International Journal of Software Engineering
and Knowledge Engineering
Vol. 16, No. 1 (2006) 103–126

World Scientific
www.worldscientific.com

LEVEL-WISE CONSTRUCTION OF DECISION TREES FOR CLASSIFICATION

YAN ZHAO*, YIYU YAO† and JINGTAO YAO‡

Department of Computer Science, University of Regina,
Regina, Saskatchewan, Canada S4S 0A2
*yanzhao@cs.uregina.ca
†yyao@cs.uregina.ca
‡jtyao@cs.uregina.ca*

A partition-based framework is presented for a formal study of classification problems. An information table is used as a knowledge representation, in which all basic notions are precisely defined by using a language known as the decision logic language. Solutions to, and solution space of, classification problems are formulated in terms of partitions. Algorithms for finding solutions are modelled as searching in a space of partitions under the refinement order relation. We focus on a particular type of solutions called conjunctively definable partitions. Two level-wise methods for decision tree construction are investigated, which are related to two different strategies: local optimization and global optimization. They are not in competition with, but are complementary to each other. Experimental results are reported to evaluate the two methods.

Keywords: Classification; level-wise construction; partition-based framework; decision tree.

1. Introduction

Knowledge engineering is a technique used to build intelligent systems. It involves integrating knowledge into computer systems in order to solve complex problems that normally require a high level of human expertise. To effectively practise knowledge engineering, a knowledge engineer requires knowledge in two main areas, namely, knowledge representation and knowledge modelling [9]. These two areas provide a conceptual basis for the study and implementation of intelligent systems. In this paper, we study knowledge engineering in a specific field, known as classification. Classification is one of the main tasks in machine learning, data mining and pattern recognition [4, 12, 14, 24]. It deals with classifying labeled objects.

There are two distinct views of the study of classification and the study of knowledge engineering. By utilizing classification, or more generally, machine learning and data mining methods, one can discover new knowledge from data automatically. This may raise new philosophical questions and promote the philosophical study of knowledge engineering. By utilizing the knowledge engineering approaches,

one can inquire into the nature of machine learning and data mining, and simulate and improve the techniques related to them. It is important to note that these two studies are interweaved.

The mainstream research of classification focuses on classification algorithms and their experimental evaluations [1, 2, 3, 7, 11, 13, 15, 17, 18, 20]. In comparison, less attention has been paid to the study of fundamental concepts with respect to knowledge engineering, such as the structures of a search space, the solutions of a classification problem, as well as the structures of a solution space.

Knowledge for classification can be expressed in different forms, such as classification rules [2, 3, 13], discriminant functions [4], decision trees [1, 11, 17, 18, 20] and decision graphs [7, 15]. Different frameworks are required for different forms of classification. They produce different kinds of search spaces, solution representations, and solution spaces. For example, a partition-based framework is suitable for studying the fundamental concepts of decision tree classification method, and a covering-based framework is suitable for studying the classification rule method.

The objective of this paper is two-fold. First, we study a partition-based framework, which lays the groundwork for our present study of decision trees. Second, the partition-based framework is used to study the level-wise construction of decision trees. The level-wise construction method can construct a decision tree and evaluate its classification performance level by level. The framework provides some insights into the nature of classification tasks, and enables us to find solutions in different search spaces. Based on the proposed framework, there are two types of level-wise construction approaches: a parallelized depth-first method, and a breadth-first method. The former can be modelled as searching solutions in the tree-definable partition spaces, and later can be modelled as searching solutions in the attribute-induced partition spaces. For example, using conditional entropy as the searching heuristic, one can modify the ID3 algorithm [17] to a parallelized depth-first algorithm, LID3. Using the same heuristic, one can modify the 1R algorithm [5] to a breadth-first algorithm, kLR. They are complementary to the existing decision tree algorithms. The study of the level-wise construction is an application of the proposed framework. This conceptual study can provide guidelines and set the stage for the technique and application studies.

This paper is an extended version of the paper published in the proceedings of SEKE 2004 [27]. The rest of the paper is organized as follows. A formal partition-based framework for classification problems, as well as the decision tree solutions, are presented in Sec. 2. Two level-wise construction methods are discussed in Sec. 3. Experimental evaluations of the proposed methods are summarized in Sec. 4. The conclusions are drawn in Sec. 5.

2. Partition-Based Framework for Classification

In this section, we introduce the concepts of information tables for knowledge representation. Based on the notion of partition, many concept spaces can be defined.

The partition-based framework can be applied to classification tasks, and finding a solution to a classification problem can be modelled as a search for a partition in a certain partition space.

2.1. *Information tables and definability of sets*

An information table provides a convenient way to describe a finite set of objects by a finite set of attributes. It deals with the issues of knowledge representation for classification problems [6, 16, 23]. Basic definitions related to information tables are summarized as follows.

Definition 1. An information table S is the tuple:

$$S = (U, At, \mathcal{L}, \{V_a | a \in At\}, \{I_a | a \in At\}), \tag{1}$$

where U is a finite nonempty set of objects, At is a finite nonempty set of attributes, $\mathcal{L}$ is a language defined by using attributes in At, commonly known as a decision logic language [23]. V_a is a nonempty set of values for a where $a \in At$, and $I_a : U \to V_a$ is an information function.

For an attribute $a \in At$ and an object $x \in U$, $I_a(x)$ denotes the value of x on a. In general, for a subset $\mathcal{A} \subseteq At$ of attributes, $\mathcal{A} = \{a_1, \ldots, a_m\}$, $I_{\mathcal{A}}(x)$ stands for the value of x on the attribute set $\mathcal{A}$, and $I_{\mathcal{A}}(x) = (I_{a_1}(x), \ldots, I_{a_m}(x))$ is a vector.

Definition 2. Formulas of $\mathcal{L}$ are defined by the following two rules:

(i) An atomic formula ϕ of $\mathcal{L}$ is a descriptor $a = v$, where $a \in At$ and $v \in V_a$;
(ii) The well-formed formulas (wff) of $\mathcal{L}$ is the smallest set containing the atomic formulas and closed under $\neg$, $\wedge$, $\vee$, $\to$ and $\equiv$.

Definition 3. In an information table S, the satisfiability of a formula ϕ by an object x, written as $x \models_S \phi$, or in short $x \models \phi$ if S is understood, is defined as follows:

$$\begin{aligned}
&(1) \quad x \models a = v \text{ iff } I_a(x) = v,\\
&(2) \quad x \models \neg\phi \text{ iff not } x \models \phi,\\
&(3) \quad x \models \phi \wedge \psi \text{ iff } x \models \phi \text{ and } x \models \psi,\\
&(4) \quad x \models \phi \vee \psi \text{ iff } x \models \phi \text{ or } x \models \psi,\\
&(5) \quad x \models \phi \to \psi \text{ iff } x \models \neg\phi \vee \psi,\\
&(6) \quad x \models \phi \equiv \psi \text{ iff } x \models \phi \to \psi \text{ and } x \models \psi \to \phi.
\end{aligned}$$

Definition 4. If ϕ is a formula, the set $m_S(\phi)$ defined by

$$m_S(\phi) = \{x \in U | x \models \phi\} \tag{2}$$

is called the meaning of the formula ϕ in S. If S is understood, we simply write $m(\phi)$.

The meaning of a formula ϕ is the set of all objects having the properties expressed by the formula ϕ. A connection between formulas of $\mathcal{L}$ and subsets of U is thus established.

Definition 5. A subset $X \subseteq U$ is called a definable granule in an information table S if there exists at least one formula ϕ such that $m(\phi) = X$. For a subset of attributes $\mathcal{A} \subseteq At$, X is an $\mathcal{A}$-definable granule if there exists at least one formula $\phi_{\mathcal{A}}$ using only attributes from $\mathcal{A}$ such that $m(\phi_{\mathcal{A}}) = X$.

The notion of definability includes two meanings: (1) The set X of objects satisfies ϕ, denoted as $x \models \phi$, for all $x \in X$, (2) All objects possessing ϕ are in X. The notion of definability of subsets in an information table is essential to data analysis. In fact, definable subsets are the basic units that can be described and discussed, upon which other notions can be developed.

In many classification algorithms, one is only interested in formulas of a certain form. Suppose we restrict the connectives of language $\mathcal{L}$ to only conjunction $\wedge$. Each formula is a conjunction of atomic formulas, and such a formula is referred to as a conjunctor.

Definition 6. A subset $X \subseteq U$ is a conjunctively definable granule in an information table S if there exists a conjunctor ϕ such that $m(\phi) = X$.

We similarly can define the disjunctively definable granules and other classes of definable sets [26].

2.2. *Partitions in an information table*

Classification involves the division of the set U of objects into many classes. The notion of partitions provides a formal means to describe classification.

Definition 7. A partition π of a set U is a collection of nonempty, and pairwise disjoint subsets of U whose union is U. Subsets in a partition are called blocks.

Different partitions may be related to each other. We can define an order relation on partitions.

Definition 8. A partition π_1 is a refinement of another partition π_2, or equivalently, π_2 is a coarsening of π_1, denoted by $\pi_1 \preceq \pi_2$, if every block of π_1 is contained in some blocks of π_2.

The refinement relation is a partial order, namely, it is reflexive, antisymmetric, and transitive. This naturally defines a refinement order on the set of all partitions, and in fact forms a partition lattice $\Pi(U)$. Given two partitions π_1 and π_2, the meet of their blocks, $\pi_1 \wedge \pi_2$, are all nonempty intersections of a block from π_1 and a block from π_2. The join of their blocks, $\pi_1 \vee \pi_2$, are the smallest subsets which are exactly a union of blocks from π_1 and π_2. The meet, $\pi_1 \wedge \pi_2$, is the largest

refinement partition of both π_1 and π_2; the join, $\pi_1 \vee \pi_2$, is the smallest coarsening partition of both π_1 and π_2.

We extend the notion of definability of subsets to the definability of partitions.

Definition 9. A partition π is called a definable partition in an information table S if every block of π is a definable granule. For a subset of attributes $\mathcal{A} \subseteq At$, a partition π is an $\mathcal{A}$-definable partition if every block of π is an $\mathcal{A}$-definable granule.

The family of all definable partitions is denoted by $\Pi_D(U)$, which is a subset of $\Pi(U)$.

Definition 10. A partition π is called a conjunctively definable partition if every block of π is a conjunctively definable granule.

The family of all conjunctively definable partitions is denoted by $\Pi_{CD}(U)$, which is a subset of $\Pi_D(U)$.

Table 1. An information table.

	A	B	C
o_1	0	0	0
o_2	0	0	0
o_3	0	0	1
o_4	1	1	0

Example 1: Given an information Table 1, $U = \{o_1, o_2, o_3, o_4\}$ and At consists of three binary attributes $\{A, B, C\}$. We can define many partitions in this table, such as:

$$\pi_1 : \{\{o_1, o_2\}, \{o_3, o_4\}\};$$
$$\pi_2 : \{\{o_1\}, \{o_2\}, \{o_3, o_4\}\};$$
$$\pi_3 : \{\{o_1\}, \{o_2, o_3\}, \{o_4\}\};$$
$$\pi_4 : \{\{o_1, o_2\}, \{o_3\}, \{o_4\}\}.$$

For each partition, the blocks are pairwise disjoint, and their union is U. The refinement relation exists among these partitions. For instance, $\pi_2 \preceq \pi_1$ since every block of π_2 is contained in some block of π_1. Similarly, we can observe $\pi_4 \preceq \pi_1$. We can also observe that,

$$\pi_1 \wedge \pi_2 = \{\{o_1\}, \{o_2\}, \{o_3, o_4\}\};$$
$$\pi_1 \wedge \pi_3 = \{\{o_1\}, \{o_2\}, \{o_3\}, \{o_4\}\};$$
$$\pi_2 \wedge \pi_4 = \{\{o_1\}, \{o_2\}, \{o_3\}, \{o_4\}\};$$
$$\pi_1 \vee \pi_2 = \{\{o_1, o_2\}, \{o_3, o_4\}\};$$
$$\pi_2 \vee \pi_3 = \{\{o_1\}, \{o_2, o_3, o_4\}\};$$
$$\pi_3 \vee \pi_4 = \{\{o_1, o_2, o_3\}, \{o_4\}\}.$$

In the information table, π_1 and π_4 are verified as definable partitions, but π_2 and π_3 are not. Objects o_1 and o_2 are not distinguishable within the table by the language $\mathcal{L}$. Thus, blocks $\{o_1\}$, $\{o_2\}$ and $\{o_2, o_3\}$ cannot be defined by any formula. Furthermore, within the definable partitions π_1 and π_4, only π_4 is conjunctively definable partition. The block $\{o_3, o_4\}$ in the partition π_1 has to use a disjunction (e.g. $C = 1 \vee A = 1$) or a negation (e.g. $\neg(A = 0 \wedge B = 0 \wedge C = 0)$) to define.

Definition 11. A partition is called a tree-definable partition π_{tree} if there exists a tree that induces the partition.

In an information table, we can partition the universe by using a tree. An internal node is labelled by an attribute, a branch from the node is labelled by a value of that attribute, a leaf consists of a subset of objects. For an internal node a and a branch v, we can form a formula $a = v$. All objects in a leaf satisfy the conjunction of conditions from the root to the leaf. Thus, a leaf is a conjunctively definable granule. The family of all leaves forms a partition of the universe. The tree-definable partitions have been used extensively in machine learning [17]. The family of all tree-definable partitions is denoted by $\Pi_{tree}(U)$, which is a subset of $\Pi_{CD}(U)$.

Table 2. Another information table.

	A	B	C	D
o_1	a_1	b_1	c_1	d_2
o_2	a_1	b_1	c_2	d_2
o_3	a_1	b_2	c_1	d_1
o_4	a_1	b_2	c_2	d_1
o_5	a_2	b_1	c_1	d_2
o_6	a_2	b_1	c_2	d_1
o_7	a_2	b_2	c_1	d_2
o_8	a_2	b_2	c_2	d_1
o_9	a_3	b_1	c_1	d_2
o_{10}	a_3	b_1	c_2	d_1
o_{11}	a_3	b_2	c_1	d_1
o_{12}	a_3	b_2	c_2	d_1

Example 2: Given an information Table 2, $U = \{o_1, o_2, \ldots, o_{12}\}$ and $At = \{A, B, C, D\}$. Suppose that attribute A is used for constructing a partition. By adding A to the conjunctor, the universe U is partitioned by A into three blocks, labelled by $A = a_1$, $A = a_2$ and $A = a_3$, respectively. Suppose, attribute B is then used for the block $A = a_1$, and attribute C is used for the block $A = a_2$ and $A = a_3$. Attribute B is added to the block $A = a_1$ to form two finer blocks, $A = a_1 \wedge B = b_1$ and $A = a_1 \wedge B = b_2$; attribute C is added to the block $A = a_2$ to form two finer blocks, $A = a_2 \wedge C = c_1$ and $A = a_2 \wedge C = c_2$. Similarly, attribute C is added to the block $A = a_3$ to form two finer blocks, $A = a_3 \wedge C = c_1$ and $A = a_3 \wedge C = c_2$. For each block there is an attribute for the further partition. The partition can be continued until the user is willing to stop or no more finer granules

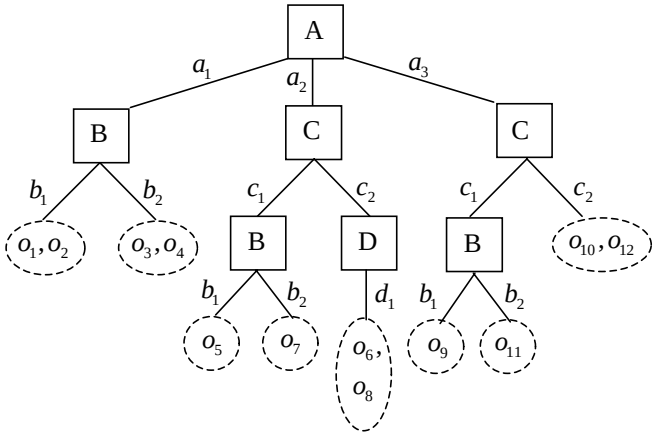


Fig. 1. A tree structure of a tree-definable partition.

can be found. Every block is conjunctively defined by attribute-values, the blocks form a tree structure. As a result, each level of the tree can use different attributes, and the same attributes can appear at different levels. Figure 1 demonstrates one of such trees associated with the tree-definable partition.

Definition 12. For a set of attributes $\mathcal{A} \subseteq At$, let $\pi_{\mathcal{A}}$ denote the partition defined by $\mathcal{A}$. The corresponding equivalence relation $E_{\mathcal{A}}$ is defined by

$$xE_{\mathcal{A}}y \Leftrightarrow I_a(x) = I_a(y), \text{ for all } a \in \mathcal{A}. \tag{3}$$

The corresponding partition is called an attribute-induced partition, more specifically, an $\mathcal{A}$-induced partition.

The attribute-induced partitions have been studied in databases [8] and rough set theory [16]. Clearly, $\pi_{\emptyset} = U$ is the coarsest partition, and π_{At} is the finest partition, for any $\mathcal{A} \subseteq At$, we have $\pi_{At} \preceq \pi_{\mathcal{A}} \preceq \pi_{\emptyset}$. The family of all attribute-induced partitions forms a partition lattice $\Pi_{attr}(U)$, which is a subset of $\Pi_{tree}(U)$.

Example 3: We use the same information Table 2, by adding the attribute A to the conjunctor, the universe U is partitioned by A into $A = a_1$, $A = a_2$ and $A = a_3$ three blocks. By adding the second attribute B to each conjunctor, each block is partitioned into two finer blocks, labelled by $A = a_1 \wedge B = b_1$, $A = a_1 \wedge B = b_2$, $A = a_2 \wedge B = b_1$, $A = a_2 \wedge B = b_2$, $A = a_3 \wedge B = b_1$, and $A = a_3 \wedge B = b_2$, respectively. Same as the tree-definable partition, the attribute-induced partition can be continued by adding more attributes to the conjunctors, until the user is willing to stop or no more finer partitions can be found. Every block is conjunctively defined by attribute-values, the blocks also form a tree structure. It is easy to identify that in an attribute-induced partition, each level of the tree has only one attribute. Therefore, a tree associated with the attribute-induced partition is a

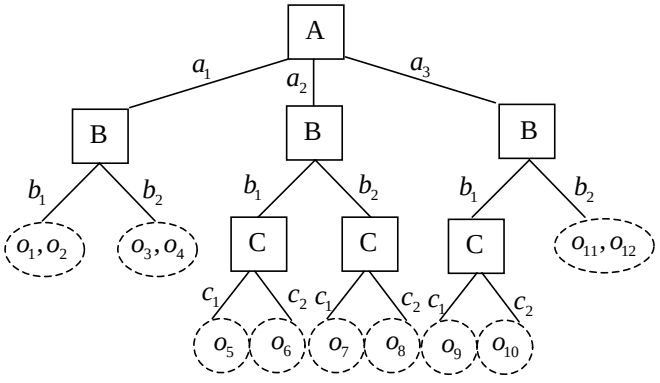


Fig. 2. A tree structure of an attribute-induced partition.

special tree in terms of the tree-definable partition. Figure 2 demonstrates a tree associated with the attribute-induced ($\{A, B, C\}$-induced) partition.

We have the following connection among the partition families. They provide a formal basis of classification problems.

$$\Pi_{attr}(U) \subseteq \Pi_{tree}(U) \subseteq \Pi_{CD}(U) \subseteq \Pi_D(U) \subseteq \Pi(U)\,. \tag{4}$$

2.3. *Classification problems and the solutions*

In an information table for classification problems, we have a set of attributes $At = F \cup \{\mathbf{class}\}$. This kind of information tables is also called *decision tables* [16]. The problem can be formally stated in terms of partitions.

2.3.1. *The consistent and inconsistent classification problems*

Definition 13. A decision table is said to define a consistent classification if objects with the same description have the same class value, namely, for any two objects $x, y \in U$, $I_F(x) = I_F(y)$ implies $I_{\mathbf{class}}(x) = I_{\mathbf{class}}(y)$, otherwise, the decision table defines an inconsistent classification.

The partition $\pi_{\mathbf{class}}$ is not always definable by a descriptive attribute set $\mathcal{A} \subseteq F$. If $\pi_{\mathbf{class}}$ is not definable then the decision table does not define a consistent classification. Therefore, the purpose of a classification task can be understood as a search for precise or approximate descriptions of $\pi_{\mathbf{class}}$ defined by $\mathcal{A} \subseteq F$.

2.3.2. *The deterministic and probabilistic solutions*

In the partition-based framework, we can easily define a deterministic solution to a classification problem.

Definition 14. A partition π defined by $\mathcal{A} \subseteq F$ is said to be a deterministic solution to a classification problem if $\pi \preceq \pi_{\mathbf{class}}$.

Suppose π is a deterministic solution to a classification problem, namely, $\pi \preceq \pi_{\mathbf{class}}$. For a pair of equivalence classes $X \in \pi$ and $C_j \in \pi_{\mathbf{class}}$ with $X \subseteq C_j$, we can derive a classification rule $Des(X) \Longrightarrow Des(C_j)$, where $Des(X)$ and $Des(C_j)$ are the formulas that describe sets X and C_j, respectively.

In many practical situations, one is satisfied with a probabilistic solution of a classification problem instead of a deterministic solution; in other practical situations, one may only obtain probabilistic solutions since the partition $\pi_{\mathbf{class}}$ is not consistently defined by the given decision table. Therefore, we define the probabilistic solution to the inconsistent classification problem.

Definition 15. Let $\rho : \pi \times \pi_{\mathbf{class}} \longrightarrow \Re^+$ be a function such that $\rho(\pi, \pi_{\mathbf{class}})$ measures the degree to which $\pi \preceq \pi_{\mathbf{class}}$ is true, where $\Re^+$ are non-negative reals. For a threshold α, a partition π is said to be a probabilistic solution if $\rho(\pi, \pi_{\mathbf{class}}) \geq \alpha$.

Suppose π is a probabilistic solution to a classification problem. For a pair of equivalence classes $X \in \pi$ and $C_j \in \pi_{\mathbf{class}}$, we can derive a classification rule $Des(X) \Longrightarrow \rho(X, C) Des(C_j)$, where $Des(X)$ and $Des(C_j)$ are the formulas that describe sets X and C_j, respectively.

The measure ρ can be defined to capture various aspects of classification. Two such measures are discussed below. They are the *ratio of sure classification* (RSC), and the *accuracy* of classification.

Definition 16. For the partition $\pi = \{X_1, X_2, \ldots, X_m\}$, the ratio of sure classification (RSC) by π is given by:

$$\rho_1(\pi, \pi_{\mathbf{class}}) = \frac{|\bigcup\{X_i \in \pi | X_i \subseteq C_j \text{ for some } C_j \in \pi_{\mathbf{class}}\}|}{|U|}, \tag{5}$$

where $|\cdot|$ denotes the cardinality of a set. The ratio of sure classification represents the percentage of objects that can be classified by π without any uncertainty. The measure $\rho_1(\pi, \pi_{\mathbf{class}})$ reaches the maximum value 1 if $\pi \preceq \pi_{\mathbf{class}}$, and reaches the minimum value 0 if for all blocks $X_i \in \pi$ and $C_j \in \pi_{\mathbf{class}}$, $X_i \subseteq C_j$ does not hold. For two partitions with $\pi_1 \preceq \pi_2$, we have $\rho_1(\pi_1, \pi_{\mathbf{class}}) \geq \rho_1(\pi_2, \pi_{\mathbf{class}})$.

Definition 17. For the partition $\pi = \{X_1, X_2, \ldots, X_m\}$, the accuracy of classification by a partition is defined by:

$$\rho_2(\pi, \pi_{\mathbf{class}}) = \frac{\sum_{i=1}^{m} |X_i \cap C_{j(X_i)}|}{|U|}, \tag{6}$$

where $C_{j(X_i)} = arg\max\{|C_j \cap X_i| \| C_j \in \pi_{\mathbf{class}}\}$. The accuracy of π is in fact the weighted average accuracies of individual rules. The measure $\rho_2(\pi, \pi_{\mathbf{class}})$ reaches the maximum value 1 if $\pi \preceq \pi_{\mathbf{class}}$.

A deterministic solution is a special case of a probabilistic solution with $\rho_1 = 100\%$ and $\rho_2 = 100\%$.

The abovementioned two measures are only two examples for evaluating the classification performance. Additional measures can be defined based on the properties of partitions. For example, many quantitative measures and information-theoretic measures can be used [25, 28].

If the partition π_F is a deterministic solution to a classification problem, namely, $\pi_F \preceq \pi_{\mathbf{class}}$, then the problem is a consistent classification problem; if π_F is a probabilistic solution with respect to a pair of ρ and $\alpha < 100\%$, then the problem is an inconsistent classification.

If the classification problem is consistent, we either can search for deterministic solutions or probabilistic solutions; if the classification problem is inconsistent, only probabilistic solutions can be obtained.

2.3.3. *The solution space*

Let the set of partitions Π_{sol} be the set of all solutions, including the deterministic and probabilistic solutions. Π_{sol} is a subset of the set of partitions, and Π_{sol} forms a solution space. The relationship between the partition-based solution space and the partition-based search spaces is illustrated in Fig. 3. Some search spaces are too broad to be practically useful.

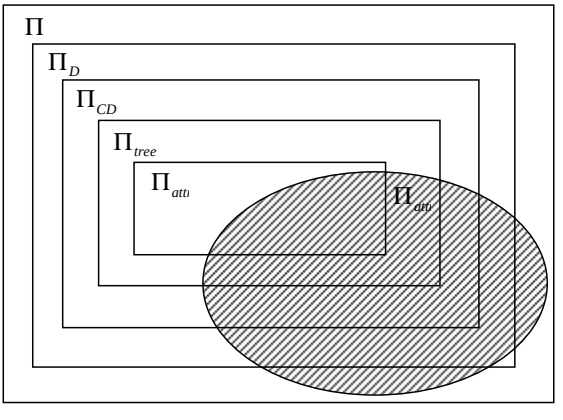


Fig. 3. The partition-based solution space and the search spaces.

For consistent classification, the partition π_F is the minimum element of Π_{sol}. A solution π is called the *most general solution* if there does not exist another solution π' such that $\pi \prec \pi' \preceq \pi_{\mathbf{class}}$, where $\pi \prec \pi'$ stands for $\pi \neq \pi'$ and $\pi \preceq \pi'$. For two partitions with $\pi_1 \preceq \pi_2$, if π_2 is a solution, then π_1 is also a solution. For two solutions π_1 and π_2, $\pi_1 \wedge \pi_2$ is also a solution. The solution space Π_{sol} is closed under meet subset. In most cases, we are interested in the most general solutions.

2.4. *Classification as searching*

Conceptually, finding a solution to a decision tree classification problem can be modelled as a search for a partition π in the whole partition space Π_D under the order relation $\preceq$. A difficulty with this straightforward search is that the space is too large to be practically applicable. One may avoid such a difficulty in several ways. For instance, one can search the space of conjunctively definable partitions. The general classification method by the conjunctively definable partition approach can be described as:

Continue to search for a finer partition of π by adding one more attribute unless

(i) $\rho(\pi, \pi_{\mathbf{class}}) \geq \alpha$, or
(ii) No more finer partitions can be found.

In particular, in searching the conjunctively definable partitions, two special search spaces deserve consideration, namely, the family Π_{tree} of the tree-definable partitions, and the family Π_{attr} of the attribute-induced partitions.

Strategy 1: The ID3 algorithm is considered as the milestone of the decision tree method [17]. There are many variants of ID3 and its later version known as the C4.5 algorithm [18]. In this paper, we refer to them as the ID3-like algorithms. The ID3-like algorithms search the space of tree-definable partitions (Π_{tree}, refer to Fig. 3). Typically, a classification is constructed in a depth-first manner until the leaf nodes are subsets that consist of elements of the same class with respect to **class**. By labeling the leaves by the class symbol of **class**, we obtain a commonly used decision tree for classification. The bias of searching solutions in the space of tree-definable partitions is to find the shortest tree.

Strategy 2: Searching solutions in the space of attribute-induced partitions (Π_{attr}, refer to Fig. 3) can be achieved by breadth-first classification methods [16, 22]. Suppose all the attributes of F have the same priorities. The goal of solution searching is to find a subset of attributes $\mathcal{A}$ so that $\pi_{\mathcal{A}}$ is the most general solution to the classification problem.

Algorithms of breadth-first classification are studied extensively in the field of rough set. The important notions of rough set based approaches are summarized below:

Definition 18. An attribute $a \in \mathcal{A}$ is called a core attribute, if $\pi_{F-\{a\}}$ is not a solution, i.e., $\neg(\pi_{F-\{a\}} \preceq \pi_{\mathbf{class}})$.

Definition 19. A subset $\mathcal{A} \subseteq F$ is called a reduct, if $\pi_{\mathcal{A}}$ is a solution and for any subset $B \subset \mathcal{A}$, π_B is not a solution. That is,

(i) $\pi_{\mathcal{A}} \preceq \pi_{\mathbf{class}}$;
(ii) For any proper subset $B \subset \mathcal{A}$, $\neg(\pi_B \preceq \pi_{\mathbf{class}})$.

A reduct is a set of attributes that, individually, are necessary and jointly sufficient for classification. There may exist more than one reduct. Each reduct provides

one solution to the classification problems. A core attribute must be presented in each reduct, namely, a core attribute is used in every solution to the classification problem. The set of core attributes is the intersection of all reducts. The bias of searching solutions in the space of attribute-induced partitions is to find a set of attributes that is, or close to a reduct.

Conceptually, one can combine these two searches together, i.e., construct a classification tree by using a selected attribute set. This enables us to compare the two types of methods in the same setting in the next two sections. The comparison leads to the introduction of level-wise construction of decision trees.

3. Level-Wise Construction of Decision Trees

Two level-wise construction methods are discussed in this section. The term "level-wise" is proposed corresponding to the term of "depth-first". It constitutes two different kinds of search methods, either parallelized depth-first search, or breadth-first search.

Most of the existing ID3-like algorithms perform the depth-first construction of a decision tree. Mitchell pointed out that the ID3-like algorithms can be conceptually interpreted in a breadth-first manner [14]. That is, all nodes are split in one level before moving to another level. Since the partition of a node is independent of the other nodes, the depth-first method can be easily parallelized. Examples of such algorithms are SLIQ [11] and SPRINT [20]. The parallel computation applies special data structures, and can improve the efficiency of classification. In this paper, we propose a simple breadth-first algorithm called LID3 as a level-wise construction version of ID3. Intuitively, both ID3 and LID3 are understood as searches in the tree-definable partition space. LID3 results in the same tree structure as the ID3 tree, but judges the overall performance of classification (such as ρ_1 RSC and ρ_2 accuracy) in a level-wise fashion. The criteria used by ID3-like methods are based on local optimization. That is, when splitting a node, an attribute is chosen based on information only about this node, but not on any other nodes on the same level. The consequence is that nodes on the same level may use different attributes, and moreover, the same attribute may be used at different levels.

In comparison with local optimization, one may think of global optimization by considering all the nodes on the same level at the same time. A breadth-first method, called kLR, is thus proposed in this paper, by extending the 1R algorithm proposed by Holte [5]. kLR is a search in the attribute-induced partition space, it chooses the attribute for all nodes on the same level. While LID3 produces the same tree as the depth-first ID3 algorithm, kLR produces a different tree. There are practical reasons for a breadth-first method. From an extensive experimental study, Holte suggested that simple classification rules perform well on many commonly used datasets [5]. In particular, Holte showed that simple rules consisting of one attribute performed reasonably well. Conceptually, such 1R rules correspond to the first level of the decision tree. By extending 1R rules, we introduce the notion of

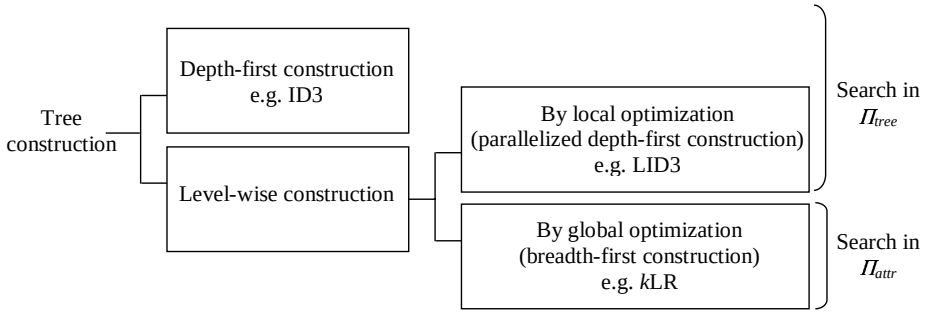


Fig. 4. Tree constructions.

kR rules, where k stands for the number of attributes used. The kR rules can be obtained by restricting the decision tree to k levels.

The main categories of decision tree construction methods are shown in Fig. 4.

3.1. *The LID3 algorithm*

The ID3 algorithm starts with the entire set of objects and is recursively partitioned by attributes, until each block is a subset of objects belong to one class [17]. A level-wise construction method of ID3, LID3, searches solutions in the tree-definable partitions, and is described in Fig. 5.

LID3: A level-wise construction of ID3

1. Let $k = 0$.
2. The k-level, $0 < k \leq |F|$, of the classification tree is built based on the $(k-1)^{th}$ level described as follows: **if** a node in $(k-1)^{th}$ level is an inconsistently classified, **then**
   - 2.1 Choose an attribute based on a certain criterion $\gamma : At \longrightarrow \Re$;
   - 2.2 Partition the inconsistent node based on the selected attribute and produce the k^{th} level nodes, which are the subsets of that node;
   - 2.3 Label the inconsistent node by the attribute name, and label the branches coming out from the node by values of the attribute.

Fig. 5. The LID3 algorithm.

The selection criterion used by ID3 is an information-theoretic measure called conditional entropy. Let S denote the set of objects in a particular node at level $(k-1)$. The conditional entropy of **class** given an attribute a is denoted by:

Table 3. A decision table.

	A	B	C	D	**Class**
o_1	a_1	b_1	c_1	d_2	$-$
o_2	a_1	b_1	c_2	d_2	$-$
o_3	a_1	b_2	c_1	d_1	$+$
o_4	a_1	b_2	c_2	d_1	$+$
o_5	a_2	b_1	c_1	d_2	$-$
o_6	a_2	b_1	c_2	d_1	$-$
o_7	a_2	b_2	c_1	d_2	$-$
o_8	a_2	b_2	c_2	d_1	$+$
o_9	a_3	b_1	c_1	d_2	$+$
o_{10}	a_3	b_1	c_2	d_1	$-$
o_{11}	a_3	b_2	c_1	d_1	$+$
o_{12}	a_3	b_2	c_2	d_1	$+$

$$\begin{aligned} H_S(\mathbf{class}|a) &= \sum_{v\in V_a} P_S(v)H(\mathbf{class}|v) \\ &= -\sum_{v\in V_a} P_S(v) \sum_{d\in V_{\mathbf{class}}} P_S(d|v)\log P_S(d|v) \\ &= -\sum_{d\in V_{\mathbf{class}}}\sum_{v\in V_a} P_S(d,v)\log P_S(d|v), \end{aligned} \tag{7}$$

where the subscript S indicates that all quantities are defined with respect to the set S. An attribute with the minimum entropy value is chosen to split a node.

For the sample decision table given by Table 3, we obtain a decision tree and its analysis of RSC and accuracy are illustrated in Fig. 6.

3.2. *The kLR algorithm*

Recall that a reduct is a set of individual, necessary and jointly sufficient attributes that correctly classify the objects. The kLR algorithm, described in Fig. 7, can find a reduct, or a subset of conditional attributes that is close to a reduct. The kLR algorithm searches in the attribute-induced partitions.

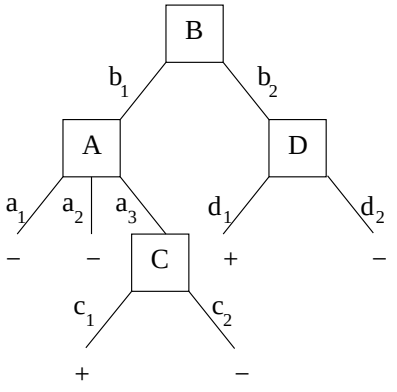
Note that when choosing an attribute, one needs to consider all the inconsistent nodes. In contrast, LID3 only considers one inconsistent node at a time.

Conditional entropy can also be used as the selection criterion γ. In this case, the subset of examples considered at each level is the union of all inconsistent nodes. Let $A_{(k-1)}$ be the set of attributes used from level 0 to level $(k-1)$. The next attribute a for level k can be selected based on the following conditional entropy:

$$H(\mathbf{class}|A_{(k-1)} \cup \{a\})\,. \tag{8}$$

The use of $A_{(k-1)}$ ensures that all inconsistent nodes at level $k-1$ are considered in the selection of level k attribute [22].

The resulting tree of the kLR algorithm is similar to the *oblivious decision tree*, i.e., given an order of the attributes, all nodes at one level are tested and



Level	Rules	RSC	Accuracy
$k = 1$ B	$b_1 \Rightarrow 5/6\ -$ $b_2 \Rightarrow 5/6\ +$	0.00	0.83
$k = 2$ ABD	$b_1 \wedge a_1 \Rightarrow 2/2\ -$ $b_1 \wedge a_2 \Rightarrow 2/2\ +$ $b_1 \wedge a_3 \Rightarrow 1/2\ +$ $b_2 \wedge d_1 \Rightarrow 5/5\ +$ $b_2 \wedge d_2 \Rightarrow 1/1\ -$	0.83	0.92
$k = 3$ $ABCD$	$b_1 \wedge a_3 \wedge c_1 \Rightarrow 1/1\ +$ $b_1 \wedge a_3 \wedge c_2 \Rightarrow 1/1\ -$	1.00	1.00

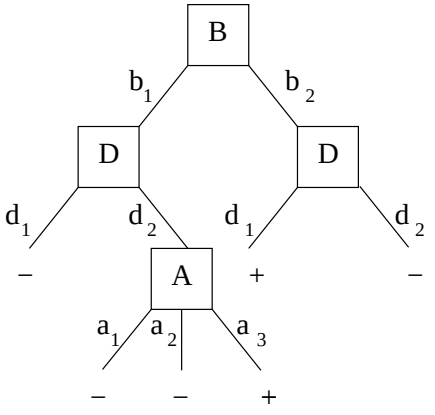
Fig. 6. An LID3 tree and its statistic.

kLR: A breadth-first level-wise construction method

1. Let $k = 0$.
2. The k-level, $0 < k \leq |F|$, of the classification tree is built based on the $(k-1)^{th}$ level described as follows: **if** there is at least one node in $(k-1)^{th}$ level that is inconsistently classified **then**
   - 2.1 Choose an attribute based on a certain criterion $\gamma : At \longrightarrow \Re$;
   - 2.2 Partition all the inconsistent nodes based on the selected attribute and produce the k^{th} level nodes, which are subsets of the inconsistent nodes;
   - 2.3 Label the inconsistent nodes by the attribute name, and label the branches coming out from the inconsistent nodes by the values of the attribute.

Fig. 7. The kLR algorithm.

partitioned by the same attribute [10]. While the order is not given, we can use the function $\gamma : At \longrightarrow \Re$ to decide an order. The criterion γ can be one of the information measures, for example, conditional entropy (as shown above) or mutual



Level	Rules	RSC	Accuracy
$k=1$ B	$b_1 \Rightarrow 5/6\ -$ $b_2 \Rightarrow 5/6\ +$	0.00	0.83
$k=2$ BD	$b_1 \wedge d_1 \Rightarrow 2/2\ -$ $b_1 \wedge d_1 \Rightarrow 3/4\ -$ $b_2 \wedge d_1 \Rightarrow 5/5\ +$ $b_2 \wedge d_2 \Rightarrow 1/1\ -$	0.67	0.92
$k=3$ ABD	$b_1 \wedge d_2 \wedge a_1 \Rightarrow 2/2\ -$ $b_1 \wedge d_2 \wedge a_2 \Rightarrow 1/1\ -$ $b_1 \wedge d_2 \wedge a_3 \Rightarrow 1/1\ +$	1.00	1.00

Fig. 8. A kLR tree and its statistic.

information, which indicate how much information the attributes contribute to the decision attribute **class**. Or, the statistical measures, for example, the χ^2 test, which indicate the dependency level between the test attribute and the decision attribute **class**. We can get such an order by testing all the attributes. The constant ordering applied by the oblivious decision tree is not enough. We need to update the order level by level. There are two reasons for level-wise updating. First, one can stop further construction when the deterministic solutions or the probabilistic solutions are found. The search space is possibly changed at different levels. Second, for each test that partitions the search space into uneven-sized blocks, the value of function γ is the sum of the function value of γ for each block multiplies the probability distribution of the block.

Consider the earlier example, based on the conditional entropy, the decision tree is built by the kLR algorithm. The analysis of RSC and accuracy is summarized in Fig. 8.

Comparing the two level-wise decision tree construction methods, we notice that kLR can construct a tree possessing the same RSC and accuracy as the LID3

decision tree with fewer attributes involved. In this example, the set of attributes $\{A, B, D\}$ is a reduct, since $\pi_{\{A,B,D\}} \preceq \pi_{\mathbf{class}}$, and for any proper subset $X \subset \{A, B, D\}$, $\neg(\pi_X \preceq \pi_{\mathbf{class}})$.

4. Experimental Evaluations

In order to evaluate level-wise construction methods, we choose some well-known datasets from the UCI machine learning repository [21]. The SGI MLC++ utilities 2.0 is used to discretize the datasets into categorical attribute sets [19]. The selected datasets and their descriptions are shown in Table 4.

Table 4. Databases used for experimental evaluation.

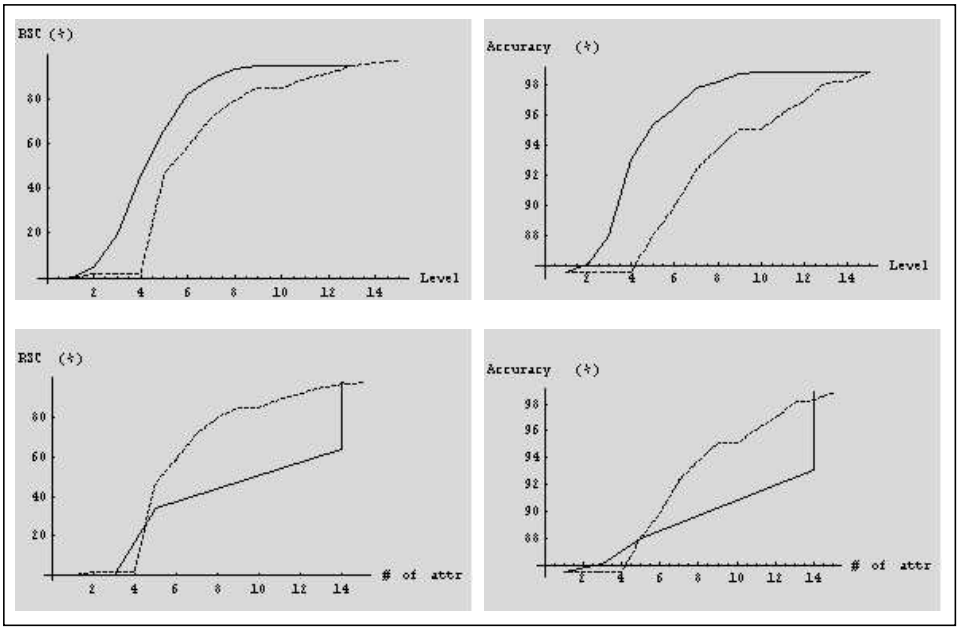
ID	Name	# of objects	# of attributes	# of classes	Classification
1	Credit	690	15	2	inconsistent
2	Cleve	303	13	2	inconsistent
3	Vote	435	16	2	consistent

Dataset 1: Credit is a discretized, inconsistently classified dataset with 15 attributes and 690 records. The detail experiment results are reported in Fig. 9.

Reading the top table of Fig. 9 we notice that both LID3 and kLR reach the same RSC and accuracy at the first level. This is because they both apply the conditional entropy as the attribute selection criterion within the entire set of conditional attributes. Also, both LID3 and kLR reach the same RSC and accuracy at the last level, level 15. This is because for the inconsistency classification tasks, classifiers will not stop searching for a better solution until all the conditional attributes are used. The classifiers reach the same performance. kLR generates more rules than its counterpart. Comparing the same level, kLR performs more inferiorly in terms of RSC and more accurately than LID3 (except RSC in level 2). However comparing the trees where the same number of attributes are used, kLR obtains higher RSC and accuracy values. For example, by using three and five attributes, kLR has a competitive RSC and accuracy to LID3, while adding up to 14 attributes at the 14th level, kLR has sufficient higher RSC and accuracy than LID3 in level 4, where 14 attributes are also used. The comparisons with respect to level and the number of attributes are plotted in Fig. 9. The two figures in the upper row compare RSC and accuracy level-by-level. The two figures in the lower row compare RSC and accuracy with respect to the number of attributes. The solid line and the dotted line indicate LID3 and kLR, respectively. When evaluating by level, the solid LID3 lines are significantly above the dotted kLR lines for both RSC and accuracy. When evaluating by number of attributes, the dotted kLR lines are above the solid LID3 lines in most cases.

The difference between RSC and accuracy at the same level of the LID3 tree is distinctly smaller than that of the kLR tree. For example, starting from level 7 of the LID3 tree, RSC and accuracy are very close to each other. It takes up to level

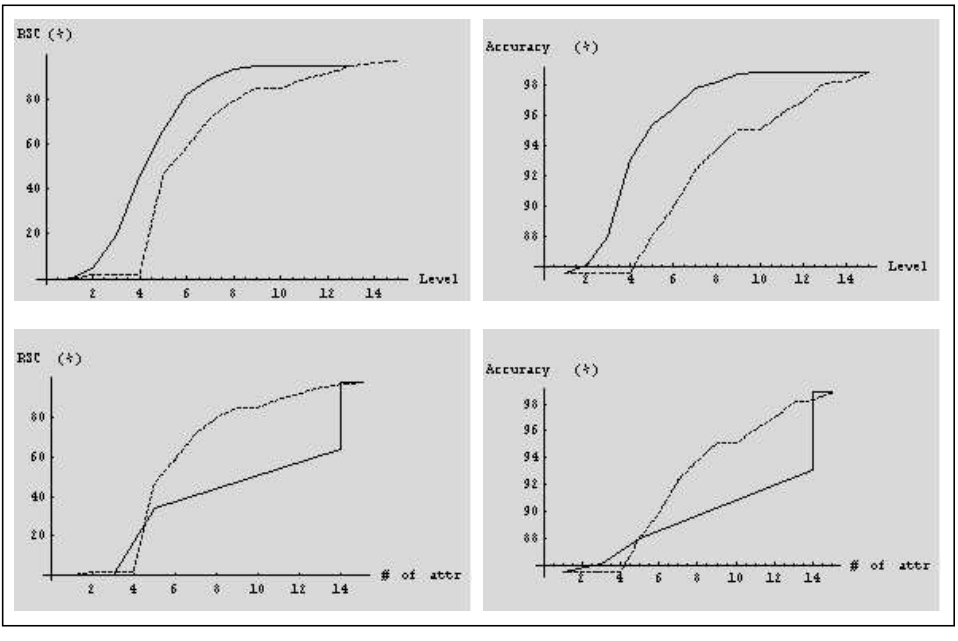
Level	LID3			kLR		
	RSC	Accuracy	# of attr.	RSC	Accuracy	# of attr.
1	0.00%	85.51%	1	0.00%	85.51%	1
2	0.29%	86.09%	3	1.74%	85.51%	2
3	34.20%	87.97%	5	1.74%	85.51%	3
4	63.48%	93.04%	14	1.74%	85.51%	4
5	81.59%	95.36%	14	46.67%	87.97%	5
6	88.41%	96.38%	14	58.55%	89.86%	6
7	94.35%	97.83%	14	71.59%	92.32%	7
8	95.94%	98.26%	14	79.57%	93.77%	8
9	96.96%	98.70%	14	84.78%	95.07%	9
10	97.25%	98.84%	14	84.78%	95.07%	10
11	97.25%	98.84%	15	88.99%	96.09%	11
12	97.25%	98.84%	15	91.59%	96.96%	12
13	97.25%	98.84%	15	94.78%	98.12%	13
14	97.25%	98.84%	15	96.23%	98.26%	14
15	97.25%	98.84%	15	97.25%	98.84%	15



Goal	LID3		kLR	
$RSC \geq 85.00\%$	$k = 6$	14 attr.	$k = 11$	11 attr.
$RSC \geq 90.00\%$	$k = 7$	14 attr.	$k = 12$	12 attr.
$RSC \geq 95.00\%$	$k = 8$	14 attr.	$k = 14$	14 attr.
$Accu. \geq 85.00\%$	$k = 1$	1 attr.	$k = 1$	1 attr.
$Accu. \geq 90.00\%$	$k = 4$	14 attr.	$k = 7$	7 attr.
$Accu. \geq 94.00\%$	$k = 5$	14 attr.	$k = 9$	9 attr.

Fig. 9. Evaluation of Dataset 1.

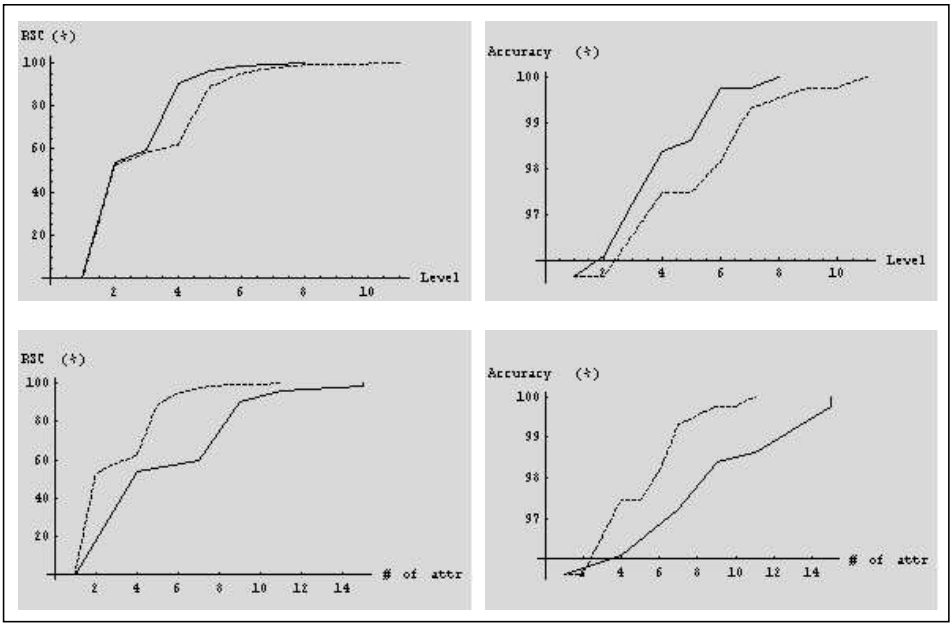
Level	LID3			kLR		
	RSC	Accuracy	# of attr.	RSC	Accuracy	# of attr.
1	0.00%	76.57%	1	0.00%	76.57%	1
2	4.95%	79.21%	4	1.98%	78.55%	2
3	19.80%	86.80%	8	7.26%	85.81%	3
4	46.86%	88.78%	11	14.85%	86.47%	4
5	65.68%	92.41%	11	39.60%	87.79%	5
6	82.18%	95.05%	11	56.11%	89.11%	6
7	89.44%	96.70%	11	69.64%	92.08%	7
8	93.40%	98.02%	11	81.85%	94.72%	8
9	94.72%	98.35%	11	88.45%	96.70%	9
10	94.72%	98.35%	13	93.07%	97.69%	10
11	94.72%	98.35%	13	94.72%	98.35%	11
12	94.72%	98.35%	13	94.72%	98.35%	12
13	94.72%	98.35%	13	94.72%	98.35%	13



Goal	LID3		kLR	
$RSC \geq 80.00\%$	$k = 6$	11 attr.	$k = 8$	8 attr.
$RSC \geq 85.00\%$	$k = 7$	11 attr.	$k = 9$	9 attr.
$RSC \geq 90.00\%$	$k = 8$	11 attr.	$k = 10$	10 attr.
$RSC \geq 94.72\%$	$k = 9$	11 attr.	$k = 11$	11 attr.
$Accu. \geq 85.00\%$	$k = 3$	8 attr.	$k = 3$	3 attr.
$Accu. \geq 90.00\%$	$k = 5$	11 attr.	$k = 7$	7 attr.
$Accu. \geq 95.00\%$	$k = 6$	11 attr.	$k = 9$	9 attr.
$Accu. = 98.35\%$	$k = 9$	11 attr.	$k = 11$	11 attr.

Fig. 10. Evaluation of Dataset 2.

	LID3			kLR		
Level	RSC	Accuracy	# of attr.	RSC	Accuracy	# of attr.
1	0.00%	95.63%	1	0.00%	95.63%	1
2	54.02%	96.09%	4	52.18%	95.63%	2
3	59.77%	97.24%	7	58.16%	96.55%	3
4	90.11%	98.39%	9	62.07%	97.47%	4
5	96.09%	98.62%	11	88.74%	97.47%	5
6	98.39%	99.77%	15	94.94%	98.16%	6
7	99.54%	99.77%	15	97.70%	99.31%	7
8	100.00%	100.00%	15	98.85%	99.54%	8
9				99.31%	99.77%	9
10				99.54%	99.77%	10
11				100.00%	100.00%	11



Goal	LID3		kLR	
$RSC \geq 95.00\%$	$k = 5$	11 attr.	$k = 7$	7 attr.
$RSC = 100.00\%$	$k = 8$	15 attr.	$k = 11$	11 attr.
$Accu. \geq 95.00\%$	$k = 1$	1 attr.	$k = 1$	1 attr.
$Accu. = 100.00\%$	$k = 8$	15 attr.	$k = 11$	11 attr.

Fig. 11. Evaluation of Dataset 3.

12 for the kLR tree to produce the distinction. This indicates that LID3 is more biased towards generating a more deterministic solution. That also indicates that for Dataset 1 at least 12 attributes are required to achieve a relatively deterministic solution.

If we want the algorithms to reach a certain performance, for example, with RSC not lower than 90%, or accuracy greater than or equal to 95%, then the search is stopped when such a criterion is reached. This is also called the *pre-pruning* method. By using pre-pruning method, the experiment results of Dataset 1 are plotted by the last table in Fig. 9. According to the first row of the table, to reach an RSC greater than or equal to 85%, LID3 needs to construct a tree of 6 levels with 14 attributes; kLR needs to construct a tree of 11 levels, but only requires 11 attributes. It shows that although kLR has higher computing complexity, it does reach a certain classification performance by using fewer attributes.

Dataset 2: Cleve is a discretized, inconsistently classified dataset with 13 attributes and 303 records. We also search for a probabilistic solution. The experimental results of this dataset are presented in the same fashion. Comparing the trees with the same number of attributes being used, we can observe that the kLR tree demonstrates much higher RSC and accuracy than the LID3 tree. For example, when eight attributes are used, kLR has RSC = 81.85% and accuracy = 94.72%, LID3 has RSC = 19.80% and accuracy = 86.80%. The kLR tree consists of 11 attributes. The experiment results are reported in Fig. 10.

Dataset 3: Vote is a discretized, consistently classified dataset with 16 attributes and 435 records. We search for the deterministic solution. In the case of LID3, the tree is 8 levels with 15 attributes for 100% of RSC and accuracy. In the case of kLR, we can reach the same level of RSC and accuracy by a 11-level-tree with 11 attributes. The experiment results are reported in Fig. 11.

Suppose we use a hybrid algorithm, i.e., first use kLR to generate a selected attribute set, consisting of 11 attributes as we mentioned above, then use LID3 to generate an LID3 tree with local optimization. The statistics of such a hybrid tree are shown in Table 5. It demonstrates that we can use the hybrid algorithm to generate a short tree with high classification performance and fewer attributes.

Table 5. The statistics of the hybrid tree of Dataset 3.

Level	RSC	Accuracy	# of attr.
1	0.00%	95.63%	1
2	54.02%	96.09%	4
3	60.69%	97.01%	6
4	89.89%	98.16%	8
5	95.63%	98.62%	9
6	97.93%	99.54%	11
7	99.54%	99.77%	11
8	100.00%	100.00%	11

The same experiments can be carried out on the other datasets. From the results of the experiments, we have the following observations. The difference of local and global selection causes different tree structures. Normally, LID3 may obtain a shorter tree as shown in the above three experiments. On the other hand, if we restrict the height of decision trees, LID3 may use more attributes than kLR. With respect to the RSC measure, an LID3 tree is normally better than a kLR tree at the same level. With respect to the accuracy measure, an LID3 tree is not substantially better than a kLR tree at the same level. With respect to different levels of two trees with the same number of attributes used, kLR obtains much better accuracy and RSC than LID3. The comparisons are summarized in Table 6. LID3 generates a more deterministic solution than kLR. The main advantage of kLR method is that it uses a fewer number of attributes to achieve the same level of accuracy. A hybrid algorithm uses the subset of attributes (suggested by kLR) to generate a short tree with level-wise higher RSC and accuracy (achieved by LID3).

Table 6. Comparison of the LID3 trees and the kLR trees.

	LID3	kLR
Length of tree	shorter	longer
# of attributes	more	less
# of blocks	less	more
RSC/$level_k$	higher	lower
Accuracy/$level_k$	higher	lower
RSC/n attributes	lower	higher
Accuracy/n attributes	lower	higher

5. Conclusion

In this paper, we develop a formal partition-based framework for the decision tree classification method. Within the framework, we are able to define precisely and concisely many fundamental notions. All the information is stored in a decision table. Notions related to the information table and the definability of set are defined. Four specific partition spaces can be distinguished. They are definable partitions, conjunctively definable partitions, tree-definable partitions, and attribute-induced partitions. Finding solutions of decision trees can be understood as a search in the tree-definable partition spaces and the attribute-induced partition spaces. An important observation is: every decision tree can be mapped to a partition. On the other hand, not all partitions can be represented in a tree structure, since the blocks of a partition may not be definable, or not definable in a conjunctive form. Even if they are conjunctively definable, they may not be able to form a tree structure. In fact, we have a many-to-one mapping between a decision tree and a tree-definable partition. We define the different types of classification tasks and solutions. The structure of solution space is studied. The connections between the partition-based search spaces and the solution space are established.

Two level-wise construction methods for the decision tree construction are suggested in the partition-based framework: a parallelized depth-first version of ID3, called LID3, searches the space of the tree-definable partitions; and a breadth-first method, called kLR, searches solutions in the attribute-induced partitions. Experimental results are reported to compare these two methods. The results are encouraging. They suggest that one needs to pay more attention to the less studied level-wise construction methods.

Acknowledgments

The authors thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of Regina for financial support, and the anonymous referees for precise and inspiring comments.

References

1. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.
2. J. Cendrowska, PRISM: An algorithm for inducing modular rules, *Int. J. Man-Machine Studies* **27** (1987) 349–370.
3. P. Clark and T. Niblett, The CN2 induction algorithm, *Machine Learning* **3** (1989) 261–283.
4. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
5. R. C. Holte, Very simple classification rules perform well on most commonly used datasets, *Machine Learning* **11** (1993) 63–91.
6. E. B. Hunt, *Concept Learning*, Wiley, New York, 1962.
7. R. Kohavi, Bottom-up induction of oblivious read-once decision graphs, in *Proc. European Conference on Machine Learning*, 1994, pp. 154–169.
8. T. T. Lee, An information-theoretic analysis of relational databases — Part I: Data dependencies and information metric, *IEEE Trans. on Software Engineering* **SE-13** (1987) 1049–1061.
9. D. Lukose, *Knowledge Engineering, PART A: Knowledge Representation*, University of New England, Australia, 1996.
http://pages.cpsc.ucalgary.ca/∼kremer/courses/CG/CGlecture_notes.html
10. O. Maimon and L. Rokach, Improving supervised learning by feature decomposition, in *Proc. Foundations of Information and Knowledge Systems*, 2002, pp. 178–196.
11. M. Mehta, R. Agrawal, and J. Rissanen, SLIQ: A fast scalable classifier for data mining, in *Proc. Int. Conf. on Extending Database Technology*, 1996.
12. R. S. Michalski, J. G. Carbonell, and T. M. Mirchell (eds.), *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, Palo Alto, CA, 1983, pp. 463–482.
13. R. S. Michalski and J. B. Larson, Selection of most representative training examples and incremental generation of vl1 hypotheses: The underlying methodology and description of programs ESEL and AQ11, *Report 867*, University of Illinois, 1978.
14. T. M. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
15. J. J. Oliver, Decision graphs — an extension of decision trees, in *Proc. Fourth Int. Workshop on Artificial Intelligence and Statistics*, 1993, pp. 343–350.

16. Z. Pawlak, *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic, Dordrecht, 1991.
17. J. R. Quinlan, Induction of decision trees, *Machine Learning*, **1** (1986) 81–106.
18. J. R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.
19. SGI's MLC++ utilities 2.0: The discretize utility. http://www.sgi.com/tech/mlc
20. J. C. Shafer, R. Agrawal, and M. Mehta, SPRINT: A Scalable Parallel Classifier for Data Mining, in *Proc. VLDB*, 1996, pp. 544–555.
21. UCI Machine Learning Repository. http://www1.ics.uci.edu/∼mlearn/MLRepository.html
22. G. Y. Wang, H. Yu, and D. C. Yang, Decision table reduction based on conditional information entropy, *Chinese Journal of Computers* **25**(7) (2002).
23. J. T. Yao and Y. Y. Yao, Induction of classification rules by granular computing, in *Proc. Int. Conf. on Rough Sets and Current Trends in Computing*, 2002, pp. 331–338.
24. J. T. Yao, Y. Y. Yao, and Y. Zhao, Foundations of classification, in *Foundations and Novel Approaches in Data Mining*, eds. T. Y. Lin, S. Ohsuga, C. J. Liau, and X. Hu, Springer-Verlag, 2005, pp. 75–97.
25. Y. Y. Yao, Information-theoretic measures for knowledge discovery and data mining, in *Entropy Measures, Maximum Entropy and Emerging Applications*, ed. Karmeshu, Springer, Berlin, 2003, pp. 115–136.
26. Y. Y. Yao, A comparative study of formal concept analysis and rough set theory in data analysis, in *Proceedings of Rough Sets and Current Trends in Computing* (*RSCTC'04*), 2004, pp. 59–68.
27. Y. Y. Yao, Y. Zhao, and J. T. Yao, Level construction of decision trees in a partition-based framework for classification, in *Proc. 16th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'04)*, 2004, pp. 199–204.
28. Y. Y. Yao and N. Zhong, An analysis of quantitative measures associated with rules, in *Proc. Third Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 1999, pp. 479–488.