us assume that an attribute "Age" has domain [0,120]. Logic rules R_1 : $Age > 70, Income > 80 \Rightarrow GoodCustomer$ and R_2 : $Age > 20, Suburb\_Code = Rich \Rightarrow GoodCustomer$ can be considered as significantly different since they have well separated split points for Age. SysFor algorithm aims to discover such patterns through multiple trees.

Another interesting characteristic of SysFor algorithm is that based on number of good attributes and number of records of horizontal segments at Level 1 the algorithm calculates possible number of trees

$t_p = \frac{\sum_{j=1}^{|D_f|} |A_j^g| \times |D_j|}{\sum_{j=1}^{|D_f|} |D_j|}$ such that bigger size segments have greater influence on number of possible trees calculation. Therefore, SysFor explores more alternative patterns when bigger size segments support this.

An important advantage of SysFor over existing techniques is its ability to build many trees even from a low dimensional data set due to its unique characteristic of using good attributes both at Level 0 and Level 1. Moreover, SysFor is easily extendable in the sense that it is not restricted to Level 0 and Level 1 nodes only, rather it can use the same approach on Level 2, Level 3 and so on nodes, if that is necessary and desirable.

We have also presented two novel voting systems. The main distinctive characteristic of our Voting-1 is that if among the multiple trees there is one or more trees having (pure) logic rule/s with high support (as defined by the Support threshold) and 100% accuracy (high confidence) for an unlabeled record then the class value of the unlabeled record is predicted based on only the pure logic rule/s. In that case the voting system overlooks the logic rules having heterogeneity as they are nonconclusive/uncertain. However, in the absence of such pure logic rules our voting technique uses all logic rules to predict the class value. SysFor Voting-2 uses the most accurate logic rule as it is considered to be our best bet.

Our initial experiments indicate that both SysFor and its voting systems are better than existing techniques and their voting systems. SysFor with its two voting systems has better accuracy than the accuracy of CS4, MDMT and C4.5, whereas experimental results [7, 6] indicate superiority of CS4 and MDMT over several other existing techniques including Random Forest, AdaBoostC4.5, SVM method, BaggingC4.5 and C4.5.

SysFor suffers from performance drop when it uses other voting systems such as CS4 Voting and MDMT Voting. However, other techniques make a performance improvement when they use our voting technique (Table 2) indicating a superiority of our voting techniques. Besides, SysFor always achieves better accuracy than the accuracy of other techniques, when an identical voting technique is used for all of them. Therefore, both SysFor and its voting techniques appear to be better than others.

From our initial experiments it appears that even on a low dimensional data set SysFor can build significantly bigger number of trees than the number of trees generated by CS4 and MDMT. When a single tree reveals a set logic rules (patterns) for a data set, a systematically developed set of multiples trees can reveal more logic rules in order to get much better information on the data set and thereby help better knowledge discovery.

Moreover, the quality of the trees built by SysFor is also consistently good as opposed to the trees generated by MDMT and CS4. SysFor builds trees one by one. Even a tree built at a later stage also has good accuracy on the testing data set. Within just the first 25 trees, there are seven SysFor trees which have better accuracy on testing data set than the accuracy of any (even the best) CS4 and MDMT trees (Figure 3). It also appears that regardless of the total number of trees generated SysFor always achieves better accuracy than the accuracy of MDMT and CS4 (Figure 2).

SysFor along with its Voting-1 performs the best on CMC and Lung Cancer data sets. However, it performs the best with its Voting-2 on Central Nervous System data set. Therefore, a data miner may want to test both voting systems on a training data set with an n-Fold Cross Validation and then for prediction purpose he/she can use the one that appears to be better on the data set. Our future work plans include the plan to improve the voting techniques, explore the relationships between number of trees and accuracy, improve SysFor algorithm, and explore if any type of data sets suits a voting system better.

References

[1] J. Abellan and S. Moral. Building classification trees using the total uncertainty criterion. *International Journal of Intelligent Systems*, 18(12), 2003.

[2] J. Abellan and S. Moral. Upper entropy of credal sets. application to credal classification. *International Journal of Approximate Reasoning*, 39(2-3), 2005.

[3] Joaquin Abellan and Andres R. Masegosa. An ensemble method using credal decision trees. *European Journal of Operational Research*, December 2009.

[4] Institute for Infocomm Research Data Mining Department. Kent ridge bio-medical dataset. available from http://datam.i2r.a-star.edu.sg/datasets/krbd/index.html. visited on 10.03.11.

[5] Jiawei Han and Micheline Kamber. *Data Mining Concepts and Techniques.* Morgan Kaufmann Publishers, San Diego, CA 92101-4495,USA, 2001.

[6] Hong Hu, Jiuyong Li, Ashley Plank, Hua Wang, and Grant Daggard. A comparative study of classification mehtods for microarray data analysis. In Jiuyong Li Simeon Simoff Peter Christen, Paul Kennedy and Graham Williams, editors, *Proceedings of the Australasian Data Mininig Conference (AusDM 2006)*, volume 61, pages 33–37, December, 2006.

[7] Hong Hu, Jiuyong Li, Hua Wang, Grant Daggard, and Mingren Shi. A maximully diversified multiple decision tree algorithm for microarray data classification. In *Proceedings of the Workshop on Intelligent Systems for Bioinformatics (WISB 2006), Conferences in Research and Practice in Information Technology (CRPIT)*, volume 73, 2006.

[8] Hong Hu, Jiuyong Li, Hua Wang, Grant Daggard, and Li-Zhen Wang. Robustness analysis of diversified ensemble decision tree algorithms for microarray data classification. In *Proceedings of the seventh International conference on Machine Learning and Cybernetics*, pages 115–120, 12-15 July 2008.

[9] Md Zahidul Islam. *Privacy Preservation in Data Mining through Noise Addition.* PhD thesis, School of Electrical Engineering and Computer Science, The University of Newcastle, Australia, June 2008.

[10] Md. Zahidul Islam. Explore: A novel decision tree classification algorithm. In *Proceedings of the 27th International Information Systems Conference, British National Conference on Databases (BNCOD 2010), Springer LNCS 6121 (in press)*, Dundee, Scotland, June 29 - July 01, 2010.

[11] Md Zahidul Islam, Payam Mamaani Barnaghi, and Ljiljana Brankovic. Measuring data quality: Predictive accuracy vs. similarity of decision trees. In *Proceedings of the 6th International Conference on Computer & Information Technology (ICCIT 2003)*, volume 2, pages 457–462, Dhaka, Bangladesh, 2003.

[12] Md Zahidul Islam and Ljiljana Brankovic. Privacy preserving data mining: A noise addition framework using a novel clustering technique. *Knowledge-Based Systems (accepted)*.

[13] Md Zahidul Islam and Ljiljana Brankovic. Noise addition for protecting privacy in data mining. In *Proceedings of of the 6th Engineering Mathematics and Applications Conference (EMAC 2003)*, volume 2, pages 85–90, Sydney, Australia, 2003.

[14] Jinyan Li and Huiqing Liu. Ensembles of cascading trees. In *Proceedings of the third IEEE international conference on Data Mining (ICDM 03)*, pages 9–17, Maebashi City, Japan, 2003. Australian Computer Society, Inc.

[15] Jinyan Li, Huiqing Liu, See-Kiong Ng, and Limsoon Wong. Discovery of significant rules for classifying cancer diagnosis data. *Bioinformatics*, 19(2):ii93–ii102, 2003.

[16] Jinyan Li and Kotagiri Ramamohanarao. A tree-based approach to the discovery of diagnostic biomarkers for ovarian cancer. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 04)*, pages 682 – 691, Maebashi City, Japan, 2004. Springer-Verlag Berlin Heidelberg.

[17] UCIrvine University of California. Uci machine learning. available from http://www.ics.uci.edu/∼mlearn/MLRepository.html. visited on 12.10.06.

[18] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, California, USA, 1993.

[19] J. Ross Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90, March 1996.