Constructing X -of- N Attributes for Decision Tree Learning

ZIJIAN ZHENG

School of Computing and Mathematics, Deakin University, Geelong, Victoria 3217, Australia

zijian@acm.org

Editor: Raul Valdes-Perez

Abstract. While many constructive induction algorithms focus on generating new binary attributes, this paper explores novel methods of constructing nominal and numeric attributes. We propose a new constructive operator, X -of- N . An X -of- N representation is a set containing one or more attribute-value pairs. For a given instance, the value of an X -of- N representation corresponds to the number of its attribute-value pairs that are true of the instance. A single X -of- N representation can directly and simply represent any concept that can be represented by a single conjunctive, a single disjunctive, or a single M -of- N representation commonly used for constructive induction, and the reverse is not true. In this paper, we describe a constructive decision tree learning algorithm, called XofN. When building decision trees, this algorithm creates one X -of- N representation, either as a nominal attribute or as a numeric attribute, at each decision node. The construction of X -of- N representations is carried out by greedily searching the space defined by all the attribute-value pairs of a domain. Experimental results reveal that constructing X -of- N attributes can significantly improve the performance of decision tree learning in both artificial and natural domains in terms of higher prediction accuracy and lower theory complexity. The results also show the performance advantages of constructing X -of- N attributes over constructing conjunctive, disjunctive, or M -of- N representations for decision tree learning.

Keywords: constructive induction, new attributes, decision tree learning, classification, induction

1. Introduction

Conventional inductive learning algorithms usually create theories containing only simple tests on single attributes selected from a set of task-supplied attributes which are used to describe the training data. This kind of induction is called *selective induction* (Michalski, 1983). A well-known elementary limitation of selective induction algorithms is that when task-supplied attributes are not adequate for describing hypotheses, their performance in terms of prediction accuracy and theory complexity is poor. To overcome this limitation, *constructive induction* algorithms (Michalski, 1978) transform the original instance space into a more adequate space by creating *new attributes*. By contrast to new attributes, the task-supplied attributes are called *primitive attributes*. New attributes are expected to be more appropriate for representing theories to be learned than primitive attributes from which the new attributes are constructed.

In real-world application domains, three different types of attribute, binary, nominal, and numeric,¹ are used to describe examples and concepts in the machine learning community. Different types of attribute have different advantages. For example, boolean attributes are very simple; whereas nominal and numeric attributes are complex but more powerful for

representing concepts. Note that attributes with more than two ordered discrete values can be specified as either nominal or numeric attributes.

Most selective induction algorithms can accept attributes of these three kinds. However, many existing constructive induction algorithms such as FRINGE (Pagallo, 1990), CITRE (Matheus & Rendell, 1989), CI (Zheng, 1992), LFC (Ragavan & Rendell, 1993), and CAT (Zheng, 1998) only construct new binary attributes by using logical operators such as conjunction, negation, and disjunction. On the other hand, ID2-of-3 (Murphy & Pazzani, 1991) creates at-least M -of- N attributes. An M -of- N representation consists of a set of conditions (attribute-value pairs) and a value M . For a given instance, the value of an at-least M -of- N representation is true if at least M of its conditions are true of the instance; it is false, otherwise. Similarly, we can define values of at-most and exactly M -of- N representations. Conjunctive and disjunctive representations are two special cases of at-least M -of- N representations. A single at-least M -of- N representation can represent any concept that can be represented by a single conjunctive or a single disjunctive representation, but the reverse is not true. Nevertheless, M -of- N representations still have binary values.

A few systems explore methods of constructing new numeric attributes using mathematical operators (e.g. BACON by Langley et al. (1987), and INDUCE by Michalski (1978)) or *attribute counting attributes*² (e.g. INDUCE by Michalski (1978), AQ17-DCI by Bloedorn and Michalski (1998), and AQ17-MCI by Bloedorn, Michalski and Wnek (1993)). In addition, systems such as LMDT (Brodley & Utgoff, 1992), SWAP1 (Indurkha & Weiss, 1991), and CCAF (Yip & Webb, 1994) construct linear machines (Brodley & Utgoff, 1992), linear discriminant functions, or canonical discriminant functions as new attributes. A linear machine consists of a set of linear discriminant functions (Brodley & Utgoff, 1992). When used as a test, it has multiple values, one for each class. Therefore, linear machines can be considered as nominal attributes with a fixed number of values for a given learning problem.³ Subsetting, used by learning algorithms such as C4.5, groups discrete values of a single primitive nominal attribute to form a new test (Quinlan, 1993). This can be thought as a method of constructing new nominal attributes. Pazzani (1996) explores methods of constructing Cartesian products as new nominal attributes.

This paper proposes a novel constructive operator, called X -of- N , and a new decision tree learning algorithm that constructs new attributes in the form of X -of- N representations. An X -of- N representation contains attribute-value pairs. Its value for a given example corresponds to the number of its attribute-value pairs that are true of the example. An attribute-value pair is true for an example if the corresponding attribute value of the example is the same as that in the attribute-value pair. Since X -of- N representations have ordered discrete values, they can be treated as either new nominal attributes or new numeric attributes for constructive induction.

The learning system described in this paper uses decision trees (Quinlan, 1993; Breiman et al., 1984) as its theory description language. At each decision node, it constructs one X -of- N attribute by using greedy search in the space defined by all the primitive attribute-value pairs of a domain. Decision trees with X -of- N representations as tests are referred to as *X -of- N trees*. However, the idea of constructing new nominal or numeric X -of- N attributes is not limited to decision tree learning. It is not difficult to extend this idea to rule learning.

The following section presents a definition of X -of- N representations and discusses their characteristics. Section 3 describes an approach to constructing new X -of- N attributes. The constructive induction algorithm XofN uses X -of- N s as nominal attributes to build decision trees. In Section 4, XofN is experimentally evaluated and compared with four constructive decision tree learning algorithms that generate new binary attributes in a set of artificial and natural domains. Section 5 addresses a potential problem of the XofN algorithm, and investigates approaches to alleviating the problem. Section 6 relates this research to existing work. Finally, Section 7 concludes and discusses some possible directions for future research.

2. X -of- N representations

The previous section has informally described X -of- N representations. Their formal definition is as follows:

Definition 1 (X -of- N representations). Let $\{A_i \mid 1 \leq i \leq \text{MaxAtt}\}$ be the set of attributes of a domain, and for each A_i , $\{V_{ij} \mid 1 \leq j \leq \text{MaxAttVal}_i\}$ be its value set, where MaxAtt is the number of attributes, and MaxAttVal_i is the number of different values of A_i .

An X -of- N representation is a set, denoted as:

$X\text{-of-}\{AV_k \mid AV_k \text{ is an attribute-value pair denoted as } "A_i = V_{ij}"\}$.

The number of attribute-value pairs in the X -of- N representation is called the size of the X -of- N representation. The value of the X -of- N representation can be any number between 0 and the number of different attributes that appear in the X -of- N representation.

Given an instance, the value of the X -of- N representation is X if and only if X of the AV_k are true of the instance. An attribute-value pair AV_k ($A_i = V_{ij}$) is true for an instance if and only if the attribute A_i of the instance has the value V_{ij} .

X -of- N representations are defined on binary and nominal attributes. Numeric attributes are transformed into binary or nominal attributes by discretization (Quinlan, 1993; Catlett, 1991; Fayyad & Irani, 1993; Van de Merckt, 1993). The definition presented above allows one primitive attribute to appear multiple times with different attribute values in a single X -of- N representation. This makes it easy for X -of- N representations to represent concepts containing *internal disjunctions* (Michalski, 1980). Internal disjunction, here, means a disjunction between values of a single variable (attribute).

The main advantage of X -of- N representations over the commonly used conjunctive, disjunctive, and M -of- N representations is that the former can directly represent more concepts than the latter. We expand on this point in the following subsection. Then, in Subsection 2.2, we discuss the *fragmentation* problem (Pagallo & Haussler, 1990), a difficulty that confronts nominal X -of- N representations.

2.1. Advantages of X -of- N representations

Many constructive induction algorithms use a selection of conjunction, disjunction, and negation as constructive operators. These operators can create new attributes to directly

represent conjunctive and disjunctive concepts. However, some other concepts such as parity concepts, at-least, exactly, at-most M -of- N concepts, and their possible combinations cannot be effectively represented. ID2-of-3 can only create at-least M -of- N representations. From the definition, we can see that, as a nominal attribute, the X -of- N representation can directly and simply represent all of the following types of concept:

1. conjunction (with or without internal disjunction) (as X -of- $N = N$),
2. disjunction (with or without internal disjunction) (as X -of- $N \geq 1$),
3. at-least M -of- N (as X -of- $N \geq M$),
4. at-most M -of- N (as X -of- $N \leq M$),
5. exactly M -of- N (as X -of- $N = M$),
6. even parity and odd parity (for even parity, as X -of- N in $\{0, 2, 4, \dots\}$), and
7. possible combinations of the above six types of concept.

Now, let us consider a few examples to further demonstrate this point.

Example 1 (Even parity problem with binary attributes). Given seven binary attributes $\{A, B, C, D, E, F, G\}$ with the value t or f , the even parity concept can be presented using an X -of- N representation as:

$$X\text{-of-}\{A = t, B = t, C = t, D = t, E = t, F = t, G = t\} \text{ in } \{0, 2, 4, 6\}.$$

For this concept, there are many alternative representations using X -of- N s including

$$X\text{-of-}\{A = f, B = f, C = f, D = f, E = f, F = f, G = f\} \text{ in } \{1, 3, 5, 7\},$$

$$X\text{-of-}\{A = f, B = f, C = t, D = t, E = t, F = t, G = t\} \text{ in } \{0, 2, 4, 6\}, \text{ and}$$

$$X\text{-of-}\{A = t, B = t, C = t, D = t, E = t, F = t, G = f\} \text{ in } \{1, 3, 5, 7\}.$$

Figure 1 gives their tree representations. It is not difficult to understand that all of them have exactly the same meaning. They are much less complex than a univariate tree that represents the even parity concept. For the issue of whether some branches of a decision node with an X -of- N as the test should be grouped together and how to do this, see Section 5.

Example 2 (Conjunctive, disjunctive, and M -of- N concepts with binary and nominal attributes). Given attributes and their possible values:

$$\begin{array}{lll} A: 1, 2 & B: 1, 2, 3, 4 & C: 1, 2, 3, 4 \\ D: 1, 2, 3, 4, 5 & E: 1, 2, 3, 4, 5 & F: 1, 2, 3, 4, 5 \\ G: 1, 2, 3, 4, 5 \end{array}$$

- (I). *Conjunction with internal disjunction*: The concept $(A = 1 \wedge (B = 2 \vee B = 4) \wedge C = 3)$ can be represented as: $X\text{-of-}\{A = 1, B = 2, B = 4, C = 3\} = 3$ with the X -of- N as a nominal attribute. Its tree representation is shown in figure 2(a).
- (II). *Disjunction with internal disjunction*: The concept $(A = 2 \vee B = 3 \vee (D = 1 \vee D = 4))$ can be represented using the decision tree as shown in figure 2(b). The X -of- N representation is treated as a nominal attribute in the tree.

Figure 1. Tree representations with nominal $X\text{-of-}N$ attributes for the even parity concept. Figure 2. Tree representations with $X\text{-of-}N$ attributes for: (a) conjunction with internal disjunction, (b) disjunction with internal disjunction.

(III). *Combination of $M\text{-of-}N$ concepts*: The concept “at-least 5 or exactly 3 or at-most 1-of- $\{A = 2, B = 3, C = 1, D = 4, E = 5, F = 2, F = 5, G = 3\}$ ” can be presented as in figure 3 with the $X\text{-of-}N$ representation as a nominal attribute.

Let $\{AV\}$ be the set of all possible attribute-value pairs defined by a given set of primitive attributes and their values. The number of all possible $X\text{-of-}N$ representations that can be created from $\{AV\}$ is the same as the number of all possible conjunctive representations or the number of all possible disjunctive representations that can be generated from $\{AV\}$, since each subset of $\{AV\}$ can define a single conjunctive representation, a single disjunctive representation, as well as a single $X\text{-of-}N$ representation. Further, this is smaller than the number of all possible at-least $M\text{-of-}N$ representations that can be created from $\{AV\}$, because each subset of $\{AV\}$ can define n different at-least $M\text{-of-}N$ representations that share the same subset of attribute-value pairs but have different values for M , where n is the number of different attributes appearing in the attribute-value pair subset.

Figure 3. A decision tree with a nominal X -of- N attribute for a combination of M -of- N concepts.

A single X -of- N representation can provide a finer grade partition of the instance space than a single conjunctive representation, a single disjunctive representation, or a single M -of- N representation that has the same set of attribute-value pairs. The set of the decision surfaces in the instance space produced by a single X -of- N representation is a super-set of the set of the decision surfaces produced by a single conjunctive representation, the set of the decision surfaces produced by a single disjunctive representation, and the set of the decision surfaces produced by all the at-least (or at-most, or exactly) M -of- N representations that have the same set of attribute-value pairs but have different values for M .

Therefore, a single X -of- N representation can directly and simply represent any concept that can be represented by a single conjunctive, a single disjunctive, or a single M -of- N representation given that they are created from the same set of attribute-value pairs, and the reverse is not true, indicating that X -of- N representations provide a powerful means for constructive induction. The two examples above show that X -of- N representations can directly and simply represent concepts that have concise conjunctive, disjunctive, or M -of- N representations. However, to represent concepts that have concise X -of- N representations, such as a parity concept, all conjunctive, disjunctive, and M -of- N representations require a very complex tree.

2.2. Fragmentation problem

As nominal attributes, X -of- N representations have a disadvantage, namely the *fragmentation* problem (Pagallo & Haussler, 1990). When large nominal X -of- N attributes are used as tests for decision trees, they quickly split training sets into a large number of small subsets. This makes subtree generation as well as new attribute construction at lower levels of a tree harder, thus resulting in premature termination of the growth of the tree.

We will propose three approaches to alleviating the fragmentation problem of nominal X -of- N attributes. They are subsetting, subranging, and forming binary splits. Details of these methods will be discussed later.

3. Building decision trees with nominal X -of- N attributes

We have argued that a single X -of- N representation can directly and simply represent any concept that can be represented by a single conjunctive, a single disjunctive, or a single M -of- N representation, and that the reverse is not true. Now, the questions are, “Can

nominal X -of- N attributes be automatically constructed for inductive learning to solve learning problems?”, and, “Does the idea of constructing nominal X -of- N attributes work well in some real-world domains or is it limited to artificial domains?”

We answer these questions in this section and the next section. This section presents the XofN algorithm. The following section experimentally evaluates XofN and shows that constructing nominal X -of- N attributes is useful for decision tree learning not only in artificial domains but also in some natural domains. Meanwhile, it is experimentally illustrated that nominal X -of- N attributes suffer from the fragmentation problem in some artificial logical domains. The investigation of possible solutions to this problem is left to Section 5.

The XofN algorithm provides an approach to generating and using X -of- N representations as new nominal attributes for decision tree learning. Tests at decision nodes of a decision tree are either primitive attributes or new nominal attributes in the form of X -of- N representations. During the generation of a tree, the construction of X -of- N attributes occurs.

3.1. Building decision trees

Like ID2-of-3 (Murphy & Pazzani, 1991), XofN consists of a single process while other constructive induction algorithms such as FRINGE (Pagallo, 1990) and AQ17-HCI (Wnek & Michalski, 1994) interleave two processes, namely selective induction and new attribute construction. As shown in Table 1, XofN recursively builds a decision tree by constructing, at each decision node, one new nominal X -of- N attribute based on primitive attributes using the *local training set*. The local training set at a node refers to those training examples that are traced down to this node during the generation of the tree. The main difference between XofN and C4.5 (Quinlan, 1993) is that the latter only selects one primitive attribute at each decision node.

If the new attribute constructed at a node is better than all the primitive attributes and previously created new attributes, XofN uses it as the test for the node; otherwise XofN discards it and uses the best of the primitive attributes and previously constructed new attributes. Like C4.5, XofN uses information gain ratio (Quinlan, 1993) as its test selection criterion.⁴ At each decision node when building a decision tree, besides creating one new X -of- N attribute, XofN considers reusing the X -of- N attributes constructed previously for other decision nodes. Att_{active} in the algorithm is for this purpose. It contains all the primitive attributes and all the X -of- N attributes that have been constructed and used so far.

As far as the issue of how frequently new attributes are reused is concerned, Table 2 shows some examples. At one trial of our experiment in the Cleveland heart disease domain (Blake, Keogh, & Merz, 1999) selected for examination at random, the XofN algorithm constructs and uses 10 X -of- N attributes. Three of these 10 new attributes are reused twice. At another trial with the same domain, the algorithm constructs and uses 15 X -of- N attributes. Among them, two are reused twice. In the Nettealk-stress domain (Blake, Keogh, & Merz, 1999), the XofN algorithm constructs and uses 141 X -of- N attributes at one trial. Among them, 34 are reused twice; 18 are reused three times; 12 are reused four times; 3 are reused five times; and 1 is reused six times. At another trial with the Nettealk-stress domain,

Table 1. Kernel of the XofN algorithm.

```

XofN-Tree( $Att_{primitive}, Att_{active}, D_{training}, C$ )
  INPUT:  $Att_{primitive}$ : a set of primitive attributes,
            $Att_{active}$ : a set of primitive and  $X$ -of- $N$  attributes for creating a test
                for the current decision node, initialized as  $Att_{primitive}$ ,
            $D_{training}$ : a set of training examples represented using  $Att_{active}$ ,
            $C$ : majority class at the parent node of the current node,
                initialized as the majority class in the whole training set.
  OUTPUT: a decision tree,
            $Att_{active}$ : modified by adding new  $X$ -of- $N$  attributes
                constructed at this node and its subtrees.

  IF ( $D_{training}$  is empty)
  THEN RETURN a leaf node labeled with  $C$ 
  ELSE
  {  $C :=$  the majority class in  $D_{training}$ 
    IF (all examples in  $D_{training}$  have the same class  $C$ )
    THEN RETURN a leaf node labeled with  $C$ 
    ELSE
    {  $Test_{best} := \mathbf{Find-Best-Test}(Att_{active}, D_{training})$ 
       $Att_{new} := \mathbf{Construct-X-of-N}(Att_{primitive}, D_{training})$ 
      IF ( $Att_{new}$  forms a test better than  $Test_{best}$ )
      THEN
      {  $Att_{active} := Att_{active} \cup \{Att_{new}\}$ 
         $Test_{best} :=$  the test formed using  $Att_{new}$ 
        Rewrite  $D_{training}$  using  $Att_{active}$ 
      }
      ELSE Dispose of  $Att_{new}$ 
      IF ( $Test_{best}$  is reasonable (with a positive gain or gain ratio value))
      THEN
      { Use  $Test_{best}$  to partition  $D_{training}$  into  $n$  subsets
         $D_1, D_2, \dots, D_n$ , one for each outcome of the test  $Test_{best}$ 
        RETURN the tree formed by a decision node with  $Test_{best}$ 
            and subtrees  $\mathbf{XofN-Tree}(Att_{primitive}, Att_{active}, D_1, C)$ ,
                         $\mathbf{XofN-Tree}(Att_{primitive}, Att_{active}, D_2, C)$ ,
                        ...,
                         $\mathbf{XofN-Tree}(Att_{primitive}, Att_{active}, D_n, C)$ 
      }
      ELSE RETURN a leaf node labeled with  $C$ 
    }
  }

```

XofN constructs and uses 144 X -of- N attributes. Among them, 41 are reused twice; 12 are reused three times; 4 are reused four times; and 3 are reused five times. These examples suggest that the reuse of X -of- N attributes does occur in decision tree learning in natural domains. From the experimental results in Section 4, we will see that constructing X -of- N attributes significantly increases the prediction accuracy of decision tree learning in both

Table 2. New attribute reuse, as examples, in two natural domains with two trials selected for examination at random for each domain.

Domain	The number of X -of- N attributes						
	Constructed & used	Used only once	Reused twice	Reused 3 times	Reused 4 times	Reused 5 times	Reused 6 times
Cleveland	10	7	3				
	15	13	2				
Nettalk-stress	141	73	34	18	12	3	1
	144	84	41	12	4	3	

of these two domains, and significantly decreases the complexity of learned trees in the Nettalk-stress domain.

3.2. Constructing nominal X -of- N representations

Details of constructing an X -of- N representation using the local training set at a decision node are shown in Tables 3 and 4. Function “Construct- X -of- N ()” performs simple greedy search in the space defined by primitive attributes and their values. The starting point of the search is an empty X -of- N attribute. At each search step, it applies one of two operators: Adding one possible attribute-value pair, or deleting one possible attribute-value pair. This is accomplished by function “Search- X -of- N ()”. To make the search efficient, the deleting operator is applied first if possible.⁵ During the search, XofN keeps the best of the X -of- N representations found so far for each possible size. Finally, function “Construct- X -of- N ()” returns the best of the X -of- N representations retained with respect to the new attribute evaluation function.

The information gain ratio (Quinlan, 1993) is used as the evaluation function for comparing and selecting new attributes. To avoid creating very complex new attributes that

Table 3. Algorithm for constructing an X -of- N attribute.

Construct-X-of-N ($Att_{primitive}, D_{training}$)
INPUT: $Att_{primitive}$: a set of primitive attributes,
$D_{training}$: a set of training examples.
OUTPUT: One X -of- N attribute.
Let $X\text{-of-}N_{best}[i]$ be the best X -of- N with i attribute-value pairs constructed
so far, initialized as $\emptyset$
Let $l := 0$
Let function SC ($l, X\text{-of-}N_{best}$) be the stopping criterion.
WHILE (SC ($l, X\text{-of-}N_{best}$) is not true) DO
$l := \text{Search-X-of-N}(Att_{primitive}, D_{training}, X\text{-of-}N_{best}, l)$
RETURN the best X -of- N in $X\text{-of-}N_{best}$

Table 4. One step search algorithm for X -of- N representations.

```

Search- $X$ -of- $N$ ( $Att_{primitive}$ ,  $D_{training}$ ,  $X\text{-of-}N_{best}$ ,  $l$  )
  INPUT:  $Att_{primitive}$ : a set of primitive attributes,
           $D_{training}$ : a set of training examples,
           $X\text{-of-}N_{best}$ : all best  $X$ -of- $N$ s of different sizes constructed so far
                      (each one has indicators of the adding and deleting
                      operator applications),
           $l$ : the number of attribute-value pairs in the  $X$ -of- $N$  constructed
              last time.
  OUTPUT: modified  $X\text{-of-}N_{best}$ ,
           modified  $l$ .

  IF ( $l = 0$ )
  THEN  $X\text{-of-}N_{old} := \emptyset$ 
  ELSE  $X\text{-of-}N_{old} := X\text{-of-}N_{best}[l]$ 
  IF ( $l > 2$  AND the deleting operator has not been applied to  $X\text{-of-}N_{old}$ )
  THEN
  { /* deleting one possible attribute-value pair */
     $X\text{-of-}N_{temp} :=$  the best  $X$ -of- $N$  created by deleting one attribute-value
                      pair from  $X\text{-of-}N_{old}$ 
    IF ( $X\text{-of-}N_{temp}$  is better than  $X\text{-of-}N_{best}[l - 1]$  )
    THEN
    {  $l := l - 1$ 
       $X\text{-of-}N_{best}[l] := X\text{-of-}N_{temp}$ 
    }
    ELSE Dispose of  $X\text{-of-}N_{temp}$ 
  }
  ELSE
  { /* adding one possible attribute-value pair */
     $l := l + 1$ 
     $X\text{-of-}N_{temp} :=$  the best  $X$ -of- $N$  created by adding one possible
                      attribute-value pair from  $Att_{primitive}$  into  $X\text{-of-}N_{old}$ 
    IF ( $X\text{-of-}N_{temp}$  is better than  $X\text{-of-}N_{best}[l]$  )
    THEN  $X\text{-of-}N_{best}[l] := X\text{-of-}N_{temp}$ 
    ELSE Dispose of  $X\text{-of-}N_{temp}$ 
  }
  RETURN  $l$ 

```

might overfit the training data, another criterion is added based on the MDL-based coding cost of new attributes. By complex, we mean that an X -of- N representation has a large number of attribute-value pairs. Overfitting is likely to occur when including this kind of new attribute. The Minimum Description Length (MDL) principle (Rissanen, 1978, 1983; Quinlan & Rivest, 1989) states that the best theory to learn from a dataset is the one that minimizes the sum of the coding cost of the theory and the coding cost of the dataset when encoded using the theory as a predictor for the dataset. The reader may want to refer to Rissanen (1978, 1983, 1986) for further details of MDL, and refer to Wallace and Boulton (1968),

Boulton and Wallace (1973a, 1973b), Wallace and Patrick (1993), and Hart (1987) for related work.

We use a similar encoding method to that described by Quinlan and Rivest (1989). The coding cost of a new attribute is the sum of two parts. One is $\text{Cost}_{\text{structure}}$ as defined by Eqs. (1) and (2) which is the number of bits for encoding the new attribute itself.

$$\text{Cost}_{\text{structure}} = \sum_j^N \text{Cost}_j - \log_2(N!) \quad (1)$$

$$\text{Cost}_j = \log_2(N_a) + n_j \times \log_2(N_{v_j}) - \log_2(n_j!) \quad (2)$$

Where, Cost_j is the cost for encoding each different primitive attribute j used in the new attribute. N_a is the number of primitive attributes available for constructing new attributes. N_{v_j} is the number of different values of attribute j , and n_j is the number of different values of attribute j that appear in the new attribute. N is the number of different primitive attributes that occur in the new attribute. Since the order of primitive attributes and the order of different values of each primitive attribute that appear in a new attribute do not matter, the coding costs are reduced by $\log_2(N!)$ bits and $\log_2(n_j!)$ bits in Eqs. (1) and (2) respectively.

The other part is for encoding the exceptions when applying the new attribute as a classifier to the local training data at the current decision node. After these training examples are split into subsets with one for each possible value of the new attribute, each example is labeled with the majority class of the subset to which it belongs. The exceptions (incorrectly labeled examples) in each subset are encoded in turn in the following manner. The resulting coding costs are then summed up. For each subset, we first encode the majority class. It costs $\log_2(C)$ bits, where C is the number of classes. We, then, indicate the positions of the exceptions. This costs $L(N_{\text{all}}, N_{\text{exceptions}}, N_{\text{all}} - 1)$ bits. N_{all} is the number of training examples in the subset. $N_{\text{exceptions}}$ is the number of exceptions in the subset. The function $L(n, k, b)$ (Quinlan & Rivest, 1989) equals $\log_2(b+1) + \log_2(\binom{n}{k})$. It is the bits needed for encoding a binary string of length n with k “1”s, where b is a known a priori upper bound on k . Note that, for multi-class problems, the upper bound on $N_{\text{exceptions}}$ is $N_{\text{all}} - 1$. For problems with more than two classes, we need to further encode the classes of exceptions. This is carried out by using an iterative approach. Actually, the calculation discussed above is used except that the most common class occurring among the exceptions (called the first alternative class) is used this time. In addition, the locations of the second-order exceptions⁶ within the exceptions are indicated. Therefore, in the formulas above, N_{all} is replaced with the number of exceptions; $N_{\text{exceptions}}$ is replaced with the number of the second-order exceptions; C decreases by one after each iteration since the number of possible classes for remaining exceptions is reduced by one. This process is repeated with higher order exceptions and higher order alternative classes until no further exceptions remain or C becomes one.

The newly constructed X -of- N representation ($X\text{-of-}N_{\text{new}}$) will replace the current best X -of- N ($X\text{-of-}N_{\text{best}}$) only if the following condition is true.

$$\begin{aligned}
& (\text{gain ratio}(X\text{-of-}N_{\text{new}}) > \text{gain ratio}(X\text{-of-}N_{\text{best}}) \wedge \\
& \quad \text{coding cost}(X\text{-of-}N_{\text{new}}) \leq \text{coding cost}(X\text{-of-}N_{\text{best}})) \vee \\
& (\text{gain ratio}(X\text{-of-}N_{\text{new}}) = \text{gain ratio}(X\text{-of-}N_{\text{best}}) \wedge \\
& \quad \text{coding cost}(X\text{-of-}N_{\text{new}}) < \text{coding cost}(X\text{-of-}N_{\text{best}}))
\end{aligned}$$

With this condition, the algorithm accepts a new attribute with a higher gain ratio value if its coding cost is no higher. The algorithm also accepts a new attribute with the same gain ratio value but with a lower coding cost value.⁷

The search for $X\text{-of-}N$ representations ceases when no further attribute-value pairs can be added. To reduce the search time, another restriction is applied: If no better new attribute has been found in five consecutive search steps,⁸ the algorithm terminates.

3.3. Pre-processing

As mentioned in Section 2, $X\text{-of-}N$ representations are constructed directly from binary and nominal attributes. To deal with primitive numeric attributes, the XofN algorithm uses a pre-process that discretizes primitive numeric attributes. In the current implementation, we use a very simple method, although some better, but more complex, methods can be used (see Section 6 for a discussion on this issue). When there are some primitive numeric attributes, XofN runs C4.5 once on all the primitive attributes, including binary, nominal, and numeric attributes, to generate a pruned tree. Cut points for numeric attributes are extracted from decision nodes of the pruned tree where the primitive numeric attributes are used. XofN, then, discretizes the numeric attributes using the cut points. The new attribute construction is carried out on the discretized attributes, primitive binary attributes, and primitive nominal attributes. This method has another effect. That is, C4.5 is used to select primitive numeric attributes since only those numeric attributes that appear in the pruned tree are discretized and passed to the process of new attribute generation.

3.4. Post-processing

In some domains with a large number of irrelevant primitive attributes (John, Kohavi, & Pfleger, 1994), XofN may not find good $X\text{-of-}N$ representations at the beginning when building a tree. Instead, it may find some $X\text{-of-}N$ representations containing irrelevant attributes or just chooses some irrelevant primitive attributes. This is because these irrelevant primitive attributes or new attributes containing irrelevant attributes by chance have high information gain ratio values on the training set. However, at some nodes after some splits have been done, good $X\text{-of-}N$ attributes may be created.

To solve this problem, the XofN algorithm uses a very simple method, although several other attribute selection approaches (Almuallim & Dietterich, 1992; Kira & Rendell, 1992; John, Kohavi, & Pfleger, 1994; Caruana & Freitag, 1994; Langley, 1994; Langley & Sage, 1994; Moore & Lee, 1994; Skalak, 1994) could also be used. It performs selective tree learning by using C4.5 a second time to build the final tree by using only those primitive and new attributes contained in the pruned tree that is built in the kernel of XofN discussed above. The attributes appearing in the tree are treated as relevant attributes; others are taken

Table 5. Error rates (%) in the Parity5 domain. The final tree built by XofN is correct at every trial, while the tree built in the kernel of the XofN algorithm is not correct at one of the ten trials.

Trial	C4.5	XofN (Kernel)	XofN (Final)
0	47.2	0.0	0.0
1	51.1	25.0	0.0
2	50.9	0.0	0.0
3	49.3	0.0	0.0
4	32.8	0.0	0.0
5	45.8	0.0	0.0
6	52.8	0.0	0.0
7	50.3	0.0	0.0
8	49.6	0.0	0.0
9	48.1	0.0	0.0
Mean	47.8	2.5	0.0

as irrelevant and are deleted. No new attribute construction is involved in the final phase. The following example gives a simple demonstration using an artificial domain.

Example 3 (Parity5). Parity5 from Pagallo (1990) has 32 binary attributes. Five of them are relevant. A random data generator is used to create training sets and test sets. Ten trials have been conducted, each with a different training set of size 4000 and an independent test set of size 2000. The error rates are summarized in Table 5. The results of C4.5 are included as a reference. XofN (Kernel) refers to the pruned tree built in the kernel of the XofN algorithm, while XofN (Final) refers to the pruned tree built in the post-processing part.

From the table, we can see that XofN (Kernel) fails to solve the problem at one out of the ten trials. At this trial, the tree has 873 nodes. Its root is an irrelevant primitive attribute. In one subtree of the root, which is created first, many irrelevant primitive attributes and five new attributes containing irrelevant attributes are used. In the other subtree of the root, the appropriate X -of- N attribute is generated and used. Therefore, XofN (Final) solves the problem with the appropriate X -of- N attribute.

Another purpose of the post-process is to alleviate the following problem. Since the XofN algorithm considers reusing new X -of- N attributes constructed previously at other decision nodes when building a decision tree, the sequence in which decision nodes and the corresponding new attributes are created might affect the performance of the algorithm. For example, if the local training set at a decision node is small, the algorithm may not construct a good X -of- N attribute. In this case, if a good X -of- N attribute has already been created at another decision node, the algorithm can reuse it as a test for the current decision node and build a good subtree. Otherwise, if the order of exploring these two nodes is reversed, no good X -of- N attribute can be reused for this decision node.

The current implementation of the algorithm builds (by using C4.5) subtrees under a decision node in the natural order of the outcomes of the test⁹ at the decision node. XofN uses the post-process described above to alleviate the decision node generation order problem. During the growth of the tree in the post-process, all the X -of- N attributes are available for building each decision node.

An alternative solution to the decision node generation order problem is, as Ross Quinlan has suggested,¹⁰ to build subtrees under a decision node in descending order of the sizes of training subsets going to the subtrees. In such a way, subtrees with larger local training sets are explored first. Since large local training sets are good for constructing appropriate X -of- N attributes at decision nodes, good X -of- N attributes are more likely to be constructed earlier, thus being able to be reused at decision nodes with small local training sets. Therefore, this method may improve the performance of the algorithm and is worthy of future investigation.

In summary, the XofN algorithm contains three parts: The pre-processing part, the kernel, and the post-processing part. The pre-processing part is for discretizing primitive numeric attributes when necessary. The construction of new X -of- N attributes is carried out in the kernel during the growth of a tree. At each decision node, it creates one X -of- N attribute by performing heuristic search. The post-processing part conducts selective induction and builds another tree using X -of- N attributes generated in the kernel and primitive attributes. Figure 4 shows the skeleton of the whole XofN algorithm. All the trees built in these three parts are pruned using the pruning mechanism of C4.5 (Quinlan, 1993). The pruned tree generated in the post-processing part will be used when measuring the prediction accuracy and theory complexity of the XofN algorithm later on in this paper.

4. Experimental evaluation of the XofN algorithm

The previous section has described the XofN algorithm that constructs nominal attributes in the form of X -of- N representations for decision tree learning. This section empirically evaluates XofN and compares it with other constructive decision tree learning algorithms, namely SFRINGE, CI3, CAT, and ID2-of-3. C4.5 is used as the baseline for the comparisons since most of these constructive induction algorithms use it as their selective induction component. SFRINGE is our implementation of the FRINGE algorithm (Pagallo, 1990) with extensions (Zheng, 1996). It follows the idea of SYMFINGE (Yang, Rendell, & Blix, 1991). For each leaf, SFRINGE constructs one new attribute using the conjunction of two conditions at the parent and grandparent nodes of the leaf. CI3 (Zheng, 1992, 1996) and CAT (Zheng, 1998) are also constructive decision tree learning algorithms. CI3 creates new attributes from production rules that are transformed from a decision tree. For each rule, it uses the conjunction of two conditions near the root of the tree as a new attribute (default option setting of the algorithm). Instead of using fixed numbers of conditions from fixed positions in a path of a tree as SFRINGE does, CAT searches for conditions to form a conjunction as a new attribute from a path by carrying out systematic search with pruning over conditions of the path. Both CI3 and CAT try to filter out irrelevant conditions from new attributes (Zheng, 1996, 1998). While SFRINGE, CI3, and CAT use conjunction and negation (implicitly) as constructive operators, ID2-of-3 (Murphy & Pazzani, 1991) uses M -of- N as its constructive

Figure 4. Skeleton of the XofN algorithm.

operator. This is the work most closely related to the XofN algorithm proposed in this paper. At each decision node when building a decision tree, ID2-of-3 constructs one at-least M -of- N as a test. All of these four constructive decision tree learning algorithms generate new binary attributes.

The main performance metrics for our learning algorithms are prediction accuracy and theory complexity. We expect learned theories to be highly accurate on unseen cases. In addition, we prefer simple theories, since complex theories are usually difficult for humans to understand and have high computational requirements when being used to classify cases. While many studies used the number of decision nodes or the number of all nodes in a decision tree as the complexity measure of decision trees (Matheus & Rendell, 1989; Pagallo & Haussler, 1990; Murphy & Pazzani, 1991), in this paper, we use a modified tree size as the theory complexity. It is the sum of the sizes of all the nodes, including leaves, of a tree. The size of a leaf is 1. The size of a decision node is 1 for a univariate tree, and is the number of attribute-value pairs, or conditions, in the test of the node for a multivariate tree. The modified tree size is a fair measure of the theory complexity when comparisons involve both selective and constructive induction algorithms. The reason is that decision nodes in trees created by constructive induction algorithms are more complex than those by selective induction algorithms. The modified tree size takes account of the complexity difference between these two types of decision nodes. However, neither the number of decision nodes nor the number of all nodes in a tree reflects this complexity difference.

We conduct three experiments in a set of artificial and natural domains. Each of them tests one of our expectations about the behavior of X -of- N representations as nominal attributes. At the end of this section, the computational requirements of the XofN algorithm are briefly addressed.

4.1. *Experimental domains and methods*

Twenty-seven domains, in total, are used for conducting empirical studies. Among them, seventeen are artificial domains and the rest are natural domains. Fourteen out of the seventeen artificial domains are logical domains from Pagallo (1990). The others are the three Monks problems from Thrun et al. (1991). The ten natural domains are from the UCI repository of machine learning databases (Blake, Keogh & Merz, 1999).

Table 6 summarizes the characteristics of the fourteen artificial logical domains (Pagallo, 1990). Each CNF is the dual concept of the corresponding DNF. For CNF concepts, columns “No. of terms” and “Term length” give the number of disjunctions and disjunction length respectively. These domains cover a variety of well-studied artificial logical concepts in the machine learning community: Randomly generated boolean concepts including DNF and CNF concepts, multiplexor concepts, parity concepts, and majority concepts. We use the same experimental method as given by Pagallo (1990), including the sizes of training and test sets. The sizes of test sets are always 2000 for all of these logical domains. The sizes of training sets are listed in Table 6. For each experiment, a training set and a test set are independently drawn from the uniform distribution. Experiments are repeated ten times in each of these domains.

The three Monks domains (Thrun et al., 1991) are also chosen because they have been studied previously by many other researchers. There are published results for more than twenty different learning algorithms in these domains. They represent three different types of learning task with binary and nominal attributes.

Table 6. Characteristics of the artificial logical domains.

Domain name	Description	No. of terms	Term length			No. of att.	Training set size
			Min.	Max.	Average		
DNF1	random DNF	9	5	6	5.8	80	3292
DNF2	random DNF	8	4	7	4.5	40	2185
DNF3	random DNF	6	4	7	5.5	32	1650
DNF4	random DNF	10	3	5	4.1	64	2640
CNF1	random CNF	9	5	6	5.8	80	3292
CNF2	random CNF	8	4	7	4.5	40	2185
CNF3	random CNF	6	4	7	5.5	32	1650
CNF4	random CNF	10	3	5	4.1	64	2640
MX6	6-multiplexor	4	3	3	3.0	16	720
MX11	11-multiplexor	8	4	4	4.0	32	1600
Parity4	4-parity	8	4	4	4.0	16	1280
Parity5	5-parity	16	5	5	5.0	32	4000
Maj11	11-majority	462	6	6	6.0	32	3000
Maj13	13-majority	1716	7	7	7.0	64	3000

Given the six attributes and their respective values:¹¹

$$\begin{array}{lll}
 A_1: 1, 2, 3 & A_2: 1, 2, 3 & A_3: 1, 2 \\
 A_4: 1, 2, 3 & A_5: 1, 2, 3, 4 & A_6: 1, 2
 \end{array}$$

the target concept of Monks1 is $(A_1 = A_2) \vee (A_5 = 1)$. This is a disjunctive concept, but it needs an extended zeroth-order language (adding equality relations between pairs of attributes) to be represented concisely. The target concept of Monks2 is “exactly two of the six attributes have their first value”. It is an “exactly M -of- N ” concept with nominal attributes. Monks3’s concept is $(A_5 = 3 \wedge A_4 = 1) \vee (A_5 \neq 4 \wedge A_2 \neq 3)$. It is a DNF concept, but to represent it concisely, the negation operator on attribute-value pairs is needed. Monks1 and Monks3 have irrelevant attributes. In each of the three domains, the fixed training set (a subset of the whole dataset) and test set (the whole dataset) are given by the problem designers (Thrun et al., 1991). The training set sizes are 124, 169, and 122 for Monks1, Monks2, and Monks3 respectively. The test set size is 432 for each of them. There is no noise in the test sets of the three domains. Only Monks3 has 5% classification noise in its training set.¹² In the Monks domains, since the fixed training set and test set are given for each problem by the problem designers, we follow this methodology and run experiments once on the given training set and test set for each domain.

The ten natural domains consist of five medical domains (Cleveland heart disease, Hepatitis, Liver disorders, Pima Indians diabetes, Wisconsin breast cancer), one molecular biology domain (Promoters), three linguistics domains (Nettalk-phoneme, Nettalk-stress, Nettalk-letter), and one game domain (Tic-tac-toe). For the three Nettalk domains, we use the 1000 most common English words containing 5438 letters. In each of these ten domains,

Table 7. Description of the ten domains from UCI.

Domain	Size	No. of Attributes				No. of classes	Default accuracy (%)
		<i>B</i>	<i>N</i>	<i>C</i>	<i>T</i>		
Cleveland heart disease	303	0	0	13	13	2	54.1
Hepatitis	155	13	0	6	19	2	79.4
Liver disorders	345	0	0	6	6	2	58.0
Diabetes	768	0	0	8	8	2	65.1
Wisconsin breast cancer	699	0	0	9	9	2	65.5
Promoters	106	0	57	0	57	2	50.0
Nettalk-phoneme	5438	0	7	0	7	52	18.7
Nettalk-stress	5438	0	7	0	7	5	40.1
Nettalk-letter	5438	0	7	0	7	163	11.2
Tic-tac-toe	958	0	9	0	9	2	65.3

a 10-fold cross-validation (Breiman et al., 1984) is conducted on the entire data set. Table 7 gives a brief summary of the ten domains, including the dataset size, the number of binary (*B*), nominal (*N*), numeric (*C*) attributes, the total (*T*) number of attributes, the number of classes, and the default accuracy (the relative frequency of the most common class). The ten domains cover the spectrum of properties such as dataset size, attribute types and numbers, the number of different nominal attribute values, and the number of classes. In addition, *M*-of-*N*-like concepts are expected to be found in some of these domains (Spackman, 1988). The objective of using a test suite with this property is to test whether the algorithms capable of learning this kind of concept can work well in some real-world applications.

In all the experiments reported throughout this paper, all the algorithms are run with their default option settings. For the XofN algorithm, the whole algorithm including the pre-process and the post-process as illustrated in figure 4 is always used.¹³ To compare prediction accuracies and theory complexities of two algorithms, a two-tailed block-based pairwise t-test is conducted. In the Monks domains, because only one block is available for each problem, a two-tailed instance-based pairwise sign-test is used for comparing prediction accuracies. No significance test can be performed when comparing theory complexities in the Monks domains, since only one complexity value for each algorithm in each domain is available. A difference is considered as significant if the significance level of the t-test or sign-test is better than 0.05. In tables, $\oplus$ indicates that the prediction accuracy or theory complexity of an algorithm is significantly better than that of XofN. $\ominus$ signifies that the accuracy or complexity of an algorithm is significantly worse than that of XofN.

4.2. Comparison with C4.5

If domains involve concepts such as conjunctive, disjunctive, *M*-of-*N*-like, and parity-like concepts, nominal *X*-of-*N* representations should be able to simplify the tree representations of the target concepts in these domains. Therefore, XofN is expected to improve the

performance of selective decision tree learning significantly in these kinds of domain in terms of both higher prediction accuracy and lower theory complexity. It is known that interesting concepts such as DNF, CNF, majority, and parity concepts that are often studied in the machine learning community belong to these types of domain. Conjunctions and disjunctions are often used by humans to represent knowledge and appear in many real-world domains. M -of- N -like concepts are used in some real-world domains such as medical domains (Spackman, 1988). Although a single nominal X -of- N representation can represent more complex concepts than a primitive attribute, as discussed in Subsection 2.2 the former suffers from the fragmentation problem in domains that need many X -of- N representations whose few values are worth distinguishing for splitting examples of different classes. This might lead to a decline in the performance of XofN. Now, let us analyze the experimental results.

Table 8 shows the accuracies and theory complexities of C4.5 and XofN in the artificial logical, Monks, and natural domains. In all the parity and majority domains, XofN learns very good tree representations of the concepts. They have 100% prediction accuracy, and are very concise. The performance improvement over C4.5 in terms of both prediction accuracy and theory complexity is dramatic. However, in the DNF, CNF, and multiplexor domains, XofN achieves almost no improvement over C4.5. In the DNF2 and CNF2 domains, XofN even produces significant lower prediction accuracies than C4.5. The reason is that all these domains, except MX6 which is quite simple, need a large number of long X -of- N representations. Consequently, nominal X -of- N attributes suffer from the fragmentation problem in these domains. The next section will propose approaches to alleviating this problem and demonstrate their success.

As shown in Table 8, XofN solves all the three Monks problems with correct and simple tree representations. It finds a perfect representation for the target concept of the Monks2 problem.

In seven out of the ten natural domains, XofN achieves a significant improvement on prediction accuracy over C4.5. In the other three domains, the accuracy differences are not significant. The reason why XofN does not work well in these three domains might be that there are no appropriate X -of- N representations in them, or there are some, but XofN cannot find them due to the simple search strategy of the current implementation. For example, XofN performs quite well in the Cleveland heart disease domain. In this domain, the pruned trees contain, on average, 11.3 X -of- N attributes of size 1.6 over the ten trials. However, XofN performs similarly to (slightly better than) C4.5 in the Hepatitis domain where the pruned trees built by XofN use only, on average, 1.8 X -of- N attributes of size 1.4 over the ten trials.

As far as the theory complexity in the ten natural domains is concerned, XofN learns significantly less complex trees than C4.5 in five domains. Note that in these five domains, the prediction accuracies of XofN are also significantly higher than those of C4.5. Only in one domain, does XofN create significantly more complex trees than C4.5. Complexity differences in the other natural domains are not significant.

The results of this experiment provide evidence of our expectation. Except in the DNF, CNF, and multiplexor domains where XofN suffers from the fragmentation problem, constructing nominal X -of- N attributes can significantly improve the performance of decision

Table 8. Results of C4.5 and XofN. It is shown that XofN can significantly improve the performance of decision tree learning, but it suffers from the fragmentation problem in the DNF, CNF, and multiplexor domains. XofN learns correct tree representations of the three Monks problems. In most of the natural domains, it can significantly increase the prediction accuracies and reduce the theory complexities of decision tree learning.

Domain	Accuracy(%)		Complexity	
	C4.5	XofN	C4.5	XofN
DNF1	87.2	86.1	263.0	300.6
DNF2	$\oplus 90.7$	87.7	202.0	234.0
DNF3	93.8	93.1	101.8	88.0
DNF4	74.0	74.5	$\oplus 525.0$	593.2
CNF1	86.9	84.6	271.2	314.2
CNF2	$\oplus 90.6$	87.1	192.8	227.0
CNF3	93.3	93.2	94.4	90.0
CNF4	72.9	73.5	$\oplus 532.8$	607.8
MX6	100.0	100.0	$\oplus 48.6$	60.6
MX11	97.2	96.6	$\oplus 168.6$	243.6
Parity4	$\ominus 67.5$	100.0	$\ominus 238.4$	9.0
Parity5	$\ominus 52.2$	100.0	$\ominus 1339.4$	13.4
Maj11	$\ominus 82.9$	100.0	$\ominus 461.6$	23.0
Maj13	$\ominus 76.3$	100.0	$\ominus 527.4$	27.0
Monks1	$\ominus 75.7$	100.0	18.0	17.0
Monks2	$\ominus 65.0$	100.0	31.0	13.0
Monks3	$\ominus 97.2$	100.0	12.0	9.0
Cleveland	$\ominus 73.3$	79.8	49.8	41.1
Hepatitis	78.2	79.4	13.6	13.4
Liver	$\ominus 62.1$	70.1	79.4	89.1
Diabetes	71.5	70.8	128.8	153.6
Wisconsin	94.8	94.9	$\oplus 20.6$	26.3
Promoters	$\ominus 76.3$	88.5	$\ominus 22.6$	13.9
Nettalk-p	$\ominus 81.1$	83.9	$\ominus 2339.2$	1506.0
Nettalk-s	$\ominus 82.7$	87.6	$\ominus 2077.3$	739.6
Nettalk-l	$\ominus 73.7$	76.9	$\ominus 3394.9$	2242.4
Tic-tac-toe	$\ominus 84.7$	98.4	$\ominus 128.5$	42.8

tree learning in terms of both higher prediction accuracy and lower theory complexity in most domains tested. We have not found that nominal X -of- N attributes significantly suffer from the fragmentation problem in the natural domains from the UCI repository of machine learning databases (Blake, Keogh & Merz, 1999) under investigation. This may suggest that the natural domains from UCI are not so complex. “Complex”, here, means that many large X -of- N representations are needed to represent target concepts.

4.3. Comparison with other constructive decision tree learning algorithms

Subsection 2.1 has shown that a single X -of- N representation can directly and simply represent any concept that can be represented by a single conjunctive, a single disjunctive, or a single M -of- N representation, and that the reverse is not true. Therefore, we expect XofN to achieve significantly higher accuracy than constructive induction algorithms that construct, as new binary attributes, conjunctive, disjunctive, or M -of- N representations in some natural and artificial domains, especially in domains where M -of- N -like or parity-like concepts are involved.

Despite the advantages of X -of- N s in concept representation over conjunctions and disjunctions, XofN cannot be expected to perform better than the tree learning algorithms that construct conjunctions or disjunctions as new attributes on DNF and CNF concepts. The reason is that DNF and CNF concepts exactly fit the bias of these algorithms. What we can expect is that XofN performs as well as these algorithms when learning this type of concept if it does not suffer from the fragmentation problem or it works with a technique for alleviating this problem. Similarly, XofN should perform as well as the tree learning algorithms that construct M -of- N attributes in domains involving M -of- N concepts.

Table 9 presents the prediction accuracies and theory complexities of SFRINGE, CI3, CAT, ID2-of-3, and XofN in the artificial logical domains. XofN demonstrates performance advantages over SFRINGE, CI3, and CAT in the parity and majority domains, and performance advantages over ID2-of-3 in the parity domains. XofN achieves significantly higher

Table 9. Accuracies (%) and theory complexities of SFRINGE, CI3, CAT, ID2-of-3, and XofN in the artificial logical domains. XofN demonstrates its performance advantage over SFRINGE, CI3, and CAT in the parity and majority domains, as well as its performance advantage over ID2-of-3 in the parity domains.

Domain	Accuracy(%)					Complexity				
	SFRINGE	CI3	CAT	ID2-of-3	XofN	SFRINGE	CI3	CAT	ID2-of-3	XofN
DNF1	⊕97.1	⊕100.0	⊕99.4	⊕93.3	86.1	⊕125.3	⊕60.7	⊕69.1	247.5	300.6
DNF2	⊕99.6	⊕99.9	⊕99.2	⊕97.6	87.7	⊕45.5	⊕70.6	⊕47.5	⊕107.6	234.0
DNF3	⊕99.5	⊕99.9	⊕99.4	⊕97.8	93.1	⊕37.9	⊕43.4	⊕45.2	85.5	88.0
DNF4	⊕100.0	⊕100.0	⊕99.2	⊕98.7	74.5	⊕48.2	⊕43.7	2480.0	⊕90.6	593.2
CNF1	⊕96.5	⊕100.0	⊕99.4	⊕93.1	84.6	⊕140.6	⊕55.3	⊕61.2	266.5	314.2
CNF2	⊕99.3	⊕99.5	⊕99.2	⊕97.9	87.1	⊕36.0	⊕62.3	⊕34.0	⊕104.6	227.0
CNF3	⊕99.7	⊕99.7	⊕99.3	⊕98.8	93.2	⊕39.9	⊕50.8	⊕30.5	70.0	90.0
CNF4	⊕99.4	⊕100.0	⊕99.1	⊕96.7	73.5	⊕71.6	⊕44.0	⊕98.5	⊕148.4	607.8
MX6	100.0	100.0	100.0	99.8	100.0	⊕15.8	⊕15.4	⊕14.0	⊕30.2	60.6
MX11	⊕100.0	⊕100.0	⊕100.0	97.6	96.6	⊕55.1	⊕76.1	⊕34.0	⊕112.1	243.6
Parity4	100.0	100.0	99.8	⊕89.9	100.0	⊕38.4	⊕47.6	⊕42.9	⊕225.7	9.0
Parity5	87.3	⊕74.7	⊕65.6	⊕59.2	100.0	⊕445.2	⊕351.3	⊕2022.9	⊕1585.0	13.4
Maj11	⊕92.7	⊕92.8	⊕92.5	100.0	100.0	⊕725.1	⊕1076.7	⊕1538.9	⊕13.0	23.0
Maj13	⊕82.7	⊕86.5	⊕85.7	100.0	100.0	⊕692.6	⊕1496.7	⊕916.5	⊕15.0	27.0

accuracies than SFRINGE in the Maj11 and Maj13 domains,¹⁴ than CI3 and CAT in the Parity5, Maj11, and Maj13 domains, than ID2-of-3 in the Parity4 and Parity5 domains. In terms of theory complexity, XofN creates much less complex trees than SFRINGE, CI3, and CAT in the Parity4, Parity5, Maj11, and Maj13 domains, and creates much less complex trees than ID2-of-3 in the Parity4 and Parity5 domains. All these reductions in theory complexity are significant. In the Maj11 and Maj13 domains, XofN generates significantly more complex trees than ID2-of-3. Actually, XofN and ID2-of-3 construct new attributes containing the same attribute-value pairs in these two domains. Since XofN treats these new attributes as nominal attributes while ID2-of-3 treats them as binary attributes, XofN creates more leaves than ID2-of-3. Consequently, XofN builds more complex trees than ID2-of-3 in these two domains. We will see that XofN with some techniques for alleviating the fragmentation problem can generate trees of the same complexity as ID2-of-3 in the majority domains.

As discussed before, XofN suffers from the fragmentation problem in the DNF, CNF, and multiplexor domains. This results in significantly worse prediction accuracies and theory complexities of XofN than those of SFRINGE, CI3, CAT, and ID2-of-3 in most of these ten DNF, CNF, and multiplexor domains. We will show, in the next section, that XofN with the techniques for alleviating the fragmentation problem performs similarly to or better than these four algorithms in these ten domains.

In the Monks domains as shown in Table 10, only XofN achieves 100% accuracies for all the three problems. Furthermore, it learns the smallest tree among these algorithms in the Monks2 domain. In the Monks3 domain, it learns a smaller tree than all other algorithms except CAT.

To make the Monks2 problem harder, especially for simple M -of- N learning methods, Bloedorn, Michalski, and Wnek (1993) create the “Noisy and Irrelevant Monks2” problem by adding 5% random classification noise (by inverting the classes) in the training set, and adding seven random five-value irrelevant attributes into both training and test sets. Like the three original Monks domains, there is no classification noise in the test set of the Noisy and Irrelevant Monks2 domain. Table 11 gives our results of C4.5, SFRINGE, CI3, CAT, ID2-of-3, and XofN, as well as the results of AQ17-DCI, AQ17-HCI, and AQ17-MCI from Bloedorn, Michalski, and Wnek (1993).¹⁵ Only XofN learns a correct concept representation in this domain. Because of noise, the learned tree is not the perfect representation. Instead, XofN finds two new attributes X -of- $\{A_4 = 1\}$ and X -of- $\{A_1 = 1, A_2 = 1, A_3 = 1, A_5 = 1, A_6 = 1\}$. However, it is still the most concise representation among those learned by these

Table 10. Accuracies (%) and theory complexities of SFRINGE, CI3, CAT, ID2-of-3, and XofN in the Monks domains. Only XofN correctly solves all the three Monks problems.

Domain	Accuracy(%)					Complexity				
	SFRINGE	CI3	CAT	ID2-of-3	XofN	SFRINGE	CI3	CAT	ID2-of-3	XofNcr
Monks1	100.0	100.0	100.0	100.0	100.0	11.0	12.0	9.0	18.0	17.0
Monks2	64.1	67.1	75.9	98.1	100.0	13.0	22.0	73.0	24.0	13.0
Monks3	97.2	95.8	100.0	97.2	100.0	14.0	12.0	6.0	21.0	9.0

Table 11. In the Noisy and Irrelevant Monks2 domain. Only XofN generates a correct tree representation in this domain.

Algorithm	Accuracy (%)	Complexity
C4.5	⊖67.1	1.0
AQ17-DCI	⊖81.5	139.0
AQ17-HCI	⊖42.1	68.0
AQ17-MCI	⊖90.2	31.0
SFRINGE	⊖67.1	1.0
CI3	⊖67.1	1.0
CAT	⊖67.1	1.0
ID2-of-3	⊖55.6	67.0
XofN	100.0	15.0

Table 12. Accuracies (%) and theory complexities of SFRINGE, CI3, CAT, ID2-of-3, and XofN in the natural domains. The overall performance of XofN is better than that of the other constructive decision tree learning algorithms in terms of higher prediction accuracy.

Domain	Accuracy(%)					Complexity				
	SFRINGE	CI3	CAT	ID2-of-3	XofN	SFRINGE	CI3	CAT	ID2-of-3	XofN
Cleveland	75.9	75.5	75.9	76.8	79.8	45.6	⊕20.5	⊖81.1	⊖62.2	41.1
Hepatitis	80.0	82.7	82.0	77.0	79.4	10.8	14.0	12.2	⊖24.6	13.4
Liver	64.9	⊖63.4	67.0	⊖63.2	70.1	75.4	⊕44.3	⊖152.7	⊖108.9	89.1
Diabetes	71.5	73.7	73.0	69.2	70.8	128.8	⊕25.3	⊕64.5	⊖191.4	153.6
Wisconsin	95.4	96.0	95.0	94.4	94.9	⊕14.9	⊕14.1	28.3	⊖37.3	26.3
Promoters	⊖78.1	82.9	86.9	87.6	88.5	14.0	11.8	11.9	11.2	13.9
Nettalk-p	83.7	⊖82.4	83.5	83.1	83.9	⊕1176.7	⊖1615.9	⊕876.1	⊕1188.4	1506.0
Nettalk-s	⊖85.8	⊖86.2	88.0	⊖86.2	87.6	⊖858.6	812.9	⊕372.4	⊖961.5	739.6
Nettalk-l	77.8	⊖66.8	⊖74.6	⊖75.1	76.9	⊕1821.5	⊕1172.5	⊕762.0	⊕1654.8	2242.4
Tic-tac-toe	97.6	98.4	98.3	⊖94.9	98.4	69.3	31.9	26.0	⊖95.8	42.8

algorithms except for C4.5, SFRINGE, CI3, and CAT which return a tree having only one leaf. This illustrates that XofN can, to some extent, tolerate irrelevant attributes in conjunction with noise, but this matter remains to be explored further.

Now, we compare XofN with SFRINGE, CI3, CAT, and ID2-of-3 in the ten natural domains. Table 12 presents the prediction accuracies and theory complexities of these algorithms. XofN is significantly more accurate than SFRINGE, CI3, CAT, and ID2-of-3 in two, four, one, and four domains respectively. None of SFRINGE, CI3, CAT, and ID2-of-3 obtains significantly higher accuracies than XofN in any of these domains. As far as the theory complexity is concerned, XofN generates significantly less complex trees than SFRINGE, CI3, CAT, and ID2-of-3 in one, one, two, and seven out of the ten domains respectively. It

creates significantly more complex trees than SFRINGE, CI3, CAT, and ID2-of-3 in three, five, four, and two domains respectively.

It is worth mentioning that the post-process, as a part of the XofN algorithm, is helpful for increasing the accuracy in some domains. For example, the post-process increases the accuracy of XofN in four out of the ten natural domains, very slightly reduces the accuracy in two domains, and does not affect the accuracy in the other four domains. On average over these ten natural domains, XofN with the post-process is 0.67 percentage points more accurate than without it. However, XofN without the post-process is still more accurate than all the other algorithms studied in this paper on average over the ten natural domains. In addition, we did experiments by replacing the construction of X -of- N attributes in the XofN algorithm with the construction of conjunctive, disjunctive, or M -of- N attributes (Zheng, 1996). All other parts of the algorithm, including the pre-process and the post-process, are kept exactly the same. The experimental results (Zheng, 1996) show the advantages of constructing X -of- N attributes over constructing conjunctive, disjunctive, or M -of- N attributes for decision tree learning in terms of higher prediction accuracy and lower theory complexity.

In summary, XofN is significantly better than SFRINGE, CI3, and CAT in the majority, parity, and Monks domains in terms of both higher prediction accuracy and lower theory complexity. It is significantly better than ID2-of-3 in the parity and Monks domains in terms of both higher prediction accuracy and lower theory complexity. In the DNF, CNF, and multiplexor domains, XofN is worse than the other constructive decision tree induction algorithms since it suffers from the fragmentation problem. In the natural domains, XofN more frequently generates significantly less complex trees than ID2-of-3, but it less frequently builds significantly less complex trees than SFRINGE, CI3, and CAT. In some natural domains, XofN achieves significantly higher accuracies than SFRINGE, CI3, CAT, and ID2-of-3. It has not built any significantly less accurate trees than any of these algorithms in the natural domains under investigation.

4.4. Learning curves

Having studied the prediction accuracies and theory complexities of XofN using fixed-sized training sets, we move to investigate, using learning curves, the scaling up characteristic of XofN. Here, only two domains, Tic-tac-toe and Nettealk-stress which have relatively large datasets, are used due to the space limit. The reference algorithms are C4.5 and ID2-of-3. Figure 5 shows the prediction accuracy and theory complexity learning curves of these algorithms. Each point of a learning curve is an average value over ten trials. A bar in the figures indicates one standard error on each side of a curve. For each trial, the training set used at every point is a randomly selected subset of the training set used at the corresponding trial of the 10-fold cross-validation on the entire dataset of the domain. At each trial, the training set at a point is a proper subset of the training set at the next adjacent point. The test set at every point of a trial is the same as the test set used at the corresponding trial of the 10-fold cross-validation.

The figures illustrate the clear advantages of XofN over both C4.5 and ID2-of-3 in terms of higher prediction accuracy and lower theory complexity. The accuracy of XofN grows

Figure 5. Learning curves in the Tic-tac-toe and Nettetalk-stress domains. XofN is more accurate than both C4.5 and ID2-of-3 for all the training set sizes, and less complex than C4.5 and ID2-of-3 for most training set sizes in these two domains.

faster than that of C4.5 and ID2-of-3 when the training set size increases in both domains. The theory complexity of XofN grows much more slowly than that of C4.5 and more slowly than ID2-of-3 in the Nettetalk-stress domain. In the Tic-tac-toe domain, the theory complexity of XofN is close to that of C4.5 and ID2-of-3 when the training set size is less than 500, but XofN generates less complex trees than C4.5 and ID2-of-3 when the training set size exceeds 500. The complexity of XofN has a sharp drop after the training set size 500, since XofN constructs appropriate new attributes with training sets of more than 500 examples in the Tic-tac-toe domain. This results in more compact decision trees.

4.5. Computational requirements of XofN

The execution time of XofN depends on the number of decision nodes in a tree that XofN builds and the time requirements for constructing one *X-of-N* attribute at each decision

Figure 6. Execution time of XofN, CAT, and C4.5 (CPU seconds on a DEC AXP 3000/500 workstation) in the Tic-tac-toe and Nettalk-stress domains.

node. Since the greedy search with two operators, adding and deleting one attribute-value pair, makes it possible for XofN to create an $X\text{-of-}N$ attribute containing any possible combination of attribute-value pairs. Therefore, the worst case computational complexity of constructing one $X\text{-of-}N$ attribute at a decision node is $O(n \cdot 2^m)$ for n local training examples at that node and m possible attribute-value pairs. It is linear in the size of the local training set at the decision node, and is exponential on the number of attribute-value pairs. Nevertheless, in practice, it is unlikely for the algorithm to search through the whole search space defined by attribute-value pairs, which results in the exponential part. The reason is that the search proceeds greedily. At each search step, the algorithm accepts the best one in terms of the new attribute selection criterion among all the new attributes that can be created by adding or deleting one attribute-value pair. In addition, the search ceases when no better new attribute has been created in five consecutive search steps by default.

Now, we report experimental results to show the computational requirements of XofN in practice. The timing results of XofN in the Tic-tac-toe and Nettalk-stress domains are depicted in figure 6. They are CPU seconds on a DEC AXP 3000/500 workstation. C4.5 and CAT are used as references. ID2-of-3 is most similar to XofN, so it should be used for comparison. However, since ID2-of-3 is relatively inefficiently implemented, their timing results are not comparable.

XofN is much slower than C4.5, since it constructs a new attribute using search at each decision node, while C4.5 only chooses a primitive attribute. However, the execution time of XofN increases linearly when the training set size increases in both domains. This is acceptable. Compared with CAT, XofN and CAT have a similar trend in the Tic-tac-toe domain, while the execution time of CAT grows faster than that of XofN in the Nettalk-stress domain.

5. Solutions to the fragmentation problem of nominal $X\text{-of-}N$ attributes

It has been pointed out in Subsection 2.2 and illustrated in the previous section that, for some complex learning tasks, large $X\text{-of-}N$ representations as nominal attributes may suffer from

the fragmentation problem. For example, when X -of- N representations are used to learn a CNF concept with many long disjunctions, only the value 0 of each X -of- N representation that represents a disjunction of the CNF concept is worth discriminating from all other values because only the value 0 identifies a subset of negative examples. Generating a different branch for each of the other values does not help to separate examples of different classes. However, this does speed up splitting a training set into a large number of small subsets. In such a case, XofN has difficulty constructing all appropriate new attributes. Therefore, it may not be able to build a correct tree even though decision trees with X -of- N representations as nominal attributes can, in principle, represent CNF concepts. In this section, we present three approaches to alleviating this problem, and use experiments to illustrate their efficacy.

5.1. *Subsetting and subranging*

One approach to alleviating the fragmentation problem of nominal X -of- N s is the *subsetting* mechanism of C4.5 (Quinlan, 1993). For a nominal X -of- N attribute, XofN with subsetting groups all possible values of the attribute into a variable number of sets. Instead of individual values, the sets correspond to outcomes of the test. A greedy algorithm is used to find sets of attribute values. It starts with the initial value sets, one for each individual value of an X -of- N attribute. It, then, iteratively merges attribute value sets. In each iteration, subsetting evaluates the results of merging every pair of sets using the test evaluation function,¹⁶ and performs the best merger. The process stops when only two value sets remain or no merger creates a better partition of the training examples.

Another method of alleviating the fragmentation problem of nominal X -of- N s is *subranging*, which is very similar to and is inspired by subsetting. XofN with subranging uses the same method to generate subranges of the values of an X -of- N attribute as that used by XofN with subsetting for generating subsets of the values of an X -of- N attribute except that the former merges only adjacent values of an X -of- N attribute. When doing this, XofN with subranging utilizes the ordering information of the values of an X -of- N representation. This differs the subranging approach from the subsetting approach. From now, XofN with subsetting is referred to as XofN(s), and XofN with subranging is referred to as XofN(r).

After constructing an X -of- N representation at a decision node, XofN(s) and XofN(r) find the best subsets and the best subranges of the values of the X -of- N respectively to form a test. When subsetting or subranging is used with XofN, extra search is involved. The following example demonstrates the effect of subsetting and subranging on XofN by using an artificial concept.

Example 4 (Subsetting and subranging for alleviating the fragmentation problem). At one trial in the CNF4 domain, the tree built by XofN on a randomly generated training set of size 2640 has 565 nodes with a theory complexity of 667 and an error rate of 26.1% on an independent test set of size 2000. It is slightly more accurate than the tree built by C4.5 (error rate: 28.7%, size and theory complexity: 555), but has a higher theory complexity. The reason is that the training set of 2640 examples is small compared with the entire

universe consisting of 2^{64} different examples. Using such a small training set, XofN cannot construct ten appropriate X -of- N representations. Note that the target concept of CNF4 is a conjunction of 10 disjunctions of average length 4.1. In fact, XofN constructs only three out of the ten appropriate X -of- N representations near the root. After that, because the local training set at each decision node is too small, XofN does not create any other appropriate attributes.

However, when XofN(s) is run on the same training set, a tree with only 21 nodes is built. It is a correct representation of the target concept with no errors on the same test set. All ten appropriate X -of- N representations are constructed and used as tests at the ten decision nodes.

Similarly, XofN(r) also builds a correct representation of the CNF4 concept with 21 nodes on the same training set, with no errors on the same test set. It constructs the ten appropriate X -of- N attributes and uses them as tests at the ten decision nodes of the tree.

Our experiments¹⁷ in the 27 artificial and natural domains show that both subsetting and subranging can alleviate the fragmentation problem of nominal X -of- N attributes. There is no significant difference between the prediction accuracies of XofN(s) and XofN(r) in any of these domains. Only in one domain (Parity4), does XofN(r) build significantly more complex trees than XofN(s), and only in one domain (Nettalk-phoneme), does XofN(r) build significantly less complex trees than XofN(s). In all other domains, the theory complexity differences between XofN(r) and XofN(s) are not significant. Table 13 gives the accuracies and theory complexities of XofN(r) compared with those of C4.5 and XofN in the 27 domains. Since XofN performs poorly in the DNF, CNF, and multiplexor domains as mentioned before, the results of CAT and ID2-of-3 as two examples of other constructive induction algorithms are also included in this table for ease of comparison. These results are the same as those in Tables 9, 10, and 12. The accuracies and theory complexities of XofN(s) can be found in Appendix A.

From Table 13, we can see that XofN(r) learns significantly more accurate and less complex trees than both C4.5 and XofN in all the DNF, CNF, and multiplexor domains where XofN suffers from the fragmentation problem.¹⁸ Note that XofN(r) obtains very similar accuracies to those achieved by SFRINGE, CI3, and CAT in all these domains. Compared with ID2-of-3, XofN(r) learns more accurate and less complex trees in these domains. The accuracy improvement in six out of these ten DNF, CNF, and multiplexor domains is significant. The theory complexity decreases in seven out of these ten domains are significant. This illustrates the effects of subranging on alleviating the fragmentation problem of nominal X -of- N representations.

In the parity, majority, and Monks domains, like XofN, XofN(r) learns very good tree representations with 100% prediction accuracy, except for the Monks3 domain. The trees learned by XofN(r) are less complex than those learned by XofN in most of these domains because XofN(r) merges some outcomes of X -of- N attributes, thus reducing the number of branches of the decision trees. In the Monks3 domain, XofN and XofN(r) first construct a good new attribute X -of- $\{A_2 = 3, A_5 = 4\}$. If this X -of- N has the value 0, the concept has the value true. If the X -of- N has the value 2, the concept has the value false. If the X -of- N has the value 1, the truth value of the concept needs to be further decided. Due to the effect

Table 13. Accuracies (%) and theory complexities of C4.5, XofN, and XofN(r). XofN(r) refers to XofN with subranging. Since XofN performs poorly in the DNF, CNF, and multiplexor domains, the results of CAT and ID2-of-3 as two examples of other constructive induction algorithms are also included in this table for ease of comparison. These results are the same as those in Tables 9, 10, and 12. In this table, $\ominus$ ($\oplus$) indicates that C4.5, CAT, ID2-of-3, or XofN is significantly worse (better) than XofN(r). It is shown that subranging can effectively alleviate the fragmentation problem of nominal X -of- N representations.

Domain	Accuracy(%)					Complexity				
	C4.5	CAT	ID2-of-3	XofN	XofN(r)	C4.5	CAT	ID2-of-3	XofN	XofN(r)
DNF1	$\ominus 87.2$	99.4	$\ominus 93.3$	$\ominus 86.1$	97.9	$\ominus 263.0$	69.1	$\ominus 247.5$	$\ominus 300.6$	116.7
DNF2	$\ominus 90.7$	$\ominus 99.2$	$\ominus 97.6$	$\ominus 87.7$	99.6	$\ominus 202.0$	47.5	$\ominus 107.6$	$\ominus 234.0$	49.2
DNF3	$\ominus 93.8$	99.4	97.8	$\ominus 93.1$	99.1	$\ominus 101.8$	45.2	85.5	$\ominus 88.0$	48.4
DNF4	$\ominus 74.0$	99.2	98.7	$\ominus 74.5$	99.5	$\ominus 525.0$	2480.0	90.6	$\ominus 593.2$	56.8
CNF1	$\ominus 86.9$	99.4	$\ominus 93.1$	$\ominus 84.6$	99.5	$\ominus 271.2$	61.2	$\ominus 266.5$	$\ominus 314.2$	61.1
CNF2	$\ominus 90.6$	$\ominus 99.2$	$\ominus 97.9$	$\ominus 87.1$	99.4	$\ominus 192.8$	34.0	$\ominus 104.6$	$\ominus 227.0$	42.1
CNF3	$\ominus 93.3$	99.3	98.8	$\ominus 93.2$	99.6	$\ominus 94.4$	$\oplus 30.5$	$\ominus 70.0$	$\ominus 90.0$	36.1
CNF4	$\ominus 72.9$	$\ominus 99.1$	$\ominus 96.7$	$\ominus 73.5$	100.0	$\ominus 532.8$	98.5	$\ominus 148.4$	$\ominus 607.8$	52.7
MX6	100.0	100.0	99.8	100.0	100.0	$\ominus 48.6$	$\oplus 14.0$	30.2	$\ominus 60.6$	25.2
MX11	$\ominus 97.2$	100.0	$\ominus 97.6$	$\ominus 96.6$	100.0	$\ominus 168.6$	$\oplus 34.0$	$\ominus 112.1$	$\ominus 243.6$	54.4
Parity4	$\ominus 67.5$	99.8	$\ominus 89.9$	100.0	100.0	$\ominus 238.4$	$\ominus 42.9$	$\ominus 225.7$	9.0	9.0
Parity5	$\ominus 52.2$	$\ominus 65.6$	$\ominus 59.2$	100.0	100.0	$\ominus 1339.4$	$\ominus 2022.9$	$\ominus 1585.0$	13.4	14.4
Maj11	$\ominus 82.9$	$\ominus 92.5$	100.0	100.0	100.0	$\ominus 461.6$	$\ominus 1538.9$	13.0	$\ominus 23.0$	13.0
Maj13	$\ominus 76.3$	$\ominus 85.7$	100.0	100.0	100.0	$\ominus 527.4$	$\ominus 916.5$	15.0	$\ominus 27.0$	15.0
Monks1	$\ominus 75.7$	100.0	100.0	100.0	100.0	18.0	9.0	18.0	17.0	23.0
Monks2	$\ominus 65.0$	$\ominus 75.9$	$\ominus 98.1$	100.0	100.0	31.0	73.0	24.0	13.0	9.0
Monks3	97.2	$\oplus 100.0$	97.2	$\oplus 100.0$	97.2	12.0	6.0	21.0	9.0	4.0
Cleveland	$\ominus 73.3$	75.9	76.8	79.8	78.2	49.8	81.1	$\ominus 62.2$	41.1	42.0
Hepatitis	78.2	82.0	77.0	79.4	79.4	13.6	12.2	$\ominus 24.6$	13.4	13.3
Liver	62.1	67.0	63.2	70.1	66.1	79.4	152.7	108.9	89.1	95.9
Diabetes	71.5	73.0	69.2	70.8	71.7	$\oplus 128.8$	$\oplus 64.5$	191.4	$\oplus 153.6$	170.8
Wisconsin	94.8	95.0	94.4	94.9	95.1	20.6	28.3	$\ominus 37.3$	26.3	22.1
Promoters	76.3	86.9	87.6	88.5	83.5	$\ominus 22.6$	11.9	$\oplus 11.2$	13.9	14.3
Nettalk-p	$\ominus 81.1$	83.5	83.1	83.9	84.1	$\ominus 2339.2$	$\oplus 876.1$	$\oplus 1188.4$	1506.0	1483.9
Nettalk-s	$\ominus 82.7$	88.0	$\ominus 86.2$	87.6	87.6	$\ominus 2077.3$	$\oplus 372.4$	$\ominus 961.5$	$\oplus 739.6$	816.0
Nettalk-l	$\ominus 73.7$	$\ominus 74.6$	$\ominus 75.1$	76.9	77.0	$\ominus 3394.9$	$\oplus 762.0$	$\oplus 1654.8$	2242.4	2229.0
Tic-tac-toe	$\ominus 84.7$	98.3	$\ominus 94.9$	98.4	97.9	$\ominus 128.5$	$\oplus 26.0$	95.8	42.8	79.6

of noise, XofN(r) combines the values 1 and 2 of the X -of- N to form one branch. This makes further learning slightly harder. Actually, XofN(r) creates two other appropriate new attributes X -of- $\{A_4 = 1\}$ and X -of- $\{A_5 = 3\}$. Unfortunately, subtrees containing these two new attributes are pruned at the end. Consequently, XofN(r) does not learn a correct tree for Monks3. In the Monks1 domain, XofN(r) learns a larger tree than XofN

because they generate different X -of- N representations, while the tree built by $XofN(r)$ is also correct. It is worth mentioning that $XofN(r)$ achieves the same accuracies and theory complexities as ID2-of-3 in the majority domains. The former performs much better than the latter in the parity domains in terms of both higher prediction accuracy and lower theory complexity.

In the natural domains, $XofN(r)$ behaves similarly to $XofN$. $XofN(r)$ learns more accurate trees than C4.5 in all the ten domains with accuracy increases in five domains being significant. In terms of theory complexity, $XofN(r)$ is significantly better than C4.5 in five out of the ten domains. Only in one domain, does $XofN(r)$ build significantly more complex trees than C4.5. The accuracy difference between $XofN(r)$ and $XofN$ is not significant in any of these natural domains. Only in two natural domains, does $XofN(r)$ build significantly more complex trees than $XofN$. Compared with CAT, although $XofN(r)$ generates significantly more complex trees in five out of the ten natural domains, it achieves significantly higher accuracies in one domain. Compared with ID2-of-3, $XofN(r)$ builds significantly less complex trees in four out of the ten domains, and significantly more complex trees in three domains. $XofN(r)$ is significantly more accurate than ID2-of-3 in three out of the ten domains. In these natural domains, $XofN(r)$ does not obtain any significantly lower accuracy than SFRINGE, CI3, CAT, or ID2-of-3.

5.2. Forming binary splits using X -of- N attributes

So far, X -of- N representations have been investigated as nominal attributes for constructive induction. Since they have ordered values, X -of- N representations can also be treated as numeric attributes. In decision tree learning, numeric attributes are used to produce binary splits. This provides a method of alleviating the fragmentation problem of nominal X -of- N s.

Based on this consideration, a variant of the $XofN$ algorithm, called $XofN(c)$, is developed. It is the same as $XofN$ except that when forming tests at decision nodes, X -of- N representations are treated as numeric attributes. That is, $XofN(c)$ searches for the best value as the cut point for each available X -of- N attribute with respect to information gain ratio, and uses the best test to build the decision node. As $XofN$, $XofN(c)$ also considers using primitive attributes if tests formed using X -of- N representations are not better than them. If an X -of- N is used at a decision node, two branches corresponding to X -of- $N \leq \tau$ and X -of- $N > \tau$ are created, where τ is the cut point. Transforming an X -of- N into a binary test by using one cut point is a special case of subsetting or subranging (with two subsets or subranges). It is sufficient for learning concepts such as DNF, CNF, at-least, and at-most M -of- N concepts that require X -of- N representations with only one cut point.

In $XofN$ and $XofN(c)$, constructing X -of- N representations and using them to build trees are two separate processes. X -of- N representations are treated as nominal attributes when being constructed in both $XofN$ and $XofN(c)$. They can also be treated as numeric attributes when being created. This suggests another variant of the $XofN$ algorithm, which can also alleviate the fragmentation problem of nominal X -of- N s. The $XofN(cc)$ algorithm implements this idea. It treats X -of- N representations as numeric attributes both when creating them and when using them to build decision trees.

When building decision nodes, $XofN(cc)$ is the same as $XofN(c)$. When constructing an X -of- N representation at a decision node, $XofN(cc)$ differs from $XofN$ and $XofN(c)$ in the following manner. For each candidate X -of- N representation examined, $XofN(cc)$ treats it as a numeric attribute and finds the cut point that results in the highest evaluation function value for the X -of- N attribute. This value is used when this candidate X -of- N is compared with other candidate X -of- N s.

X -of- N s in $XofN(cc)$ are very similar to M -of- N representations. The difference is that when searching for an M -of- N attribute, a cut point is found and is fixed as a part of the new attribute, while for a numeric X -of- N attribute, a cut point is found only for obtaining the evaluation function value of the new attribute. The cut point of a numeric X -of- N used when forming a test for a decision node can be different from that found when it is created, especially when an X -of- N is reused. This gives $XofN(cc)$ an advantage over a similar algorithm that constructs M -of- N attributes. For example, in the Monks2 domain, both algorithms create and use an appropriate new attribute at the root. At a decision node underneath the root, $XofN(cc)$ reuses the numeric X -of- N created at the root but with a different cut point and builds a correct tree. However, the algorithm constructing M -of- N attributes fails to create another appropriate M -of- N due to the fact that the local training set is small. In addition, since the M -of- N attribute with a fixed M generated at the root cannot be reused at nodes underneath the root, the algorithm fails to build a correct tree with M -of- N attributes.

A single X -of- N representation as a numeric attribute produces less complex partitions in the instance space than it does as a nominal attribute, because numeric attributes are usually transformed into binary tests by using cut points when used to generate decision trees. However, a single numeric X -of- N representation still can directly and simply represent any concept that can be represented by a single conjunctive, a single disjunctive, a single at-least M -of- N , or a single at-most M -of- N representation, and the reverse is not true. To represent each of an exactly M -of- N concept, an even parity concept, and an odd parity concept, the same X -of- N representation needs to be used several times with different cut points. For example, the representation of an exactly M -of- N concept with a numeric X -of- N is: $(X\text{-of-}N \leq M) \text{ AND } (X\text{-of-}N > M - 1)$.

To investigate whether treating X -of- N s as numeric attributes can avoid the fragmentation problem in practice, we conduct a set of experiments using $XofN(c)$ and $XofN(cc)$ in the 27 artificial and natural domains. The same experimental methods presented in the previous section are used. The prediction accuracies and theory complexities of $XofN(c)$ and $XofN(cc)$ are given in Table B.1 in Appendix B. The observations are as follows. For a detailed analysis of the results, see Zheng (1996).

1. $XofN(c)$ and $XofN(cc)$ do not suffer from the fragmentation problem. In all the DNF, CNF, and multiplexor domains where $XofN$ suffers from the fragmentation problem, both $XofN(c)$ and $XofN(cc)$ achieve significantly improvement over C4.5 in terms of both higher prediction accuracy and lower theory complexity.¹⁹ In these ten domains, the prediction accuracies and theory complexities of $XofN(c)$ and $XofN(cc)$ are better than those of ID2-of-3, and similar to those of SFRINGE, CI3, and CAT. Most of these accuracy increases and complexity decreases over ID2-of-3 are significant.

2. As XofN, both XofN(c) and XofN(cc) demonstrate their advantage over SFRINGE, CI3, and CAT in the parity and majority domains in terms of higher prediction accuracy and lower theory complexity. They also show their advantage over ID2-of-3 in the parity domains, and achieve the same accuracies and complexities as ID2-of-3 in the majority domains.
3. XofN performs worse than XofN(c) and XofN(cc) in domains which need X -of- N representations with only one cut point, such as the DNF domains. The reason is that nominal X -of- N s suffer from the fragmentation problem. The former performs better than the latter in domains which need X -of- N representations with more than one cut point, such as the parity domains. The reason is that it is hard for XofN(c) and XofN(cc), especially XofN(cc), to find appropriate cut points for X -of- N representations in this kind of domain.
4. In the natural domains, the overall performance of XofN is slightly better than that of XofN(c) and XofN(cc), while the overall performance of XofN(c) is slightly better than that of XofN(cc).

As far as computational requirements are concerned, XofN(c) and XofN(cc) are generally slower than XofN, while XofN(cc) is slower than XofN(c). The reason is that XofN searches for one X -of- N representation as a nominal attribute at each decision node, while XofN(c) needs extra time to find a cut point when forming a test for a decision node after creating an X -of- N representation. Also, XofN divides the training data more rapidly, thus building trees with fewer nodes. XofN(cc) searches for a cut point for each candidate X -of- N representation during the process of generating new attributes. Therefore, it spends more time than XofN and XofN(c). For experimental results about the computational requirements of XofN(c) and XofN(cc) compared with XofN and C4.5, see Zheng (1996).

6. Related work

The closest related work is ID2-of-3 (Murphy & Pazzani, 1991). It constructs new binary attributes in the form of M -of- N representations, while XofN constructs X -of- N representations. When building a decision tree, both of them construct one new attribute for each decision node using the local training set. Instead of building decision trees, CRLS (Spackman, 1988) and MoN (Ting, 1994) learn M -of- N rules. The MOFN algorithm (Towell & Shavlik, 1993) extracts M -of- N rules from refined neural networks. The symbolic theory revision system NEITHER (Baffes & Mooney, 1993) refines M -of- N rules. In addition, Hampson and Volper (1986, 1987) present a connectionist method for learning at-least M -of- N concepts.²⁰ Ortega (1995) uses M -of- N concepts to improve the domain theory of the DNA promoter problem.

The rule learning algorithms INDUCE (Michalski, 1978), AQ17-DCI (Bloedorn & Michalski, 1998), and AQ17-MCI (Bloedorn et al., 1993) use the counting operator²¹ $\#VarEQ(x)$ to construct new attributes that count the number of attributes which take the value x . For primitive boolean attributes, a boolean counting operator takes a vector of n boolean attributes ($n \geq 2$) and counts the number of true values for an instance. Like X -of- N s, new

attributes constructed using these two operators have ordered discrete values. When used to generate production rules, they are treated more like numeric attributes than nominal attributes.²² The boolean counting attribute is a special case of the $\#VarEQ(x)$ attribute, while the $\#VarEQ(x)$ attribute is a special case of the X -of- N representation. Two variants of Michalski's attribute counting operators are used to construct new terms (attributes) for learning evaluation functions over search states in problem solving systems. They are CINDI (Callan & Utgoff, 1991) and ZENITH (Fawcett & Utgoff, 1992). The UQ transformation of CINDI creates a numeric attribute from a boolean expression beginning with a universal quantifier. A generated UQ term calculates the percentage of permutations of variable bindings that satisfy the boolean expression. New features (attributes) generated by ZENITH consist of two components: A formula in the form of a conjunction or a disjunction of terms, and a variable list. A feature is evaluated in a state by counting the distinct values of its variable list that satisfies the formula.

BSEJ (Pazzani, 1996) is a method of constructing new nominal attributes using Cartesian products of existing nominal attributes. It adopts the wrapper model (John, Kohavi, & Pfleger, 1994). Starting from the set of primitive nominal attributes, BSEJ carries out a hill-climbing search to iteratively combine two existing attributes to form a Cartesian product or delete one existing attribute. It can achieve substantial increases in accuracy in some natural domains for naive Bayesian classifier learning and instance-based learning. However, it has not demonstrated general benefit for decision tree learning in natural domains.

Some systems construct new numeric attributes by using mathematical operators such as multiplication and division. The science discovery system BACON (Langley et al., 1987) and rule induction system INDUCE (Michalski, 1978) are two examples.

Most hypothesis-driven constructive induction (Wnek & Michalski, 1994) algorithms such as FRINGE (Pagallo, 1990), CITRE (Matheus & Rendell, 1989), CI (Zheng, 1992), CAT (Zheng, 1998), and AQ17-HCI (Wnek & Michalski, 1994) construct a set of new attributes based on the entire training set. This strategy has a shortcoming: New attributes that have high values of an evaluation function for the entire training set might have lower values than other unselected new attributes for a training subset after a part of a decision tree or a ruleset has been created (Matheus & Rendell, 1989). To overcome this, the XofN algorithm constructs one new attribute using the local training set for each decision node. Therefore, the new attribute constructed by this algorithm at each decision node is the best that can be found in their search space in terms of the evaluation function. Another difference between the XofN algorithm and the other algorithms is that the latter interleave the theory learning phase and the process of building new attributes, and generate new attributes by analyzing the previously learned theory, while the XofN algorithm only uses one iteration and constructs new attributes by analyzing data.

Like ID2-of-3 and XofN, LFC (Ragavan & Rendell, 1993) is also a data-driven constructive induction algorithm that builds multivariate trees, but it uses negation and conjunction as constructive operators. LFC creates one conjunction for each decision node by using directed lookahead search. It achieved quite high prediction accuracies in some natural domains such as Pima Indians diabetes, but the problem is that it has a sensitive parameter "Lookahead Depth" which needs to be set when applied to a domain. Another multivariate tree learning algorithm is LMDT (Brodley & Utgoff, 1992) that generates a linear machine

as a nominal attribute with a fixed number of values at each decision node when building a tree. LMDT is reported to build significantly more accurate decision trees than C4.5 in two natural domains.

XofN uses subsetting, subbranching, or forming binary splits to alleviate the fragmentation problem of nominal X -of- N representations. Another possible solution is building decision graphs (Oliver, Dowe, & Wallace, 1992; Kohavi & Li, 1995; Oliveira & Sangiovanni-Vincentelli, 1995) instead of decision trees. Subsetting and subbranching combine several values of a nominal X -of- N attribute to form one outcome for the test derived using the X -of- N , thus forming only one subtree for these values. When building a decision graph using nominal X -of- N attributes, some outcomes of decision nodes could be joined to one subgraph. Therefore, it should be able to alleviate the fragmentation problem of nominal X -of- N attributes as well. On the other hand, Pagallo and Haussler (1990) and Friedman, Kohavi, and Yun (1996) address the fragmentation problem for selective decision tree learning. Pagallo and Haussler (1990) generate conjunctions to form tests at decision nodes as a solution to the problem, while Friedman, Kohavi, and Yun (1996) build lazy decision trees, creating one decision path for each test example.

At the moment, XofN uses cut points found by C4.5 to discretize primitive numeric attributes. Other discretization methods that could be used include multi-interval discretization methods (Catlett, 1991; Fayyad & Irani, 1993), supervised/unsupervised methods (Van de Merckt, 1993), and an entropy method (Ragavan & Rendell, 1993). Both Catlett (1991) and Fayyad and Irani (1993) recursively apply a binary splitting procedure to a numeric attribute, but they use different stopping criteria. Van de Merckt (1993) employs a clustering method with an unsupervised monothetic contrast criterion or a mixed supervised/unsupervised monothetic criterion to obtain cut points. The difference between the two criteria is that the latter incorporates an entropy measure. Ragavan and Rendell (1993) intervalize numeric attributes by minimizing the class entropy in each interval. Furthermore, the current XofN discretizes numeric attributes statically in the sense that discretization occurs before new attribute construction. An alternative is dynamic discretization, i.e. carrying out discretization while constructing new attributes. This method might be able to create good discretization but with increased computational complexity.

As far as search methods are concerned, the greedy search with two operators: Adding and deleting attribute-value pairs, which is used in the “Search- X -of- N ()” function, can be considered as a combination of forward selection and backward elimination. The forward selection and the backward elimination have been used for relevant attribute subset selection (John, Kohavi, & Pfleger, 1994). They have been used for improving the naive Bayesian classifier (Pazzani, 1996) as well. In the statistics community, they have been studied under the names forward stepwise selection and backward stepwise elimination (Draper & Smith, 1981; Neter, Wasserman, & Kutner, 1990).

7. Conclusions and future work

This paper has proposed a novel new attribute construction operator, X -of- N . Since X -of- N representations have ordered discrete values, they can be used as either nominal or numeric

attributes for constructive induction. We have explained that X -of- N representations can directly and simply represent more concepts than conjunctive, disjunctive, and M -of- N representations commonly used for constructive induction. It has been indicated that nominal X -of- N attributes suffer from the fragmentation problem when learning problems need many long X -of- N representations. Three methods, subsetting, subranging, and forming binary splits, have been discussed and experimentally shown to be able to alleviate the fragmentation problem of nominal X -of- N attributes. We have found that nominal X -of- N attributes do not significantly suffer from the fragmentation problem in the natural domains from UCI under investigation.

Based on the constructive operator X -of- N , we have explored methods of constructing nominal and numeric attributes. A novel constructive decision tree learning algorithm, XofN, has been described. It employs the data-driven constructive strategy. At each decision node, it constructs one X -of- N representation by using a greedy search based on the local training set. When building decision trees, XofN uses X -of- N representations as nominal or numeric attributes.

As mentioned before, since the XofN algorithm considers the reuse of new attributes, generating subtrees in descending order of the sizes of training subsets that go to the subtrees could be useful. More powerful discretization methods for primitive numeric attributes and other search methods such as Swap (Indurkha & Weiss, 1991) and Random Mutation hill climbing (Skalak, 1994) may be helpful for creating good X -of- N attributes. Up to now, we have only explored approaches to constructing X -of- N attributes for decision tree learning. Approaches to constructing X -of- N attributes for rule learning and decision graph learning are worthy of future investigation. In addition, the current XofN algorithm constructs X -of- N representations based only on primitive attributes. It can be extended to using both primitive attributes and previously created new attributes. This would allow XofN to construct more complex new attributes. However, an open question is whether such complex concepts exist in real-world applications.

The XofN algorithm has been evaluated using experiments in artificial and natural domains. The results illustrate the learning power of this algorithm in the domains studied in terms of both higher prediction accuracy and lower theory complexity. It has been shown that the performance of decision tree learning can be significantly improved by constructing X -of- N attributes in most of the artificial and natural domains tested.

From the experimental comparison with the constructive decision tree learning algorithms that construct conjunctive, disjunctive (implicitly), or M -of- N representations as new binary attributes, we find that the overall performance of XofN is better than that of these algorithms in the set of artificial and natural domains under investigation. XofN achieves significantly higher prediction accuracies than these algorithms in some artificial and natural domains, while none of these algorithms gains a significantly higher accuracy than XofN in any of these domains. It has been clearly demonstrated that XofN performs significantly better than the algorithms that construct conjunctions or disjunctions for the parity-like and M -of- N -like concepts, and significantly better than the algorithm that constructs M -of- N representations for the parity-like concepts in terms of both higher prediction accuracy and lower theory complexity.

Appendix A: Experimental results of XofN(s)

Table A.1. Accuracies (%) and theory complexities of C4.5, XofN, and XofN(s). XofN(s) refers to XofN with subsetting. In the table, $\ominus$ ($\oplus$) indicates that C4.5 or XofN is significantly worse (better) than XofN(s). As sub-ranging, subsetting can also effectively alleviate the fragmentation problem of nominal X -of- N representations.

Domain	Accuracy(%)			Complexity		
	C4.5	XofN	XofN(s)	C4.5	XofN	XofN(s)
DNF1	$\ominus 87.2$	$\ominus 86.1$	98.5	$\ominus 263.0$	$\ominus 300.6$	118.9
DNF2	$\ominus 90.7$	$\ominus 87.7$	99.6	$\ominus 202.0$	$\ominus 234.0$	50.7
DNF3	$\ominus 93.8$	$\ominus 93.1$	99.4	$\ominus 101.8$	$\ominus 88.0$	46.2
DNF4	$\ominus 74.0$	$\ominus 74.5$	99.4	$\ominus 525.0$	$\ominus 593.2$	78.0
CNF1	$\ominus 86.9$	$\ominus 84.6$	99.2	$\ominus 271.2$	$\ominus 314.2$	103.0
CNF2	$\ominus 90.6$	$\ominus 87.1$	99.4	$\ominus 192.8$	$\ominus 227.0$	42.8
CNF3	$\ominus 93.3$	$\ominus 93.2$	99.5	$\ominus 94.4$	$\ominus 90.0$	44.0
CNF4	$\ominus 72.9$	$\ominus 73.5$	100.0	$\ominus 532.8$	$\ominus 607.8$	52.7
MX6	100.0	100.0	100.0	$\ominus 48.6$	$\ominus 60.6$	28.2
MX11	$\ominus 97.2$	$\ominus 96.6$	100.0	$\ominus 168.6$	$\ominus 243.6$	65.3
Parity4	$\ominus 67.5$	100.0	100.0	$\ominus 238.4$	$\ominus 9.0$	6.0
Parity5	$\ominus 52.2$	100.0	100.0	$\ominus 1339.4$	13.4	11.7
Maj11	$\ominus 82.9$	100.0	100.0	$\ominus 461.6$	$\ominus 23.0$	13.2
Maj13	$\ominus 76.3$	100.0	100.0	$\ominus 527.4$	$\ominus 27.0$	15.9
Monks1	$\ominus 75.7$	100.0	100.0	18.0	17.0	15.0
Monks2	$\ominus 65.0$	100.0	100.0	31.0	13.0	9.0
Monks3	97.2	$\oplus 100.0$	97.2	12.0	9.0	4.0
Cleveland	$\ominus 73.3$	79.8	78.5	49.8	41.1	42.4
Hepatitis	78.2	79.4	79.4	13.6	13.4	13.6
Liver	62.1	70.1	64.7	79.4	89.1	93.1
Diabetes	71.5	70.8	72.1	$\oplus 128.8$	$\oplus 153.6$	176.0
Wisconsin	94.8	94.9	95.1	20.6	26.3	22.2
Promoters	76.3	88.5	83.5	$\ominus 22.6$	13.9	14.0
Nettalk-p	$\ominus 81.1$	83.9	83.8	$\ominus 2339.2$	1506.0	1586.8
Nettalk-s	$\ominus 82.7$	87.6	87.6	$\ominus 2077.3$	$\oplus 739.6$	839.6
Nettalk-l	$\ominus 73.7$	76.9	76.6	$\ominus 3394.9$	2242.4	2339.4
Tic-tac-toe	$\ominus 84.7$	98.4	98.8	$\ominus 128.5$	42.8	74.1

Appendix B: Experimental results with X-of-N s as numeric attributes

Table B.1. Accuracies (%) and theory complexities of C4.5, XofN, XofN(c), and XofN(cc) in the artificial and natural domains. XofN(c) and XofN(cc) are two variants of XofN. XofN(c) treats X-of-Ns as numeric attributes when using them to build decision trees. XofN(cc) treats X-of-Ns as numeric attributes both when constructing them and when using them to build decision trees. In the table, $\oplus$ ($\ominus$) indicates that the prediction accuracy or theory complexity of an algorithm is significantly better (worse) than that of C4.5. These results show that numeric X-of-N representations do not suffer from the fragmentation problem, but on average they perform slightly worse than nominal X-of-N representations in the natural domains.

Domain	Accuracy(%)				Complexity			
	C4.5	XofN	XofN(c)	XofN(cc)	C4.5	XofN	XofN(c)	XofN(cc)
DNF1	87.2	86.1	$\oplus 98.9$	$\oplus 100.0$	263.0	300.6	$\oplus 83.3$	$\oplus 62.0$
DNF2	90.7	$\ominus 87.7$	$\oplus 99.5$	$\oplus 100.0$	202.0	234.0	$\oplus 55.6$	$\oplus 70.9$
DNF3	93.8	93.1	$\oplus 99.0$	$\oplus 99.8$	101.8	88.0	$\oplus 45.7$	$\oplus 56.9$
DNF4	74.0	74.5	$\oplus 99.2$	$\oplus 100.0$	525.0	$\ominus 593.2$	$\oplus 65.2$	$\oplus 52.2$
CNF1	86.9	84.6	$\oplus 97.1$	$\oplus 100.0$	271.2	314.2	$\oplus 122.6$	$\oplus 62.0$
CNF2	90.6	$\ominus 87.1$	$\oplus 99.4$	$\oplus 99.6$	192.8	227.0	$\oplus 47.1$	$\oplus 63.6$
CNF3	93.3	93.2	$\oplus 99.5$	$\oplus 99.9$	94.4	90.0	$\oplus 49.0$	$\oplus 64.0$
CNF4	72.9	73.5	$\oplus 99.7$	$\oplus 100.0$	532.8	$\ominus 607.8$	$\oplus 72.6$	$\oplus 52.2$
MX6	100.0	100.0	100.0	100.0	48.6	$\ominus 60.6$	$\oplus 30.3$	$\oplus 19.4$
MX11	97.2	96.6	$\oplus 100.0$	$\oplus 100.0$	168.6	$\ominus 243.6$	$\oplus 49.4$	$\oplus 49.9$
Parity4	67.5	$\oplus 100.0$	$\oplus 100.0$	$\oplus 100.0$	238.4	$\oplus 9.0$	$\oplus 21.0$	$\oplus 21.0$
Parity5	52.2	$\oplus 100.0$	$\oplus 100.0$	$\oplus 95.1$	1339.4	$\oplus 13.4$	$\oplus 34.0$	$\oplus 285.2$
Maj11	82.9	$\oplus 100.0$	$\oplus 100.0$	$\oplus 100.0$	461.6	$\oplus 23.0$	$\oplus 13.0$	$\oplus 13.0$
Maj13	76.3	$\oplus 100.0$	$\oplus 100.0$	$\oplus 100.0$	527.4	$\oplus 27.0$	$\oplus 15.0$	$\oplus 15.0$
Monks1	75.7	$\oplus 100.0$	$\oplus 100.0$	$\oplus 100.0$	18.0	17.0	23.0	12.0
Monks2	65.0	$\oplus 100.0$	$\oplus 100.0$	$\oplus 100.0$	31.0	13.0	15.0	15.0
Monks3	97.2	$\oplus 100.0$	97.2	$\ominus 95.4$	12.0	9.0	4.0	11.0
Cleveland	73.3	$\oplus 79.8$	$\oplus 77.9$	75.9	49.8	41.1	42.6	43.9
Hepatitis	78.2	79.4	79.4	79.4	13.6	13.4	12.2	11.8
Liver	62.1	$\oplus 70.1$	65.8	61.8	79.4	89.1	101.9	86.7
Diabetes	71.5	70.8	72.0	$\oplus 74.5$	128.8	153.6	$\ominus 169.3$	146.5
Wisconsin	94.8	94.9	94.6	95.6	20.6	$\ominus 26.3$	$\ominus 24.8$	20.4
Promoters	76.3	$\oplus 88.5$	83.5	81.7	22.6	$\oplus 13.9$	$\oplus 14.3$	$\oplus 12.9$
Nettalk-p	81.1	$\oplus 83.9$	$\oplus 84.4$	$\oplus 83.4$	2339.2	$\oplus 1506.0$	$\oplus 1595.7$	$\oplus 1848.8$
Nettalk-s	82.7	$\oplus 87.6$	$\oplus 87.5$	$\oplus 86.5$	2077.3	$\oplus 739.6$	$\oplus 791.9$	$\oplus 986.1$
Nettalk-l	73.7	$\oplus 76.9$	$\oplus 77.9$	$\oplus 76.7$	3394.9	$\oplus 2242.4$	$\oplus 2337.7$	$\oplus 2640.8$
Tic-tac-toe	84.7	$\oplus 98.4$	$\oplus 98.1$	$\oplus 97.4$	128.5	$\oplus 42.8$	$\oplus 70.4$	$\oplus 68.0$

Acknowledgments

Much of the research reported here was carried out when the author was at the University of Sydney. It was partially supported by an ARC grant (to Ross Quinlan) and by a research agreement with Digital Equipment Corporation. I appreciate the fruitful advice and suggestions on the idea and earlier versions of this paper Ross Quinlan gave. Many thanks to Geoff Webb, Kai Ming Ting, Douglas Newlands, Alen Varšek, Thierry Van de Merckt, Pavel Brazdil, Pat Langley, Jason Catlett, Philip Chung, Nitin Indurkha, Mike Cameron-Jones, Larry Rendell, William Cohen, Michael Pazzani, David Aha, Peter Turney, Larry Hunter, and Haym Hirsh for their very useful comments that improved the ideas and earlier versions, also to Ross Quinlan for providing C4.5, and Patrick Murphy for supplying the code of ID2-of-3. This paper has benefited greatly from the anonymous reviewers' suggestions and comments.

Notes

1. Some more sophisticated attributes such as structured attributes may be used, but here we discuss only these three most commonly used types of attribute.
2. These count how many attributes of an example have a specified value.
3. This equals the number of classes of the learning problem.
4. When constructing new attributes, coding cost is also used. See the next subsection for details.
5. If the size of an X -of- N representation is less than or equal to two, the deleting operator cannot be applied because the best X -of- N of size one has already been found.
6. These are examples that are neither the majority class of the subset nor the first alternative class.
7. It is worthwhile mentioning that this new attribute selection criterion is not an optimal one. It is just a learning bias, and works reasonably well in our preliminary experiments. More appropriate criteria are worthy of future exploration.
8. This is an arbitrary default setting.
9. For tests derived from numeric attributes, the outcome " $\leq \dots$ " goes first. For tests derived from primitive nominal attributes, outcomes are in the order of the attribute values that appear in the training set description. For nominal X -of- N attributes, outcomes are in ascending order of values of the X -of- N s.
10. Personal communication, 1995.
11. In the original description of the Monks problems (Thrun et al., 1991), the six attributes and their values have meaningful names. Here, they are simplified.
12. The classes of 5% of the training examples are reversed.
13. In the experiments reported in this section, neither subsetting nor subranging is used.
14. The accuracy improvement of XofN over SFRINGE in Parity5 is quite large, but the significance level of the t-test is 0.0858 which is slightly worse than 0.05.
15. The theory complexities of the rule learning algorithms given in Table 11 are different from those presented in Zheng (1995). The theory complexity of a ruleset here is a sum of the number of conditions in all the rules of the set and the number of rules, while it is only the number of conditions in all the rules of the set in that paper.
16. Each set of attribute value sets corresponds to a test on the attribute, thus producing a partition of the training examples.
17. The same experimental methods described in the previous section are used here.
18. In the MX6 domain, there is no accuracy difference as all these three algorithms achieve 100% prediction accuracy.
19. There is no accuracy improvement in the MX6 domain, since all the algorithms have the 100% accuracy.
20. Although Hampson and Volper (1986, 1987) use the term " $(X \text{ of } N)$ ", what they refer to is the M -of- N concept described in this paper and other literature.

21. In INDUCE, it is called #v.COND.
22. Generated rules have the form like $(\#VarEQ(1) \geq 3)$ (Thrun et al., 1991).

References

- Almuallim, H. & Dietterich, T. (1992). Efficient algorithms for identifying relevant features. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence* (pp. 38–45). Vancouver, BC: Morgan Kaufmann.
- Baffes, P. & Mooney, R. (1993). Symbolic revision of theories with M -of- N rules. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1135–1140). San Mateo, CA: Morgan Kaufmann.
- Blake, C., Keogh, E., & Merz, C. (1999). UCI repository of machine learning databases [http://www.ics.uci.edu/~mllearn/mlrepository.html]. Department of Information and Computer Science, University of California, Irvine, CA.
- Bloedorn, E. & Michalski, R. (1998). Data-driven constructive induction. *IEEE Intelligent Systems, Special issue on Feature Transformation and Subset Selection*, 13(2), 30–37.
- Bloedorn, E., Michalski, R., & Wnek, J. (1993). Multistrategy constructive induction: AQ17-MCI. In *Proceedings of the Second International Workshop on Multistrategy Learning* (pp. 188–203).
- Boulton, D. & Wallace, C. (1973a). An information measure for hierarchic classification. *Computer Journal*, 16, 254–261.
- Boulton, D. & Wallace, C. (1973b). An information measure for single-link classification. *Computer Journal*, 18, 236–238.
- Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification And Regression Trees*. Belmont, CA: Wadsworth.
- Brodley, C. & Utgoff, P. (1992). Multivariate versus univariate decision trees. COINS Technical Report 92-8, Department of Computer Science, University of Massachusetts, Amherst, MA.
- Callan, J. & Utgoff, P. (1991). A transformational approach to constructive induction. In *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 122–126). San Mateo, CA: Morgan Kaufmann.
- Caruana, R. & Freitag, D. (1994). Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 28–36). San Francisco, CA: Morgan Kaufmann.
- Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *Proceedings of the Fifth European Working Session on Learning* (pp. 164–178). Berlin: Springer-Verlag.
- Drafer, N. & Smith, H. (1981). *Applied Regression Analysis*, 2nd ed. New York: Wiley.
- Fawcett, T. & Utgoff, P. (1992). Automatic feature generation for problem solving systems. In *Proceedings of the Ninth International Workshop on Machine Learning* (pp. 144–153). San Mateo, CA: Morgan Kaufmann.
- Fayyad, U. & Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1022–1027). San Mateo, CA: Morgan Kaufmann.
- Friedman, J., Kohavi, R., & Yun, Y. (1996). Lazy decision trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (pp. 717–724). Menlo Park, CA: The AAAI Press.
- Hampson, S. & Volper, D. (1986). Linear function neurons: Structure and training. *Biological Cybernetics*, 53, 203–217.
- Hampson, S. & Volper, D. (1987). Disjunctive models of boolean category learning. *Biological Cybernetics*, 56, 121–137.
- Hart, G. (1987). Minimum information estimation of structure. Doctoral Dissertation (LIDS-TH-1664), Dept of Electrical Engineering and Computer Science, MIT.
- Indurkha, N. & Weiss, S. (1991). Iterative rule induction methods. *Journal of Applied Intelligence*, 1, 43–54.
- John, G., Kohavi, R., & Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 121–129). San Francisco, CA: Morgan Kaufmann.
- Kira, K. & Rendell, L. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 129–134). Menlo Park, CA: AAAI Press.

- Kohavi, R. & Li, C. (1995). Oblivious decision trees, graphs, and top-down pruning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1071–1077). San Mateo, CA: Morgan Kaufmann.
- Langley, P. (1994). Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance*. New Orleans, LA: AAAI Press.
- Langley, P. & Sage, S. (1994). Oblivious decision trees and abstract cases. In *Working Notes of the AAAI-94 Workshop on Case-Based Reasoning* (pp. 113–117). Seattle, WA: AAAI Press.
- Langley, P., Simon, H., Bradshaw, G., & Zytkow, J. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*. Cambridge, MA: MIT Press.
- Matheus, C. & Rendell, L. (1989). Constructive induction on decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 645–650). San Mateo, CA: Morgan Kaufmann.
- Michalski, R. (1978). Pattern recognition as knowledge-guided computer induction. Tech. Rep. 927, Department of Computer Science, The University of Illinois at Urbana-Champaign, Urbana, IL.
- Michalski, R. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2, 349–361.
- Michalski, R. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20, 111–161.
- Moore, A. & Lee, M. (1994). Efficient algorithms for minimizing cross validation error. In *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 190–198). San Francisco, CA: Morgan Kaufmann.
- Murphy, P. & Pazzani, M. (1991). ID2-of-3: Constructive induction of M -of- N concepts for discriminators in decision trees. In *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 183–187). San Mateo, CA: Morgan Kaufmann.
- Neter, J., Wasserman, W., & Kutner, M. (1990). *Applied Linear Statistical Models*, 3rd ed. Homewood, IL: Irwin.
- Oliveira, A. & Sangiovanni-Vincentelli, A. (1995). Inferring reduced ordered decision graphs of minimum description length. In *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 421–429). San Francisco, CA: Morgan Kaufmann.
- Oliver, J., Dowe, D., & Wallace, C. (1992). Inferring decision graphs using the minimum message length principle. In *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence* (pp. 361–367). Singapore: World Scientific.
- Ortega, J. (1995). Research note: On the informativeness of the DNA promoter sequences domain theory. *Journal of Artificial Intelligence Research*, 2, 361–367.
- Pagallo, G. (1990). Adaptive decision tree algorithms for learning from examples. Doctoral Dissertation, Department of Computer and Information Sciences, University of California, Santa Cruz, CA.
- Pagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5, 71–99.
- Pazzani, M. (1996). Constructive induction of Cartesian product attributes. In *Proceedings of the Conference, ISIS'96: Information, Statistics and Induction in Science* (pp. 66–77). Singapore: World Scientific.
- Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.
- Quinlan, J. & Rivest, R. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation*, 80, 227–248.
- Ragavan, H. & Rendell, L. (1993). Lookahead feature construction for learning hard concepts. In *Proceedings of the Tenth International Conference on Machine Learning* (pp. 252–259). San Mateo, CA: Morgan Kaufmann.
- Rissanen, J. (1978). Modeling by shortest data description. *Automatica*, 14, 465–471.
- Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11, 416–431.
- Rissanen, J. (1986). Stochastic complexity and modeling. *Annals of Statistics*, 14, 1080–1100.
- Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 293–301). San Francisco, CA: Morgan Kaufmann.
- Spackman, K. (1988). Learning categorical decision criteria in biomedical domains. In *Proceedings of the Fifth International Conference on Machine Learning* (pp. 36–46). San Mateo, CA: Morgan Kaufmann.
- Thrun, S., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., De Jong, K., Džeroski, S., Fahlman, S., Fisher, D., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R., Mitchell, T., Pachowicz, P., Reich, Y., Vafaie, H., Van de Welde, W., Wenzel, W., Wnek, J., & Zhang, J. (1991). The

- MONK's problems—A performance comparison of different learning algorithms. Tech. Rep. CMU-CD-91-197, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
- Ting, K. (1994). An M -of- N rule induction algorithm and its application to DNA domain. In *Proceedings of the Twenty-seventh Annual Hawaii International Conference on System Sciences, Volume V: Biotechnology Computing* (pp. 133–140). Los Alamitos, CA: IEEE Computer Society Press.
- Towell, G. & Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13, 71–101.
- Van de Merckt, T. (1993). Decision trees in numerical attribute spaces. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1016–1021). San Mateo, CA: Morgan Kaufmann.
- Wallace, C. & Boulton, D. (1968). An information measure for classification. *Computer Journal*, 11, 185–194.
- Wallace, C. & Patrick, J. (1993). Coding decision trees. *Machine Learning*, 11, 7–22.
- Wnek, J. & Michalski, R. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14, 139–168.
- Yang, D., Rendell, L., & Blix, G. (1991). A scheme for feature construction and a comparison of empirical methods. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence* (pp. 699–704). San Mateo, CA: Morgan Kaufmann.
- Yip, S. & Webb, G. (1994). Incorporating canonical discriminant attributes in classification learning. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence* (pp. 63–70). Vancouver, BC: Morgan Kaufmann.
- Zheng, Z. (1992). Constructing conjunctive tests for decision trees. In *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence* (pp. 355–360). Singapore: World Scientific.
- Zheng, Z. (1995). Constructing nominal X -of- N attributes. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence* (pp. 1064–1070). San Mateo, CA: Morgan Kaufmann.
- Zheng, Z. (1996). Constructing new attributes for decision tree learning. Doctoral Dissertation, Basser Department of Computer Science, The University of Sydney [available at <http://www3.cm.deakin.edu.au/~zijian/Papers/thesis.ps.gz>].
- Zheng, Z. (1998). Constructing conjunctions using systematic search on decision trees. *Knowledge Based Systems Journal*, 10, 421–430.

Received October 29, 1998

Accepted March 1, 1999

Final manuscript February 26, 1999