Machine Learning, 53, 199–233, 2003

Learning from Cluster Examples

TOSHIHIRO KAMISHIMA mail@kamishima.net (http://www.kamishima.net/)
FUMIO MOTOYOSHI f.motoyoshi@aist.go.jp
National Institue of Advanced Industrial Science and Technology (AIST), AIST Tsukuba Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki, 305-8568 Japan

Editor: Douglas Fisher

Abstract. Learning from cluster examples (LCE) is a hybrid task combining features of two common grouping tasks: learning from examples and clustering. In LCE, each training example is a partition of objects. The task is then to learn from a training set, a rule for partitioning unseen object sets. A general method for learning such partitioning rules is useful in any situation where explicit algorithms for deriving partitions are hard to formalize, while individual examples of correct partitions are easy to specify. In the past, clustering techniques have been applied to such problems, despite being essentially unsuited to the task. We present a technique that has qualitative advantages over standard clustering approaches. We demonstrate these advantages by applying our method to problems in two domains; one with dot patterns and one with more realistic vector-data images.

Keywords: learning from examples, clustering, dot pattern, image segmentation

1. Introduction

Clustering is a typical grouping task that involves partitioning a given object set into subsets, such that objects in the same subset are "similar." (We use the term *objects* to refer to entities that will be grouped.) Clustering is carried out based on prespecified knowledge in the form, for example, of preference potential functions or dissimilarity measures.

In this paper, we advocate the use not of prespecified knowledge, but of examples for partitioning. That is, we try to acquire a partitioning rule from an example set consisting of pairs of an object set and a true partition for the object set. The acquired rule can then be used for finding a true partition for an unseen object set (not appearing in the training example set). Since the task is similar to that of learning from examples, which acquires a classification rule from a given example set, we give the new task the composite name *learning from cluster examples* (*LCE*).

A solution technique for LCE will be useful for any problem for which users can easily identify an appropriate partition for a given object set, but cannot specify explicit knowledge for deriving these partitions in general. We know of no previous technique that has been developed for this purpose. To fill a void, clustering techniques have previously been used, but they are not particularly suited to this kind of partitioning. In this paper, we present a dedicated technique, and discuss its theoretical and practical merits.

We apply our solution to experimental tasks in two domains. One task involves an artificial domain made up of dot patterns, and the other involves a realistic domain made up of vector-data images.

Since there are no algorithms designed specifically for the tasks we consider, we pay particular attention to showing the qualitative advantages over using clustering techniques for partitioning, assessing the appropriateness of the acquired partitions, and analyzing the behavior of our method in detail.

We proceed as follows. In Section 2, we show the importance of the LCE task and formalize the problem. In Sections 3 and 4, we then present partitioning and learning methods, respectively. In Section 5 these methods are applied to experimental problems. In Section 6, we discuss LCE. Finally, Section 7 summarizes our conclusions.

2. Learning from cluster examples

In this section, we present an overview of the LCE task, explain the merits of our solution method, then lay the groundwork for the remainder of the paper by giving LCE a precise mathematical formalization.

2.1. An overview of learning from cluster examples

LCE is a composite task combining features from techniques of learning from examples and of clustering. To give an overview of LCE, we therefore begin by reviewing these existing grouping tasks.

Learning from examples is a task involving the acquisition of a rule for classification from a given example set. Each example is a pair of an object and a class to which the object should belong. The acquired rule is used to classify an unseen object into its proper class. A typical technique for this task in the machine learning field is ID3 (Quinlan, 1986), and the task is often called discriminant analysis or pattern recognition.

Clustering, on the other hand, is a task that divides a given object set into clusters that have the properties of internal cohesion and external isolation (Everitt, 1993). The minimum distance or the k-means algorithm is a typical clustering method in the numerical taxonomy literature. In the machine learning literature, the task is often called *learning by observation* or *unsupervised learning*, and COBWEB (Fisher, 1987) and AutoClass (Cheeseman & Stutz, 1965) are typical examples of such a learning system.

We have been developing *learning from cluster examples* techniques (Kamishima, Miroh, & Ikeda, 1995) as an extension of these two known approaches. The aim is not to find a rule to classify single objects, or a particular clustering, but to find a rule for partitioning based on a given example set. Each example in the training example set is a pair consisting of a set of objects accompanied by a true partition (a set of clusters) for that set. The acquired rule produced by learning from this example set is used to estimate a true partition for an unseen object set. Thus, in contrast to learning from examples, which attempts to find a rule to classify one object, LCE involves the acquisition of a rule for partitioning an object set. Although the aim of clustering is to partition an object set, the partitioning rule produced by LCE is then applicable to any object set from the same domain. In short, LCE takes the supervisory nature of learning from examples, and brings it to the task of clustering.

What must be noted here is the difference between supervisory information for a learning from examples task and that for a LCE task. Supervisory information for a learning from

examples task is specified by selecting a proper class from a predefined set of classes. On the other hand, in the case of the LCE task, only supervisory information regarding which objects should be grouped is provided, and there is no notion of a predefined set of classes.

2.2. *Why LCE is important*

We will now describe some examples that fit the LCE model. Typically, these will be examples in which a true partition for any object set is easy for a user to specify or identify, while an overall set of rules for finding these partitions is very difficult for a user to specify concretely and explicitly. A prime example of such a problem is image segmentation.

To explain the image segmentation task, we give an example of a typical problem involving the understanding of diagrammatic images. Figure 1(a) shows an image of a logic circuit diagram in vector-data form. The vector-data is a form that represents images by collections of line-segments. In the understanding process for this image, a set of line-segments composing images are first divided into a partition, so that each cluster depicts a primitive symbol. This operation is generally called *segmentation* in the machine vision literature, and is a very common technique. An appropriate treatment of the image in figure 1(a), for example, would be to partition it into clusters with each depicting one part of a logic circuit diagram. Such a partition is illustrated in figure 1(b), where the the original image has been separated by thin broken lines.

Consider the performance of this segmentation task by applying a clustering technique. The input to the clustering algorithm is a set of feature values of line-segments; for example, their length or the connectivities among them. Then, to apply a clustering algorithms, one also has to specify knowledge, such as a potential function or a dissimilarity measure. It is, unfortunately, difficult to specify such knowledge, because it is hard to detect intuitive correspondence between feature values and a notion of logic diagram parts. Even in such a case, it is easy to give the desired partitions themselves. We therefore suppose that it is useful to derive partitioning rules from partition examples. This is why we consider the LCE task important.

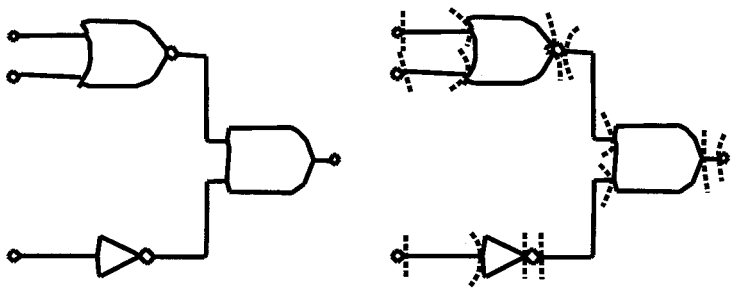


Figure 1. Examples of logic circuit diagram images in a vector-data form.

The development of successful techniques for LCE will contribute to the progress of research in any field in which the partitions that should be derived are clear, but rules to derive such partitions are not. The technique may also be applicable to problems such as knowledge transmutation by agglomeration in multistrategy learning (Michalski, 1993), and the identification of coding regions in DNA (Burset and Guigó, 1996). The rest of this paper presents our algorithm for LCE.

2.3. Formalization of LCE

This section formally states the task of learning from cluster examples. This task can be visualized as in figure 2, and consists of two major stages: a learning stage and a partitioning stage. In the learning stage (figure 2, left), the rule for partitioning is acquired from a training example set. In the partitioning stage (figure 2, right), based on the acquired rule, the true partition of an unseen object set is estimated.

Previously herein, for simplicity, we have described an example as a pair of an object set and a true partition for that set. Strictly speaking, an object set should be substituted by an object set description, $(O_I, F(O_I))$, which is an object set accompanied by its features.

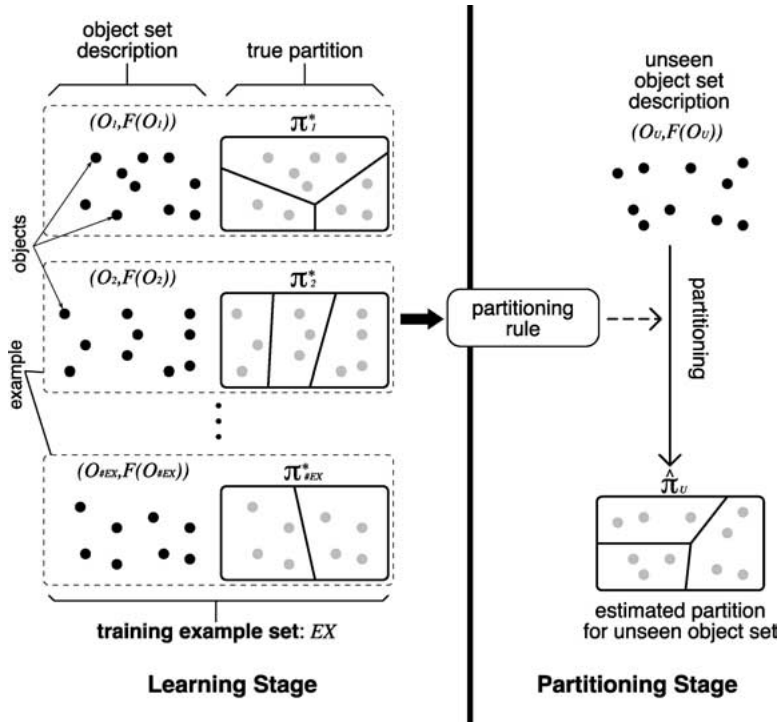


Figure 2. An illustration of learning from cluster examples.

We hereafter strictly define two distinct terms: the object set description and the object set. An object set, O_I, is a set of entities that will be grouped, and includes $\#O_I$ objects; $\{o_I^1, o_I^2, \ldots, o_I^{\#O_I}\}$. The object set features, $F(O_I)$, is a formal representation of an object set. We will describe the details of this later.

Each example is a pair of an object set description and its true partition, π_I^*, and is denoted by $\langle (O_I, F(O_I)), \pi_I^* \rangle$. An example set, EX, includes $\#EX$ examples:

$$\{\langle (O_1, F(O_1)), \pi_1^* \rangle, \langle (O_2, F(O_2)), \pi_2^* \rangle, \ldots, \langle (O_{\#EX}, F(O_{\#EX})), \pi_{\#EX}^* \rangle\}.$$

The partition, π_I, includes $\#\pi_I$ clusters; $\{C_I^1, C_I^2, \ldots, C_I^{\#\pi}\}$, and the cluster C_I^J is a subset of O_I, such that these clusters are mutually disjoint and every object is an element of exactly one cluster.

A common formal representation of an object within an object set is the attribute vector. For our LCE technique, we use a generalized form of this, and call it the *composite attribute vector form*. In the composite attribute vector form, object set features are described by the following three types of attribute vectors.

Attributes of Objects: $A(o)=(a^1(o), a^2(o), \ldots, a^{\#A(o)}(o))$

This type of attribute, assigned to each constituent object, is the same one adopted in the k-means algorithm and many other learning algorithms. In the task shown in figure 1, for example, the length or angle of each line-segment can be represented by this type of attribute. We denote the attributes of the object o by $A(o)$. $A(o)$ consists of $\#A(o)$ attributes. $\#A(o)$ is invariant across all objects.

Attributes of Object Pairs: $A(p)=(a^1(p), a^2(p), \ldots, a^{\#A(p)}(p))$

To apply the minimum distance clustering technique, for example, an object set is represented by a dissimilarity matrix, whose elements are dissimilarities between two constituent objects. We generalize this idea so that object pairs have descriptive attributes. Each entry in the dissimilarity matrix is an attribute that represents a relationship between two objects. In the task in the figure 1, the difference in angles or the connectivity between two line-segments can be represented by this type of attribute. Specifically, let a pair of objects o^i and o^j be denoted by p^{ij}, and let P be the set of all possible pairs of objects, so that P has $\#P(=\#O(\#O+1)/2)$ pairs. We denote the attributes of pairs p by $A(p)$. $A(p)$ consists of $\#A(p)$ attributes.

Attributes of Partitions: $A(\pi)=(a^1(\pi), a^2(\pi), \ldots, a^{\#A(\pi)}(\pi))$

The values of the above two types of attributes are fixed, once an object set is specified. On the other hand, the values of a third type of attribute are not fixed until some partition is specified along with an object set. That is to say, attributes of partitions are given as functions that output attribute values if any partitions are given. This type of attribute represents characteristics of entire partitions. In the task in the figure 1, for example, numbers of diagram parts can be represented. We denote the attributes of partitions by $A(\pi)$. $A(\pi)$ consists of $\#A(\pi)$ attributes.

In summary, object set features, $F(O)$, are described by using sets of the above attribute vectors, $\{A(o)\}$, $\{A(p)\}$, and $A(\pi)$, where $\{A(o)\}$ and $\{A(p)\}$ are sets of all value vectors of constituents in O and P, respectively.

Owing to restrictions of our current implementation, there are limitations on domains of attributes upon application of our LCE implementation. The attributes of objects and of object pairs must take as their domains either continuous numbers or discrete values (as in Quinlan's ID3 (Quinlan, 1986)). The attributes of partitions must be expressed as real numbers from the domain [0, 1].

3. The partitioning method

Although a partitioning rule is first learned and then the rule is applied for partitioning, a clear description of our method is best achieved by first establishing what it actually means to produce a good partition. We therefore begin by describing our partitioning method. As described above, the partition, $\hat{\pi}_U$, is estimated by applying the rule acquired at the learning stage, when given the unseen object set description, $(O_U, F(O_U))$. For simplicity, the subscript U is omitted in this section.

We advocate the LCE-Maximum A Posteriori (LCE-MAP) approach for estimation of partitions. Let π be an arbitrary partition for an unseen object set, and $\pi = \pi^*$ be an event such that π is exactly the true partition. As the most plausible partition, the LCE-MAP approach adopts the maximum a posteriori estimator; namely, the partition that maximizes the joint probability of $\pi = \pi^*$ and a given object set feature. The joint probability is:

$$\text{Joint Probability: } \Pr[\pi = \pi^*, \{A(o)\}, \{A(p)\}, A(\pi)]. \tag{1}$$

The Joint Probability (1) is hard to calculate directly because it is composed of so many elements. We therefore decompose it into the product of the three terms:

$$\Pr[\{A(o)\}, \{A(p)\}], \tag{2}$$

$$\text{Prior Probability: } \Pr[\pi = \pi^* \mid \{A(o)\}, \{A(p)\}], \tag{3}$$

$$\Pr[A(\pi) \mid \pi = \pi^*, \{A(o)\}, \{A(p)\}]. \tag{4}$$

In Eq. (4), we make the assumption that value vectors $A(\pi)$ and $\{A(o)\}$, $\{A(p)\}$ are conditionally independent given $\pi = \pi^*$. This assumption is not always true, but it is feasible to choose attributes that are as independent as possible. We rewrite this equation simply as the probability density:

$$\text{Partition Density: } \Pr[A(\pi) \mid \pi = \pi^*]. \tag{5}$$

Since Eq. (2) is constant for any choice of partition, this term can be ignored in terms of maximization of the Joint Probability (1). We therefore describe the procedure to calculate the other two terms: the Prior Probability (3) and the Partition Density (5). Figure 3 illustrates an overview of the procedure. The input is the object set description, that is an object set, O, and its feature, $F(O)$ (bottom of the figure). At Step A1 in the figure, O is partitioned into π. The Prior Probability (3) and the Partition Density (5) are then calculated through procedures B1–B4 and C1–C2, respectively. Finally, these terms are multiplied, and the product is proportional to the Joint Probability (1).

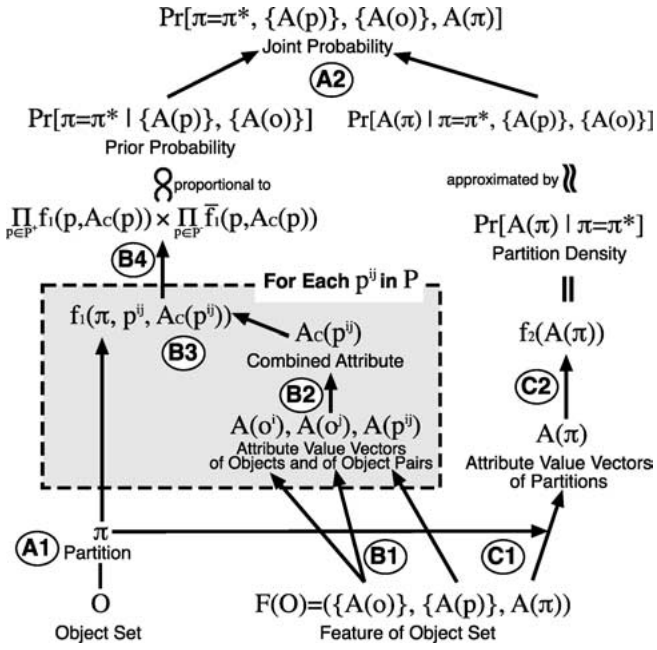


Figure 3. An overview of the calculation procedure of the Joint Probability (1).

To calculate Prior Probability (3), we show how it can be manipulated into a more manageable form through Steps B1–B4. For each p^{ij} in P (a set of all the object pairs), repeat Steps B1–B3. At Step B1, pick up three value vectors, $A(o^i)$, $A(o^j)$, and $A(p^{ij})$, from $\{A(o)\}$ and $\{A(p)\}$. These three vectors are combined into one vector, $A_C(p^{ij})$, at the next step (B2) by the method shown in Section 3.1. At Step B3, p^{ij} and $A_C(p^{ij})$ are applied to the function

$$f_1(p^{ij}, A_C(p^{ij})) = \mathtt{Pr}[\mathsf{in}(p^{ij}, \pi^*)=1 \mid A_C(p^{ij})], \tag{6}$$

where the function $\mathsf{in}(p^{ij}, \pi)$ is 1 if both o^i and o^j are in the same cluster of the partition π, and 0 otherwise. The function f_1 outputs the conditional probability of the event that two objects o^i and o^j are in the same cluster of the true partition given $A_C(p^{ij})$. The learning procedure for the function will be explained in Section 4.1. After the $\#P$ values of $f_1(p^{ij}, A_C(p^{ij}))$ are derived, these values are multiplied as follows:

$$\prod_{p\in P^+} f_1(p, A_C(p)) \times \prod_{p\in P^-} \bar{f}_1(p, A_C(p)), \tag{7}$$

where P^+ is a subset of P consisting of all the pairs satisfying the condition $\mathsf{in}(p, \pi) = 1$, P^- is its complementary set, and $\bar{f}_1(p, A_C(p)) = 1 - f_1(p, A_C(p))$. It can be shown that the Eq. (7) is proportional to the Prior Probability (3). We will give the rationale for this in the Section 3.2.

We turn now to the calculation of the Partition Density (5), achieved through Step C1 and C2. At Step C1, derive the value vectors of $A(\pi)$ when the O is partitioned into the π. At the next step (C2), the Partition Density (5) is calculated by the function

$$\text{Partition Density: } f_2(A(\pi)) = \mathtt{Pr}[A(\pi)|\pi = \pi^*].$$

This function f_2 outputs the conditional probability density of the attribute value vectors of $A(\pi)$ given $\pi = \pi^*$, an event π is exactly the true partition. We will describe the learning method for the function in Section 4.2.

Consequently, to achieve our overall goal of maximizing the Joint Probability (1), all we have to do is to maximize the product of Eq. (7) and the Partition Density (5):

$$\text{Criterion: } f_2(A(\pi)) \times \prod_{p \in P^+} f_1(p, A_C(p)) \times \prod_{p \in P^-} \bar{f}_1(p, A_C(p)). \tag{8}$$

We then describe our procedure to search for the most plausible true partition; that is, achieving the maximum of Criterion (8). According to the literature (e.g., Everitt, 1993), the number of possible partitions is

$$\sum_{j=1}^{\#O} \left(\frac{1}{j!} \sum_{i=0}^{j} (-1)^{j-i} \binom{j}{i} i^{\#O} \right),$$

and this number increases exponentially according to the number of objects. Therefore, finding the optimal partition is not tractable in general, and we rely on the greedy search algorithm in figure 4 to find a partition that may be locally optimal. In this algorithm, an initial partition is iteratively changed by applying modification operations. In each iteration, the operation that maximizes Criterion (8) is applied. This iteration stops when no operation improves Criterion (8).

The details of this algorithm are shown in figure 4. In the figure, Eq7(π) and PD5(π) denote the values of Eq. (7) and Partition Density (5) when O is partitioned into π, respectively. The algorithm begins by creating an initial partition whose constituent clusters are made up of only one object, then refines this partition so as to maximize Criterion (8). This refinement is done by applying two types of operations: a *merge*, which merges a pair of clusters, and a *move*, which moves one element from one cluster to another. When no partition that achieves a larger value of Criterion (8) is found, this algorithm stops, then outputs the current partition as the most plausible true partition. Note that the basic role of the procedure EVALUATION is to calculate a value of Criterion (8). The value is used to compare the current best partition with the new partition generated by applying a merge or a move operation to the former. The EVALUATION procedure treats separately the condition where Partition Density (5) has been zero from the beginning of the algorithm, because the product becomes zero even if the value of Eq. (7) is non-zero. Therefore, while this condition holds (The flag, f, in the figure holds this condition), EVALUATION simply returns the value of Eq. (7), and to avoid infinite loop, the moving operation is not applied. Once a partition for which Partition Density (5) is not zero is found, this special case is no longer invoked.

```
the procedure MAIN
t := 0, π^0 := {C = {o}, ∀o ∈ O}        — create an initial partition
if (PD5(π^0) > 0) then {
    f := true, E^0 := PD5(π^0) × Eq7(π^0)
} else {
    f := false, E^0 := Eq7(π^0)
}
start:
    t := t + 1, E^t := E^{t-1}
    forall (C^A ∈ π^{t-1}, C^B ∈ π^{t-1}, C^A ≠ C^B) {
                                        — check for applying the merge operation
        π' := π^{t-1} − C^A − C^B + {C^A ∪ C^B}, call EVALUATION(π')
    }
    if (f = true) {
        forall (C^A ∈ π^{t-1}, C^B ∈ π^{t-1}, C^A ≠ C^B) {
                                        — check for applying the move operation
            forall (o ∈ C^A) {
                π' := π^{t-1} − C^A − C^B + {C^A − {o}} + {C^B ∪ {o}}
                call EVALUATION(π')
            }
        }
    }
    if (f = false ∨ E^t ≠ E^{t-1}) then goto start
                                        — continue until improvement is not feasible
output π^{t-1}
end

the procedure EVALUATION(π')
if (f = false) then {
    if (PD5(π') ≠ 0) then {             — check PD5 is non-zero
        f := true, π^t := π', E^t := PD5(π') × Eq7(π')
    } else if (Eq7(π') > E^t) then {
        π^t := π', E^t := Eq7(π')       — ignore PD5 if PD5 is zero
    }
} else if (PD5(π') × Eq7(π') > E^t) then {
    π^t := π', E^t := PD5(π') × Eq7(π')
}
return
```

Figure 4. Our algorithm for searching an true partition.

3.1. Methods to combine attribute vectors

We here describe the method to combine three value vectors, $A(o^i)$, $A(o^j)$, and $A(p^{ij})$, into $A_C(p^{ij})$. This method is used in Step B2 of figure 3, and in the learning process for $f_1(p^{ij}, A_C(p^{ij}))$.

Our combination method is defined so as to be invariant under the ordering of indices, so that the value of the combined attribute $A_C(p^{ij})$ is always equal to $A_C(p^{ji})$. To produce such combined vectors, we copy all the values of $A(p^{ij})$ into the top of the combined vector. Additional elements are then concatenated to this combined vector by considering, one by one, the elements of the original vectors $A(o^i)$ and $A(o^j)$. The s-th elements of these original vectors, $a^s(o^i)$ and $a^s(o^j)$, are merged and added to the combined vector according to the following rules:

– If these two s-th elements take continuous values, the smaller value is added as an element of the combined vector, and the larger value is added as the subsequent element. That is, if the smaller value is added to the combined value as the t-th element, the larger value would be added as the $(t + 1)$-th element.
– If these two s-th elements take discrete values, the values are merged into one and added into the combined vector. If the number of possible values for the original attribute is d, the merged attribute can take one of the possible $d(d + 1)/2$ values. For example, if the possible values are "yes" and "no", the merged value can take one of the values "yes–yes", "yes–no", or "no–no".

As an example, consider the value vector $A(p^{ij})$ with two elements, the first discrete and the second continuous, and the vectors $A(o^i)$ and $A(o^j)$, both of which have two elements, the first continuous and the second discrete. Given the attribute values $A(p^{ij}) = (\mathsf{yes}, 100)$, $A(o^i) = (50, \mathsf{yes})$, and $A(o^j) = (10, \mathsf{no})$, the combined attribute $A_C(p^{ij})$ would be $(\mathsf{yes}, 100, 10, 50, \mathsf{yes–no})$.

3.2. *Why Eq. (7) is proportional to Prior Probability (3)*

In this section, we explain why Eq. (7),

$$\prod_{p \in P^+} f_1(p, A_C(p)) \times \prod_{p \in P^-} \bar{f}_1(p, A_C(p)), \tag{7}$$

is proportional to Prior Probability (3),

$$\Pr[\pi = \pi^* | \{A(o)\}, \{A(p)\}]. \tag{3}$$

We first give an intuitive explanation. Eq. (7) expresses the probability assigned to a necessary condition of $\pi = \pi^*$, an event that true partition π^* is equivalent to the partition π. The condition is that any object pairs belonging to the same cluster of π^* must also belong to the same cluster of π, and vice versa. Therefore, in Eq. (7), $f_1(p, A_C(p))$ is multiplied if $\mathsf{in}(\pi, p) = 1$, otherwise $\bar{f}_1(p, A_C(p))$ is multiplied. However, this is not independent on whether object pairs belong to the same cluster. For example, if p^{ij} and p^{ik} are in the same cluster, p^{jk} must be in the same cluster. Since some probability is assigned to such impossible events in the formulation of Eq. (7), the equation has to be divided by a proper normalization term in order to make the total sum of probabilities equal to one. However, since this term is, fortunately, constant for any choices of π, Eq. (7) is consequently proportional to Prior Probability (3).

Equation (7) is a product of conditional probabilities given distinct events, and so it does not strictly express the probability of the above necessary condition. To compensate for this shortcoming, we then give a more sophisticated rationale, based on Dempster & Shafer's rule of combination (Shafer, 1976) (*DS rule*, for short). We will describe this DS rule before moving to a full explanation.

The DS rule is used for combining probabilities based on different pieces of evidence. Let e be an event, E_a be an event set, and E_{All} be the set of all possible events. Let $P(E_{\mathrm{All}})$

be the power set of E_{All} (i.e., $\{E_a : {}^{\forall}E_a \subseteq E_{\mathrm{All}}\}$), and A_x be an evidence. $\mathtt{Pr}[E_a(A_x)]$ denotes the probability that one of the events in E_a occurs based on the evidence A_x, and is called a *basic probability*. Basic probabilities satisfy these conditions:

$$\mathtt{Pr}[E_a(A_x)] \geq 0,\ \mathtt{Pr}[\emptyset(A_x)] = 0,\ \sum_{E_a \in P(E_{\mathrm{All}})} \mathtt{Pr}[E_a(A_x)] = 1.$$

For example, when there are two events, e_1 and e_2, $E_{\mathrm{All}} = \{e_1, e_2\}$, $P(E_{\mathrm{All}}) = \{E_1{=}\emptyset$, $E_2 = \{e_1\}$, $E_3 = \{e_2\}$, $E_4 = \{e_1, e_2\}\}$, and basic probabilities are assigned to each $E_1 - E_4$. Consider the simple case that there are two basic probability sets based on each of evidences A_1 and A_2, and each of these sets consists of basic probabilities of the events $E_1 - E_4$. According to the DS rule, the probability of the event set E_2 based on these two evidences is:

$$\mathtt{Pr}[E_2(\{A_1, A_2\})] = \frac{\mathtt{Pr}[E_2(A_1)]\mathtt{Pr}[E_2(A_2)] + \mathtt{Pr}[E_2(A_1)]\mathtt{Pr}[E_4(A_2)] + \mathtt{Pr}[E_4(A_1)]\mathtt{Pr}[E_2(A_2)]}{1 - (\mathtt{Pr}[E_2(A_1)]\mathtt{Pr}[E_3(A_2)] + \mathtt{Pr}[E_3(A_1)]\mathtt{Pr}[E_2(A_2)])}$$

(the probability assigned to $E_1 = \emptyset$ is omitted). The numerator of the above equation denotes the sum of all the basic probabilities such that it leads to E_2. For example, since the $E_2 \cap E_4$ exactly equals E_2, $\mathtt{Pr}[E_4(A_1)]\mathtt{Pr}[E_2(A_2)]$ and $\mathtt{Pr}[E_2(A_1)]\mathtt{Pr}[E_4(A_2)]$ are added to the numerator. Note that $E_4 \cap E_4$ includes E_2, but it does not exactly equal E_2, thus $\mathtt{Pr}[E_4(A_1)]\mathtt{Pr}[E_4(A_1)]$ is not added. On the other hand, the denominator is 1 minus the sum of all the basic probabilities that will lead to the empty set. For example, since $E_2 \cap E_3 = \emptyset$, $\mathtt{Pr}[E_2(A_1)]\mathtt{Pr}[E_3(A_2)]$ and $\mathtt{Pr}[E_3(A_1)]\mathtt{Pr}[E_2(A_2)]$ appear in the denominator.

Strictly speaking, the premises and semantics of the probabilities manipulated by the DS and the Bayesian theories are different. However, because it is well known that the DS theory can be considered as a generalization of the Bayesian theory, we introduce the DS theory. To introduce the DS theory, we consider that the basic probability based on an evidence is equal to a conditional probability, given the evidence. Grounded on this idea, Prior Probability (3) can be considered as a basic probability of an event $\pi = \pi^*$, based on the evidence that a set of attribute value vectors $\{A(o)\}$ and $\{A(p)\}$ are simultaneously observed. The function f_1 can also be interpreted as a basic probability of $\mathsf{in}(p^{ij}, \pi^*) = 1$, based on the evidence that the value vector $A_C(p^{ij})$ is observed. We can now derive the Prior Probability (3) by combining f_1s according to the DS rule. We describe the details of this combination scheme below.

First, Prior Probability (3) and the functions f_1 are probabilities assigned to different kinds of events; namely Prior Probability (3) is the probability of $\pi = \pi^*$, and f_1 is that of $\mathsf{in}(p, \pi^*) = 1$. We present the common set of events to which these two events can be converted.

Let Π_{All} be the set of all possible partitions. For example, consider the object set $O = \{o^1, o^2, o^3, o^4\}$. For this set, Π_{All} is the set of fifteen partitions shown in figure 5, where the objects in parenthesis form one cluster. We then denote a special partition set by Π_p: a set of all the partitions according to which the object pair p is in the same cluster. For example, $\Pi_{p^{12}}$ is $\{\pi^1, \pi^4, \pi^5, \pi^6, \pi^{14}\}$; all five partitions satisfy the condition that o^1 and o^2 are in the same cluster, and no other partitions can satisfy this condition. Furthermore,

$$
\begin{array}{lll}
\pi^1 = (o^1, o^2, o^3, o^4) & \pi^2 = (o^1)(o^2, o^3, o^4) & \pi^3 = (o^2)(o^1, o^3, o^4) \\
\pi^4 = (o^3)(o^1, o^2, o^4) & \pi^5 = (o^4)(o^1, o^2, o^3) & \pi^6 = (o^1, o^2)(o^3, o^4) \\
\pi^7 = (o^1, o^3)(o^2, o^4) & \pi^8 = (o^1, o^4)(o^2, o^3) & \pi^9 = (o^1)(o^2)(o^3, o^4) \\
\pi^{10} = (o^1)(o^3)(o^2, o^4) & \pi^{11} = (o^1)(o^4)(o^2, o^3) & \pi^{12} = (o^2)(o^3)(o^1, o^4) \\
\pi^{13} = (o^2)(o^4)(o^1, o^3) & \pi^{14} = (o^3)(o^4)(o^1, o^2) & \pi^{15} = (o^2)(o^3)(o^1)(o^4)
\end{array}
$$

Figure 5. An example of all the possible partitions for the object set: $\{o^1, o^2, o^3, o^4\}$.

let $\Pi_p = \pi^*$ be the event set such that one of the partitions in Π_p is the true partition. The $\Pi = \pi^*$ corresponds to the event set E_a of the above examples. We then interpret both of the events, $\mathsf{in}(p, \pi^*) = 1$ and $\pi = \pi^*$, as these event sets.

Let us focus on the function f_1. By definition, the event $\mathsf{in}(p^{ij}, \pi^*) = 1$ is equivalent to the event set $\Pi_{p^{ij}} = \pi^*$. Since $A_C(p)$ is derived from $A(o^i)$, $A(o^j)$, and $A(p^{ij})$, the function f_1 can be interpreted as:

$$
\begin{cases}
\mathtt{Pr}[\Pi_{p^{ij}} = \pi^*(\{A(p^{ij}), A(o^i), A(o^j)\})] \equiv f_1(p^{ij}, A_C(p^{ij})), \\
\mathtt{Pr}[\bar{\Pi}_{p^{ij}} = \pi^*(\{A(p^{ij}), A(o^i), A(o^j)\})] \equiv \bar{f}_1(p^{ij}, A_C(p^{ij})),
\end{cases} \tag{9}
$$

where $\mathtt{Pr}[\Pi_{p^{ij}} = \pi^*(\{A(p^{ij}), A(o^i), A(o^j)\})]$ denotes the basic probability of the event set $\Pi_{p^{ij}} = \pi^*$ based on the evidence $\{A(p^{ij}), A(o^i), A(o^j)\}$, and $\bar{\Pi}_{p^{ij}} = \Pi_{\mathrm{All}} - \Pi_{p^{ij}}$. We assign a probability of zero to any event set except for the above two. For example, let us focus on the evidence that $A(p^{12})$, $A(o^1)$, and $A(o^2)$ are observed. The basic probability set based on the evidence consists of 2^{15} probabilities. Among this probability set, there are two event sets to which non-zero probabilities are assigned,

$$
\begin{cases}
\mathtt{Pr}[\Pi_{p^{12}} = \pi^*(\{A(p^{12}), A(o^1), A(o^2)\}] = f_1(p^{12}, A_C(p^{12})) \\
\qquad\qquad \Pi_{p^{12}} = \{\pi^1, \pi^4, \pi^5, \pi^6, \pi^{14}\}, \\
\mathtt{Pr}[\bar{\Pi}_{p^{12}} = \pi^*(\{A(p^{12}), A(o^1), A(o^2)\}] = \bar{f}_1(p^{12}, A_C(p^{12})) \\
\qquad\qquad \bar{\Pi}_{p^{12}} = \{\pi^2, \pi^7, \pi^8, \pi^9, \pi^{10}, \pi^{11}, \pi^{12}, \pi^{13}, \pi^{15}\},
\end{cases} \tag{10}
$$

and there are $2^{15} - 2$ zero-probabilities. Such basic probability sets can be drawn for every pair in P (a set of all object pairs), and $\#P$ probability sets are thus constructed.

In contrast, the event $\pi = \pi^*$ corresponds to the event set, $\{\pi\} = \pi^*$, where the partition set consists of the π only.

Now, both Prior Probability (3) and the function f_1 are interpreted as the basic probability of common event sets. We then derive Prior Probability (3) by combining f_1s. By definition, observing $A(o^i)$, $A(o^i)$, and $A(p^{ij})$ for every p^{ij} in P is equivalent to observing $\{A(o)\}$ and $\{A(p)\}$ simultaneously. The Prior Probability (3) can therefore be derived by combining these probability sets based on $\#P$ distinct evidences. According the DS rule, the numerator should be a sum of the probabilities that are assigned to the combinations of event sets of which the intersection is just equal to $\{\pi\} = \pi^*$. To find such a combination, all we have to do is choosing either Π_p or $\bar{\Pi}_p$, which includes the target partition π for each p in P, because any event sets with zero-probability can be omitted. Since no other intersection

of combinations leads to $\{\pi\} = \pi^*$, the numerator of the combined probability becomes Eq. (7). For example, consider deriving the basic probability of the event set $\{\pi^4\} = \pi^*$. Since there are six object pairs, six probability sets are generated. The numerator of the combined probability assigned to the event set $\{\pi^4\} = \pi^*$ would be

$$f_1(p^{12}, A_C(p^{12}))f_1(p^{14}, A_C(p^{14}))f_1(p^{24}, A_C(p^{24})) \\ \times \bar{f}_1(p^{13}, A_C(p^{13}))\bar{f}_1(p^{23}, A_C(p^{23}))\bar{f}_1(p^{34}, A_C(p^{34})), \tag{11}$$

since $\{\pi^4\} = \Pi_{p^{12}} \cap \Pi_{p^{14}} \cap \Pi_{p^{24}} \cap \bar{\Pi}_{p^{13}} \cap \bar{\Pi}_{p^{23}} \cap \bar{\Pi}_{p^{34}}$.

On the other hand, the denominator of the combined probability has the useful property of being constant for any possible partition. This is because the combination of event sets of which the intersection leads to an empty set is independent with respect to choice of π. Note that such combinations leading to empty sets correspond to the impossible events described in the intuitive explanation given at the beginning of this section.

From what has been discussed above, we can conclude that Prior Probability (3) is proportional to Eq. (7).

4. The learning method

We present the method for acquiring the two functions $f_1(p, A_C(p))$ and $f_2(A(\pi))$ from the given example set, *EX*, in the learning stage.

4.1. Acquisition of the function $f_1(p, A_C(p))$

As described in Section 3, the function f_1 outputs a conditional probability of the event $\mathsf{in}(p^{ij}, \pi^*) = 1$ given $A_C(p^{ij})$. A *decision list* (Yamanishi, 1992) is used to express the function f_1. Briefly, a decision list can be defined as follows.

Let T be a term that is the conjunction of literals L. The literal L is a logical function that can take the binary values true or false when an attribute values a is given, as follows:

- for attributes a that take continuous values, the three possible forms of the literal are $(\theta_l \leq a)$, $(\theta_l \leq a < \theta_u)$, and $(a < \theta_u)$, where θ_l and θ_u are proper threshold values. Such a literal takes true whenever the value a satisfies the condition specified by the literal.
- for attributes a that take discrete values from some set V, a literal has the form $(a = v_1 \vee v_2 \vee \cdots \vee v_{d'})$, where $v_1, \ldots, v_{d'}$ are elements of the set V. This literal takes true whenever the value of a is one of $v_1, \ldots, v_{d'}$.

Decision lists are defined as a pairing of an ordered term list $\langle T_1, T_2, \ldots, T_{m-1}, \mathsf{true}\rangle$ and a probability list $\langle \mathtt{Pr}_1, \mathtt{Pr}_2, \ldots, \mathtt{Pr}_m\rangle$, where true is a term that always outputs true. Specifically, when the unseen value vector A_U is applied to a term list in the order $T_1, T_2, \ldots,$ true, if T_k is the first term that outputs true, the decision list outputs the value of the corresponding $\mathtt{Pr}_k$ as the conditional probability $\mathtt{Pr}[\mathsf{in}(p^{ij}, \pi^*) = 1 \mid A_C(p^{ij})]$. An example of a decision list is shown in figure 6. This decision list is designed for the attribute vector

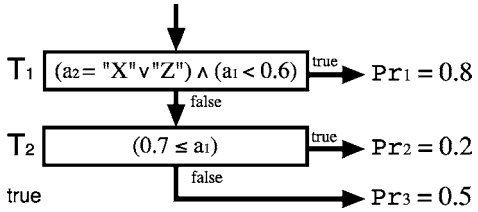


Figure 6. An example of a decision tree.

of which the first element is continuous and of the second is discrete. Suppose that the attribute vector (0.9, "X") is applied. The second attribute value satisfies the first literal of T_1, ($a_2 =$ "X" $\vee$ "Z"), but the first value does not satisfy the second literal, ($a_1 < 0.6$), thus the T_1 becomes false. The vector next applied to the second term, $T_2 = (0.7 \leq a_1)$, the term is then satisfied. Thus the $\Pr_2 = 0.2$ is outputted as the conditional probability.

We note here an explanation for adopting decision lists rather than decision trees. First, Pagallo and Haussler (1990) have pointed out that the size of the decision trees tends to increase drastically when the concept to be learned is disjunctive. Secondly, the size of the example set drastically decreases, since the decision trees are usually acquired by a so-called divide-and-conquer procedure and the example set is divided whenever a new node is created. This property weakens stochastic stability.

To acquire the function f_1, examples of pairs of an observed value vector and a target value ($A_C(p^{ij})$ and $\mathsf{in}(p^{ij}, \pi^*)$; a common format for the technique of learning from examples) are required. We therefore transform a given example set *EX* into a set of examples in this form. Each example is generated from an object pair in an object set from *EX*. Thus, the number of elements in the transformed example set is the sum of the object pairs in the training example set; i.e., $\#ex_1 = \sum_{I=1}^{\#EX} \#P_I$. We denote a transformed example by (A_x, c_x). The A_x is the combined attribute vector $A_C(p^{ij})$ where the objects o^i and o^j and object pairs p^{ij} are assumed to come from the same example $\langle (O_I, F(O_I)), \pi_I^* \rangle$. (The combination method is described in Section 3.1.) Here the class c_x takes the value $\mathsf{in}(p^{ij}, \pi_I^*)$, so the c_x becomes 0 or 1. The example set ex_1 can be simply represented by a form of $\{(A_1, c_1), (A_2, c_2), \ldots, (A_{\#ex_1}, c_{\#ex_1})\}$.

We next describe the algorithm to estimate the function f_1 from the transformed example set, ex_1. Our algorithm for acquiring the above decision lists is described in figure 7. This algorithm finds the most probable decision list based on Rissanen's MDL (Minimum Description Length) principle (Rissanen, 1983), which has been successfully adopted in learning from examples techniques (Quinlan & Rivest, 1989; Wallace & Patrick, 1993). This principle, which selects the best model from a given set of candidate stochastic models, is stated as "select the model in the observed data that permits the shortest encoding both of the observations and the model." Grounded on this principle, we formalize a set of stochastic models representing the conditional probability functions and specify a coding scheme for this set. In figure 7, we show the procedure for finding the decision list that permits the shortest code length, and the coding schemes of the decision lists are summarized in Appendix A. We make some remarks related to figure 7, below.

```
the procedure SEARCHING
example set S := ex_1, term no. i:=0
dicision list DL := ⟨⟩, conditional probability PR := ⟨⟩
do {
    i := i + 1
    the number of updating times j := 0
    j-th updated term T_i^j := true
    do {
        j := j + 1
        Let L_B be the literal maximizing the evaluation function and
            G_B be the function value for the L_B †
        if (G_B ≤ 0) then {
            T_i^j := T_i^{j-1}, goto term_end
        }
        T_i^j := T_i^{j-1} ∧ L_B
    } until(every classes of elements in S(T_i^j) is all 0 or all 1)
term_end:
    if (T_i^j = true) then goto list_end
    Add T_i^j to DL and Pr(S(T_i^j)) to PR ††
    S := S − S(T_i^j)
} until(every classes of elements in S is all 0 or all 1)
list_end:
Add true to DL and Pr(S) to PR

the procedure PRUNING
total code length ℓ := ℓ(ex_1, DL), the number of terms m := i + 1
while(m > 1) {
    S' := S(T_m) ∪ S(T_{m-1})
    DL' := ⟨T_1, ... , T_{m-2}, true⟩, PR' := ⟨Pr_1, ... Pr_{m-2}, Pr(S')⟩
    ℓ' = ℓ(ex_1, DL')
    if (ℓ ≤ ℓ') then goto noprune
    DL := DL', PR := PR', m := m − 1, ℓ = ℓ'
}
noprune:
output DL, PR
end
```

Figure 7. Our algorithm for searching decision lists.

This algorithm is composed of two procedures: SEARCHING and PRUNING. The former is the procedure for finding a decision list by repeatedly adding terms so as to achieve the shortest code length, then removing the examples satisfied by the list. In the latter procedure, the acquired decision list is polished.

We first discuss the evaluation function G_B and the literal L_B at the mark † in figure 7. This evaluation function is designed to find the term that is useful for achieving the shortest code length. Let $S(T_1)$ be the subset of the current example set S that consists of elements that satisfy the condition specified by the term T_1. Let $\#S(T_1)$ be the number of elements in $S(T_1)$, and $\ell(S(T_1))$ be the code length for $S(T_1)$. Assume two terms T_1 and T_2 that satisfy the condition $S(T_1) \supseteq S(T_2)$. If the condition

$$\frac{\ell(T_1) + \ell(S(T_1))}{\#S(T_1)} > \frac{\ell(T_2) + \ell(S(T_2))}{\#S(T_2)}$$

is satisfied, the evaluation function is

$$G(T_1, T_2) = (\ell(T_1) + \ell(S(T_1))) - (\ell(T_2) + \ell(S(T_2)) + \ell(S(T_1) - S(T_2))),$$

and otherwise 0, where $\ell(T_1)$ is the code length for T_1. The L_B is the literal that maximizes the evaluation function $G(T_i^{j-1}, T_i^{j-1} \wedge L)$ over all literals L that satisfy the condition $S(T_i^{j-1}) \supseteq S(T_i^{j-1} \wedge L)$. G_B is the output of the function at that time.

Next, we comment on $\mathtt{Pr}(S(T_i^j))$ calculated at the mark ††. Because we adopt the coding scheme of the example sets in Wallace and Patrick (1993), $\mathtt{Pr}(S(T_i^j))$ is defined as

$$\frac{\#S^+(T_i^j) + 1}{\#S(T_i^j) + 2},$$

where $S^+(T_i^j)$ is composed of the elements in $S(T_i^j)$ whose class labels are 1. Details about the code length of decision lists and example sets are shown in Appendix A.

4.2. *Acquisition of the function $f_2(A(\pi))$*

We next describe the method for acquiring the function f_2 that is the conditional probability density of $A(\pi)$ given the event $\pi = \pi^*$.

Before turning to an explanation of our algorithm, it is helpful to describe the *regression trees* (Breiman et al., 1984), since we employ them to represent the density function. An example of a regression tree is shown in figure 8. The tree in this figure has terminal and non-terminal nodes. Each non-terminal node, represented by a rectangle, has one threshold, one index that specifies which element of $A(\pi)$ should be compared, and two branches connecting it to other nodes. In addition, each terminal node, represented by a rounded rectangle, has a probability density value. When a fixed value vector $A(\pi)$ is given, the proper probability density value is found by recursively descending through the regression tree to a terminal node, as follows. First, the vector is compared to the threshold specified at the root node of the tree (for the specific index indicated at the node). If the value is smaller than the threshold, then the left branch of the node is descended. Otherwise, the right branch is descended. If the next node in the tree is also non-terminal, the process of comparing the specified attribute value and the threshold at the node is repeated until a

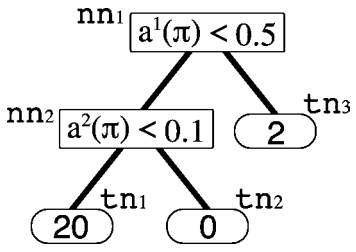


Figure 8. An example of a regression tree.

terminal node is reached. At a terminal node, the proper probability density value is simply the value specified by the node. For example, suppose that the vector (0.3, 0.5) is applied to the regression tree in figure 8. The first value of the vector is compared to a threshold of 0.5 at the root node (labeled nn_1). Since the value is smaller than the threshold, the left branch is traced and the node nn_2 is found. The node nn_2 is also non-terminal, so the second value of the vector is then compared to the threshold 0.1. As a result, we reach the terminal node tn_2. This gives a value of 0 as the target probability density 0.

We next describe the method for acquiring the regression trees expressing the function f_2. To derive the density function, it is required that the example set is composed of attribute value vectors. Therefore, we transform the original training example set, EX, into this form. Recall that the set EX is composed of examples of object set descriptions $(O_I, F(O_I))$ with their true partitions π_I^*. For each example, we calculate the value vectors $A(\pi_I^*)$ when O_I is partitioned into π_I^*. As a result, $\#EX$ vectors are derived, and we refer to the set of these vectors as the transformed example set, ex_2. Since each example of the ex_2 follows the probability density, $\mathtt{Pr}[A(\pi)|\pi = \pi^*]$, we can estimate the function $f_2(A(\pi))$ from the ex_2.

Now, all we have to do is estimating $\mathtt{Pr}[A(\pi)|\pi = \pi^*]$ from this ex_2. Our estimation algorithm is also grounded in the MDL principle. We describe a set of stochastic models and define a scheme for coding both of the models and the given example set. Then, as the function $f_2(A(\pi))$, we employ the model that permits the shortest code length.

We here present the coding scheme for the regression trees that are used for representing the target function. The code length for the structure of the regression tree equals the total number of nodes. The article (Quinlan & Rivest, 1989) presents a full explanation of the code length and of the coding scheme for the tree. For each non-terminal node, a threshold and an index at the node must be encoded. The threshold is encoded in the same scheme as that used for the threshold of the decision lists in Appendix A, and the code length for the index is $\log \#A(\pi)$. Note that the log is the logarithm with base 2, and that ln denotes the natural logarithm used in this paper. The scheme presented here makes it feasible to specify an arbitrary regression tree, RT, with code length $\ell(RT)$. Next, an example set, ex_2, must be encoded by using this regression tree. According to Yamanishi and Han (1992), the total code length is approximated by

$$\ell(ex_2, RT) = \ell(RT) + \left\{ -\log \mathcal{L}(ex_2 \mid RT) + \frac{1}{2}\#TN(\log e + \log \#ex_2) \right\},$$

where $\mathcal{L}(ex_2 \mid RT)$ is the likelihood, $\#ex_2$ is the number of examples in ex_2, and $\#TN$ is the number of terminal nodes in RT. Let TN be the set of all terminal nodes in the RT, and tn_x be its element. Let $\#\mathsf{tn}_x$ be the number of examples in ex_2 that reach the terminal node tn_x. $\mathcal{L}(ex_2|RT)$ is defined as

$$\mathcal{L}(ex_2 \mid RT) = \prod_{\mathsf{tn}_x \in TN} \mathtt{Pr}[\mathsf{tn}_x]^{\#\mathsf{tn}_x},$$

where $\mathtt{Pr}[\mathsf{tn}_x]$ is the probability density at the node tn_x, and is defined as:

$$\mathtt{Pr}[\mathsf{tn}_x] = \frac{\#\mathsf{tn}_x}{\#ex_2 \times V(R(\mathsf{tn}_x))}.$$

$R(\mathsf{tn}_x)$ is the region for a value vector such that if the vector range is $R(\mathsf{tn}_x)$, it reaches the terminal node tn_x, and $V(R(\mathsf{tn}_x))$ is the volume of $R(\mathsf{tn}_x)$. For example, in the case of node tn_2 in figure 8, any value vectors within the range $a^1(\pi) < 0.5$ and $a^2(\pi) \geq 0.1$ that are inputted to this regression tree would reach the node tn_2. So $R(\mathsf{tn}_2)$ is $(0 \leq a^1(\pi) < 0.5) \wedge (0.1 \leq a^2(\pi) \leq 1))$, and $V(R(\mathsf{tn}_2))$ is $0.45(= (0.5 - 0) \times (1 - 0.1))$. We here note why a tree style representation is used for the function $f_2(A(\pi))$, in light of remarking the advantages of a list type representation in Section 4.1. Since adopting a tree type of representation makes it possible to calculate $R(\mathsf{tn}_x)$s without referencing any other nodes, it is easier to calculate the $R(\mathsf{tn}_x)$s required only for the regression problem. In contrast, the list type representation has no such useful property.

In order to acquire the function $f_2(A(\pi))$, we must find the regression tree that permits the shortest total code length $\ell(EX_2, RT)$. For this purpose, we introduce the algorithm in figure 9. This algorithm adopts a divide-and-conquer strategy that recursively divides a

```
the procedure MAIN
ex_2 := {A(π_1), A(π_2), ... , A(π_{#ex_2})}: the example set
RT := (TN, NN): a regression tree whose root node is terminal
    TN := {tn_1}: a set of terminal nodes   (tn_1 is a root node)
    NN := {}: a set of non-terminal nodes
for s from 1 to #A(π) {
    δ^s := 6×⟨standard deviation of a^s(π_1), a^s(π_2), ... , a^s(π_{#ex_2})⟩/#ex_2
}

start:
RT_best := RT
foreach tn' in TN {
    for s from 1 to #A(π) {
        Let l and u be the lower and the upper bound
            of the s-th attribute of the region R(tn') respectively
        for d from 1 to ∞ {
            q := (1/2)^d
            if (q < δ^s) then goto checkend
            for t from 1 to 2^{(d-1)} {
                θ := q(2t - 1)
                if (l ≤ θ < u) then {
                    RT' := (TN', NN')
                        NN' := NN ∪ nn'
                        (nn' is the terminal node whose threshold is θ
                            and is placed at the position used to be tn')
                        TN' := {TN - tn'} ∪ {tn^R_new, tn^L_new}
                        (tn^L_new and tn^R_new are
                            the left and the right node of the nn' respectively)
                    if (ℓ(ex_2, RT') < ℓ(ex_2, RT)) then RT := RT'
                }
            }
        }
checkend:
    }
}
if (RT_best ≠ RT) goto start
end:
output RT_best
end
```

Figure 9. Our learning algorithm for searching regression trees.

given training example set. The initial regression tree consists of only one terminal node, and represents an uniform density function that is always at constant 1. The current tree is iteratively modified. This modification operation is as follows: One of the terminal nodes of the current tree is replaced with a new non-terminal node, and two new terminal nodes are added at the branches of the new non-terminal node. The replaced terminal and the new non-terminal node are selected so as to maximize $\ell(EX_2, RT)$. Finally, this algorithm stops when no improvement is feasible, and outputs the current tree.

5. Experiments

We have applied our technique of learning from cluster examples in two test domains: dot patterns and vector-data images. We below describe these domains, then present and discuss our test results.

5.1. Experimental domains and testing methods

First, we present what we will revealed from the results on each of the two experimental domains.

Object sets of the dot pattern domain (figure 10) are artificially generated. To partition the sets, we applied the EM algorithm, the clustering method dedicated to this domain. On application of the EM algorithm, we gave rich information that is helpful for partitioning, such as the correct number of clusters and a family of distribution. Partitions derived by the EM algorithm are compared with the one derived by the rule acquired by our LCE technique. This comparison never leads to the conclusion that one is superior to the other, since the clustering and the LCE are different kinds of tasks. What we want to show by this comparison is whether the information required for partitioning is available by the fully built-in rules acquired by our LCE technique. This will indicate whether the rules derived by the LCE technique are superior or equal to the EM algorithm with rich information for partitioning.

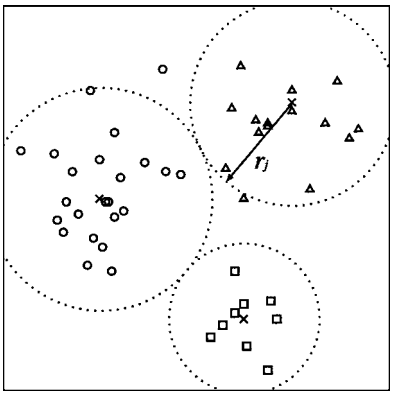
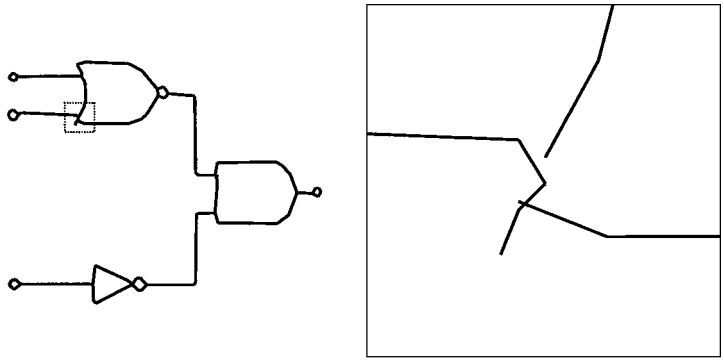


Figure 10. An example of a dot pattern.



(a) a whole image (b) an enlarged image

Figure 11. Examples of vector-data images.

Partitioning the vector-data image (figure 11) is a typical application of the LCE technique in a realistic domain, as described in Section 2.2. The difficulty of formalizing partitioning rules can be illustrated by examining some of the possible distortions in the images. For example, figure 11(b) is an enlarged image that depicts the highlighted part of the OR-gate symbol shown in figure 11(a). This figure reveals that there are several distortions (*spurs* and *gaps*) in the image. The presence of such imperfections significantly complicates the task of specifying concrete rules. Therefore, we consider the vector-image domain a good testbed of a realistic domain for examining the LCE technique.

5.1.1. Dot patterns. Segmentation of dot patterns is a basic problem domain for clustering. Dots are scattered in a 2-dimensional square space of fixed width and height as shown in figure 10. Here, all the dots from the same cluster are depicted by the same type of symbols. These dots are generated according to a Gaussian distribution (note that if any dot falls outside the square space in this generation process, it is discarded and a replacement is generated). For each region, the mean of the Gaussian distribution of its dots is at the center of the circle, and we use the following 2-dimensional Gaussian mixture distribution:

$$f(x, y) = \sum_{j=1}^{m} z_j N(x, y; \mu_j, \nu_j, \sigma_j),$$

$$N(x, y; \mu_j, \nu_j, \sigma_j) = \frac{1}{2\pi\sigma_j^2} \exp\left(-\frac{(x-\mu_j)^2 + (y-\nu_j)^2}{2\sigma_j^2}\right),$$

where m is the number of clusters, z_j specifies the ratio of mixing, μ_j and ν_j are the respective means of the x and y positions, and σ_j is a standard deviation. This standard deviation differs depending on the type of example set. We prepared three types of example sets with varying degrees of overlap between the different clusters. To create each set, we first randomly generated a value for m between 2 and 4, and created n random points within

the square space. We then created circular regions of radius r_j, $1 \leq j \leq m$ around each of these points, by generating r_j randomly under the constraint that each of the resulting circles must touch at least one other (in figure 10, these regions are depicted by dotted lines). Depending on the type of example set we wanted to create, we then assigned σ_j to be either $\sigma_j = r_j/3.0$ (for *separated* example sets), $r_j/2.5$ (for *touching* example sets), or $r_j/2.0$ (for *overlapping* example sets). Note that we force the covariance to be zero; namely, the x and y deviations are equal, since the dots follow such distributions in each type of example set.

Our three example sets each contain 100 examples. Each object set of an example set contains 50 dots comprising two to four clusters. The outline of our procedure to select attributes adopted for deriving final results is as follows: First, we implement a set of attributes of objects and of object pairs that have been used for clustering of dot patterns, and select a set of the candidate attributes shown in Table 1 (the details are described in Appendix B). This preliminary selection is guided by statistical criteria and our subjective consideration. For example, we discard the attribute, "distance to the 59th nearest dot," since no remarkable merit is not observed and it is correlated with the third or fourth attributes of objects. Second, we select attributes of partitions from the candidate attributes shown in Table 2 (these are basic statistics of $\#C_I$'s). We then generate all possible combinations

Table 1. Candidate attributes for dot patterns.

Attributes of objects	Attributes of object pairs
1. X-position	1. distance between two dots
2. Y-position	2. The smaller position index number in ascending distance order
3. Distance to the k-th nearest dot	3. The larger position index number in ascending distance order
3. Distance to the nearest dot	4. k-th nearest factor
	5. Gabriel Graph factor
	6. Relative Neighborhood Graph factor
	7. Length of the longest edge on its MST path
	8. The number of edges on the MST path joining the dots

Note: All attributes are adopted for the final attribute set. The details are described in Appendix B.

Table 2. Candidate attributes of partitions.

1. The number of partitions	$\#\pi/\#O$
2. The square mean of objects in clusters	$(1/\#O^2\#\pi)\sum_{C_I\in\pi}\#C_I^2$
3. The cubic mean number of objects in clusters	$(1/\#O^3\#\pi)\sum_{C_I\in\pi}\#C_I^3$
4. The minimum number of objects in clusters	$(1/\#O)\min_{C_I\in\pi}\#C_I$
5. The maximum number of objects in clusters	$(1/\#O)\max_{C_I\in\pi}\#C_I$

Note: $\#O$, $\#\pi$, and $\#C_I$ represent the numbers of objects, clusters, and objects in the cluster C_I, respectively. 1st and 3rd attributes are adopted for attributes of dot patterns, and 1st, 2nd, and 5th attributes are adopted for attributes of vector-data images.

of candidate attributes. By using each of these combinations together with all attributes in Table 1, we applied the test method in Section 5.2 on the above example sets. We thus selected the combination that consists of the 1st and the 3rd candidate attributes, since this leads the least error. Finally, we screen the candidate attributes of objects and of object pairs in Table 1. We generate all possible attribute sets such that one attribute is missing. By using each of these attribute sets together with the above best attribute combination of partitions, we again applied the test method on the above example sets. For each attribute set, we calculate statistics for testing the improvements against the attribute set consisting of all attributes. Since no attribute set does not lead statistically significant improvement for all three example sets, we adopt all the attributes of objects and of object pairs shown in Table 1. Consequently, we adopt four attributes of objects, eight attributes of object pairs, and two attributes of partitions. Note that the example sets used for the attribute selection should be separated from the ones used for final learning, since this might underestimate quantity of errors. We could not do so owning to the limited size of the example sets and computational resources. However, we suppose that such underestimation is very small, because we adopt very strict leave-one-out test, that derives fairly accurate estimation of errors for unseen object sets.

To provide a comparison for our LCE technique, we also applied the following EM algorithm (a common clustering technique) (Dempster, Laird, & Rubin, (1977) to the task of partitioning the dot patterns. Let n be the number of observed objects and (x_i, y_i) be the position of the i-th object. The EM algorithm leads to parameters that maximize the log-likelihood:

$$\log \mathcal{L}(x_1, \ldots, x_n, y_1, \ldots, y_n) = \sum_{i=1}^{n} \log f(x_i, y_i).$$

After the parameters are estimated, each object is classified into the k-th cluster:

$$k = \underset{j=1,\ldots,m}{\operatorname{argmax}} \; z_j N(x_i, y_i; \mu_j, \nu_j, \sigma_j).$$

The initial conditions we used for the EM algorithm parameters were as follows. First, we assumed that the correct number of clusters $\#\pi^*$ was explicitly given as m, then we initialized all the σ_js to $S/6$ (S is the width or height of the 2-dimensional square space) and all the z_j to $1/\#\pi^*$. As an initial guess at the actual clustering, we assumed that the means of the clusters were equi-distantly placed on a circle of radius $0.3 \times S$ in the center of the space.

5.1.2. Vector-data images. Vector-data images are often used in the process of diagram image understanding. A vector-data image represents objects as line-segments, and is typically used to represent images drawn with thin lines, such as diagrams or maps. A line-segment is a straight line connecting two end-points. Partitioning this type of object set is a more realistic task than that of dot pattern partitioning.

We generated our example set of vector-data images by transforming handwritten logic circuit diagrams. Handwritten diagrams were scanned by an image scanner, and the common

Table 3. Candidate attributes for vector-data images.

Attributes of objects	Attributes of object pairs
1. X coordinate of mid-point	1. Connection information
2. Y coordinate of mid-point	2. Difference of angles
3. Difference of X coordinate between end-points	3. Shortest distance between end-points
4. Difference of Y coordinate between end-points	4. The smaller position index number in ascending distance order
5. The number of line-segments in the arc including the line-segment to which attribute is assigned	5. The larger position index number in ascending distance order
6. Standard deviation of the lengths of the line-segment in the arc	6. Distance between mid-points
7. Standard deviation of the angles of the line-segments in the arc	7. Whether two line-segments are in the same arc
8. Sum of the lengths of the line-segments in the arc	

Note: 1st, 2nd, and 5th attributes of objects are NOT adopted for the final attribute set. The details are described in Appendix C.

image processing techniques of thinning and vectorization were then applied. The original handwritten diagrams consisted of five kinds of parts: AND-gates, OR-gates, buffers, terminals, and connecting lines. An example of a vector-data image of a logic circuit diagram is shown in figure 11. The segmentation task is then to divide these vector-data images into clusters such that each cluster consists of line-segments whose origins are the same diagram component.

Our example set of logic circuit diagrams consists of 100 examples. The mean number of clusters and of objects are 16.7 and 102.9, respectively. The attribute selection procedure is the same as that for the dot pattern domain. The set of candidate attributes of objects and of object pairs are shown in Table 3 (the details are described in Appendix C). For this domain, we again use the candidate attributes of partitions shown in Table 2 at Section 5.1.1. As a result of screening, we adopt the the 1st, 2nd, and 5th attributes of partitions. We finally screen the candidate attributes of objects and of object pairs, and eliminate the 1st, 2nd, and 5th attributes of objects. We consequently select five attributes of objects, seven attributes of object pairs, and three attributes of partitions.

5.2. A testing method

Before discussing our experimental results, we present a testing method for determining whether true partitions are appropriately estimated using the acquired rule. The method is a cross-validation test that is commonly used for learning from examples. We have also created a quantitative measure for comparing the estimated partition with a true partition to test how closely the true partition has been estimated.

For a cross-validation test, a given example set is first split into two parts: a training example set and a testing example set. After acquiring a partitioning rule from the training example set, the rule is evaluated to determine how appropriately it can partition the object sets in the testing example set. To get a reliable measure, we adopt a *leave-one-out-test*, a

strict cross-validation test. The first example is picked from a given example set, and a rule is acquired from the rest of the example set. Then, for an object set in the picked example, a partition is estimated by using the acquired rule. The true partition is already specified in the picked example, and the similarities between the estimated and the true partitions are calculated. This process is repeated for each of the other examples in the example set. The mean of the similarities can then be used as a measure for the chance that any unseen object set will be partitioned appropriately.

We introduce *ratio of information loss* (RIL) as a similarity measure. This is also called the uncertainty coefficient in numerical taxonomy literature. The RIL is the ratio of the information that is not acquired to the total information required for estimating a correct partition. Another definition of the RIL is posterior entropy divided by prior entropy. Let Π^* be an event where an object pair is in the same cluster of the true partition π^*. The prior entropy, which is the mean of the information required for estimating the true partition, is

$$H(\Pi^*) = \sum_{s=0}^{1} \frac{N(s)}{\#P} \log \frac{\#P}{N(s)},$$

where $N(s)$ is the number of object pairs that satisfy the condition $\mathsf{in}(p, \pi^*) = s$. Note that the base number of the logarithm is 2 in this paper. Let $\hat{\Pi}$ be an event where a pair of objects are in the same cluster of the estimated partition $\hat{\pi}$. The posterior entropy, which is the mean of the information not acquired for correct estimation, is

$$H(\Pi^* \mid \hat{\Pi}) = \sum_{s=0}^{1} \sum_{t=0}^{1} \frac{N(s,t)}{\#P} \log \frac{N(0,t) + N(1,t)}{N(s,t)},$$

where $N(s,t)$ is the number of object pairs p that satisfy the condition $\mathsf{in}(p, \pi^*) = s$ and $\mathsf{in}(p, \hat{\pi}) = t$. Consequently,

$$RIL = \frac{H(\Pi^* \mid \hat{\Pi})}{H(\Pi^*)}.$$

The smaller the RIL becomes, the more appropriately a partition is estimated. It ranges from 0 to 1 and becomes 0 if and only if the two partitions are completely identical. That the RIL is bounded was one of our main reasons for selecting this measure. Other measures are also possible, such as the ratio of correctly partitioned object pairs, which is also used in the numerical taxonomy literature (Jain & Dubes, 1988; Rand, 1971). However, for the ratio of correctly partitioned pairs, the lower bound changes according to π^*, and the resulting scale normalization problem makes it difficult to use this as the basis for calculating the means of these similarities. As another example, the correlation coefficient is commonly used for the gene finding problem (the detection of coding regions in given DNA sequences) (Burset & Guigó, 1996). However, this coefficient becomes infinite when the denominator is zero. Though this circumstance can be dealt with by using approximations, using the RIL avoids this problem altogether.

5.3. Experimental results

Here we present and discuss our experimental results on the example sets of dot patterns and of vector-data images. First, we apply our LCE method and the benchmark EM algorithm to the example sets of dot patterns. Having established this application, we then apply our method to the more realistic example set of vector-data images.

5.3.1. Testing using dot patterns. The rows in Table 4 show the respective experimental results for the three example sets: separated, touching, and overlapping dot patterns. In the second and third columns of the table, we show the means (and standard deviations, in parentheses) of the RILs and of Rand's criteria based on the rules acquired by our LCE method and based on the EM algorithm, respectively. In the last column, we show paired t-values, which are measures to test whether the difference of means is statistically significant. We show the Rand's criteria (Rand, 1971), the ratio of correctly partitioned object pairs, only for reference. (The Rand's criteria is not fit for evaluating the results of LCE tasks because of the scale normalization problem, as pointed out in Section 5.2.)

As can be seen in Table 4, the partitions estimated by our LCE methods are comparable to those by the EM algorithm. The positive t-values indicate that the mean of the RILs in the case of our LCE method is smaller, but all values are less than $t_{0.99} = 2.365$, so the differences are not statistically significant. As noted in Section 5.1, in order to produce results for the EM algorithm we supplied significant amounts of information to assist in partitioning. Therefore, it is proper to conclude that our LCE method is successful in acquiring knowledge for partitioning solely from a given example set.

To investigate the effects of Partition Density (5), we carried out an additional test, using a simplified version of our method in which no attributes of partitions are employed. Specifically, we set the function $f_2(A(\pi))$ to be constant at 1. Table 5 shows how this simplified algorithm compares with our original method. The positive t-value in this table indicates an advantage of our original method, but the difference is not statistically significant. To investigate further aspects, we applied a measure that directly takes into account the effects of the 1st attribute of partitions, which represents the numbers of clusters. The measure is

Table 4. The experimental results (means and s.d.s of RILs and of Rand's criteria) derived by the rules acquired by our LCE method and by the EM algorithm from the dot pattern example sets.

	Our LCE method	EM algorithm	t-value
RIL			
Separated	0.069 (0.1463)	0.093 (0.1809)	$+1.271 < t_{0.99}$
Touching	0.159 (0.1729)	0.165 (0.2121)	$+0.280 < t_{0.99}$
Overlapping	0.369 (0.2326)	0.400 (0.2695)	$+1.117 < t_{0.99}$
Rand's			
Separated	0.986 (0.0387)	0.982 (0.0395)	
Touching	0.969 (0.0416)	0.967 (0.0543)	
Overlapping	0.917 (0.0692)	0.905 (0.0885)	

Table 5. The experimental results (means and s.d.s of RILs) derived by the rules that our original and simplified methods acquired from the dot pattern example sets.

	Original method	Simplified method	t-value
Separated	0.069 (0.1463)	0.067 (0.1496)	$-0.476 < t_{0.99}$
Touching	0.159 (0.1729)	0.162 (0.1753)	$+1.092 < t_{0.99}$
Overlapping	0.369 (0.2326)	0.371 (0.2321)	$+1.126 < t_{0.99}$

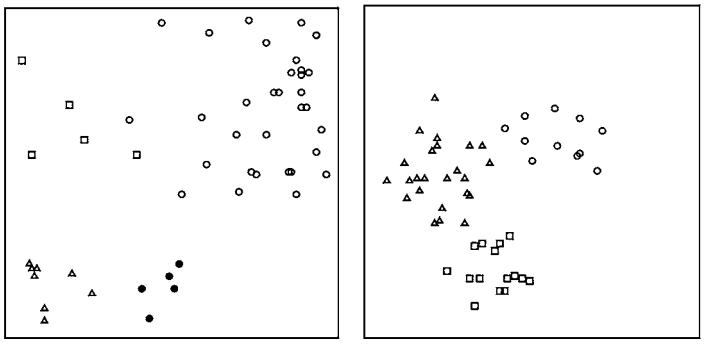
Table 6. The root mean squares of the error (the difference between the number of clusters in the estimated and the true partitions) for our original and our simplified methods.

	Original method	Simplified method	t-value
Separated	0.608	0.735	$1.4595 > F_{0.95}$
Touching	0.387	0.520	$1.8000 > F_{0.99}$
Overlapping	0.300	0.346	$1.3333 < F_{0.95}$

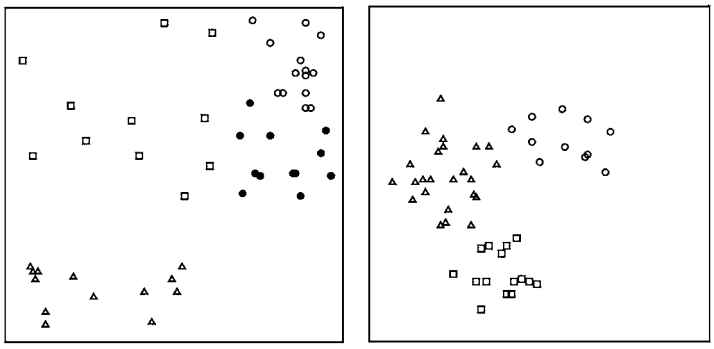
the root mean square of the error in the numbers of clusters (the difference between the number of clusters in the estimated and the true partitions). Table 6 shows how the simplified method compares with our original method on the dot pattern example sets when assessed with this measure. The F-ratios in the figure are the ratio of the mean square of the simplified method's estimations to that of the original method. It is known that this value follows F-distribution with $(n_1 - 1, n_2 - 1)$ degrees of freedom $(n_1 = n_2 = 100)$. Since the 99th- and 95th-percentiles for this distribution are 1.6015 and 1.3941, respectively, the difference between these root mean squares are statistically significant in the separated case at a significance level of 95% and in the touching case at a level of 99%. In the remaining overlapping case, the difference is very close to being significant. This demonstrates an advantage of adopting the 1st attribute of partitions, though the difference between RILs is not statistically significant.

For reference and to facilitate an intuitive comparison, in figure 12, we give an example of the partitioning of two sample dot patterns (both from the touching object sets). The true partitions are shown in figure 12(a). We selected these particular examples because they represent the most difficult data sets for the EM algorithm (left side of the figure) and our method (right side of the figure). That is, these are the examples for which the partitions produced by the algorithms had maximum RIL. Figure 12(b) and (c) show the actual partitions that were respectively derived by the EM algorithm and by the rules acquired by our LCE method. For this domain, where the EM algorithm can be given large amounts of information about the target position, it is evident that the each algorithm has its own strengths and weaknesses.

5.3.2. Testing using vector-data images. Next, we present the results for the example set of vector-data images. Table 7 shows the means and standard deviations of RILs produced by the EM algorithm, by the rule acquired by our original LCE method, and by the rules acquired by our simplified LCE method. The paired t-values for comparing our original

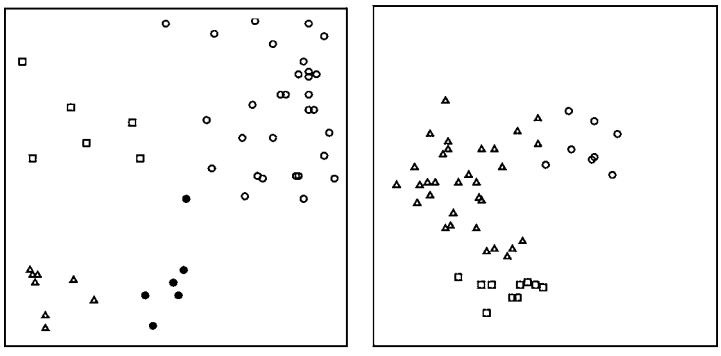


(a) True Partitions



$RIL = 0.908$ $RIL = 0.000$

(b) Partitions derived by The EM Algorithm



$RIL = 0.308$ $RIL = 0.758$

(c) Partitions derived by The Rules Acquired by Our LCE Method

Figure 12. The partition examples from the dot pattern example set.

Table 7. The experimental results (means and s.d.s of the RILs and of Rand's criteria) for the vector-data image example sets.

	EM algorithm	Original method	Simplified method	t-value
RIL	0.991 (0.0062)	0.398 (0.1329)	0.409 (0.1413)	$+3.484 > t_{0.99}$
Rand's	0.917 (0.0176)	0.974 (0.0123)	0.974 (0.0126)	

and simplified LCE methods. The means and standard deviations of Rand's criteria are also shown.

The EM algorithm applied here is almost the same as that applied to the dot-patterns except for the following four points: (1) We use five attributes of objects, described in Section 5.1.2. (2) The standard deviation of mixed Gaussian distributions is different for each set of attributes. (3) Initial parameters of Gaussian distribution are the means and standard deviations derived from the true partitions. (4) Initial mixture ratios are derived from the true partitions. The mean of the RILs derived by the EM algorithm is nearly 1, which indicates that only a small amount of information is gained. This result demonstrates that the EM algorithm is not fit for this segmentation task. This task, therefore, can be considered as a good testbed for our LCE method.

We then moved to the analysis of the results our LCE methods. In this case, the t-value gives us a clear, statistically significant result that the means of the original method's RIL are smaller. This result indicates the clear advantage of adopting Partition Density (5).

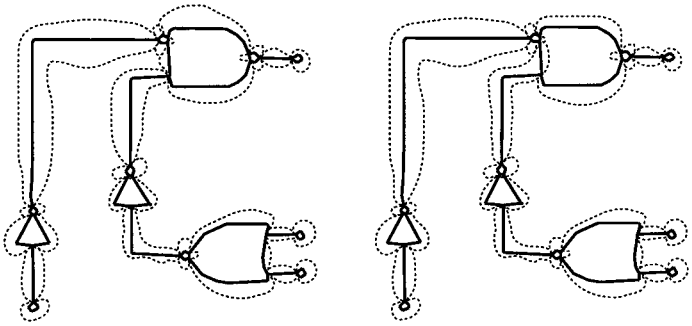
For both methods, the mean value of the RIL is larger than that found with the dot-pattern example set, but this is because the image segmentation problem is more realistic, and more difficult. Yet, even for this hard example set, the RIL between the estimated and true partitions shows that our method acquires 60% of the information required for complete partitioning.

As in the previous section, we offer several examples illustrating the performance of the partitioning algorithm. This time we choose the three examples representing the best, median, and worst results of our method. Figure 13 shows the true, then the estimated partitions for each example, together with the RIL of the estimated result. Note that we depict each cluster as a set of line-segments surrounded by a thin dotted line. In the best case, the true partition is almost perfectly estimated; only one line-segment is incorrectly grouped. On the other hand, in the worst case, the image is almost randomly partitioned. We suppose that this is an outlier, because the RIL of the second-worst case is 0.641 and the second-worst partition seems not so random. Because Criterion (8) is a function having very many local minima and our searching algorithm is greedy, a very bad partition is estimated infrequently.

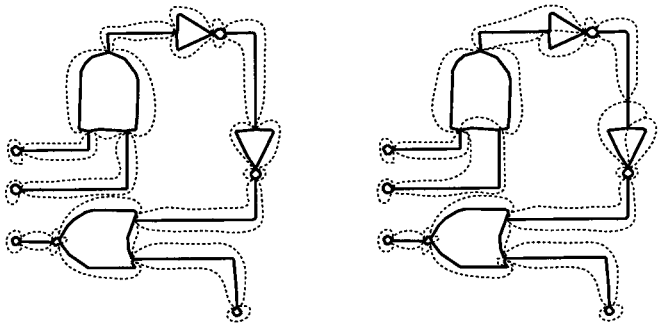
6. Discussions

In this section, we discuss the LCE task and our LCE techniques.

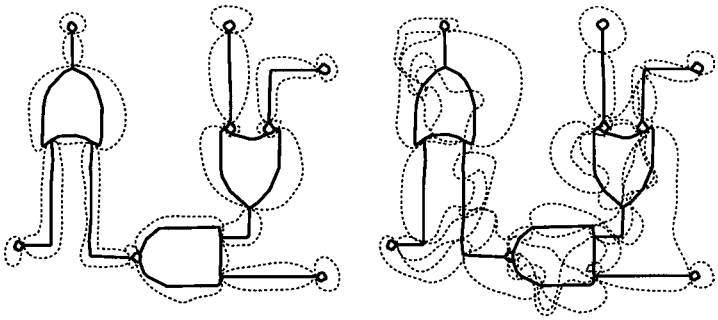
We first comment on the composite attribute vector form introduced in Section 2.3. The form is composed of three types of attributes: attributes of objects, object pairs, and



(a) The Best Result ($RIL = 0.090$)



(b) The Median Result ($RIL = 0.415$)



(c) The Worst Result ($RIL = 0.858$)

Figure 13. Examples of true and estimated partitions from the vector-data image example set.

partitions. We employed the attributes of object pairs so as to be capable of handling the interdependence of objects. For example, in the case of the vector-data segmentation task in Section 5.1.2, assume that there are two line-segments expected to depict connecting lines, and grouped together. If no other line-segments change, but an AND-gate symbol is placed between the above two line-segments, those two line-segments should belong to distinct

clusters. In this case, the existence of other objects influence whether these two objects will belong to the same cluster. To handle such interdependencies among objects, attributes of object pairs are required. It might be considered that the role attributes of objects can be merged into that of attributes of object pairs. We, however, disagree with this opinion, because there exists a category of clustering task for which the assignment of objects can be decided without referring to other objects. In McCallum, Nigam, and Ungar (2000), for example, a set of bibliographic citation strings was partitioned such that each cluster consisted of the strings referring to the same technical papers. Once it appeared that two strings were in the same cluster, those two should continue to remain so, independent of whether any new string is added to the set. In this case, to perform this type of clustering task, it is required to take into account not the interdependencies among objects but the features of objects only. We adopt both attributes of objects and of object pairs, in order to create a method equally applicable to clustering tasks with and without interdependencies among objects. Finally, the role of the attributes of partitions cannot be filled by the other two types of attributes, since the latter cannot fully represent the features of an entire set. Furthermore, the experimental results reveal the effectiveness of adopting attributes of partitions.

Concerning the LCE tasks, the question might be posed whether the task is equivalent to two class discrimination tasks to estimate whether two objects are in the same cluster. From that perspective, it would be difficult to take into account the effects of entire partitions and of interdependencies among the objects.

We next comment on related works. Though we do not know of any previous work that addresses LCE, several works (Bensaid et al., 1996; Emde, 1994) do partitioning using supervisory information. These algorithms partition a given object set that is composed of both unlabeled and labeled objects. However, since a set of class labels is given in advance in these works, they should be categorized into classification algorithms, and are clearly different from ours.

We can summarize the merits of using LCE techniques as follows: The LCE technique makes the rule acquisition process for partitioning more systematic. In order to derive the desired partitions by using a clustering technique, potential functions or dissimilarity measures must be designed to assist in deriving such partitions. The design of such functions or measures involves trial and error, and this work consequently can become troublesome. The LCE algorithms lessen this burden.

The systematic acquisition of partitioning rules is also helpful for attribute selection. In our dot patterns and vector-data images examples, many kinds of partitioning attributes had already been developed. Indeed, we did not develop new attributes specialized for our LCE method in our two domains. We gathered candidate attributes that have previously been used for partitioning, modified them so as to fit a composite attribute vector form, and selected appropriate attributes from these candidates using cross validation. It is an open question as to how difficult attribute discovery would be in novel, less studied domains.

It might be considered that if state of art attributes are available, the partitioning task will become almost trivial. We do not think so, based on the fact that many attributes for the image segmentation task have been developed, but we suppose that such state of art attribute is not yet found. Pavlidis remarked, “It seems that no matter how sophisticated a surface fitting or step finding algorithm we use, we cannot improve the results of object

outlining for a very large class of images" (Pavlidis, 1992). We also should explore another line, such as developing LCE techniques.

On the other hand, LCE techniques have the following drawbacks. The most serious obstacle is that of collecting training examples. Indeed, a great deal of time and effort was spent supplying true partition for one hundred of images in Section 5.1.2. However, we believe that it is more troublesome to design partitioning rule that is competitive to the rules acquired by LCE techniques. More importantly, while only an expert can design partitioning rules, even a non-expert can supply examples. Scalability is another issue. In order to learn the function $f_1(p, A_C(p))$, the size of the training example set must grow according to the order of the squared numbers of objects, which are very numerous. This issue can be avoided by proper sampling techniques. Note that though bitmap images are more popular than vector-data images, we could not handle bitmap images due to this scaling problem. However the focus of this paper was to determine whether our LCE algorithm could learn useful rules, so we targeted a rather small-scale task. Although binary attributes were tested in our experiments, and multi-value attributes were not, the method in Section 3.1 can be applied to multi-value attributes, so we expect that this issue will not be problematic.

7. Conclusions

We introduced learning from cluster examples as a new learning task, and discussed the merits of accomplishing this task. We proposed a solution method for the task and applied this method to object sets in two types of domains. Using a set of dot patterns, we showed that our method could automatically acquire and fully represent the information required for appropriate partitioning. We then showed that these results carried over to the more realistic domain of vector-data images.

Having laid the foundations for the basic task in this paper, we now plan to continue this research by investigating additional approaches. Another topic for further research is the extension of the composite attribute vector form, and the development of a method capable of dealing with attributes assigned to clusters.

Appendix A: The description length for the decision lists and example sets

The total description length of the decision lists and example sets is as follows.

$$\ell(ex_1, DL) = \log^*(m) + \sum_{i=1}^{m} (\ell(T_i) + \ell(S(T_i))).$$

$\log^*(\cdot)$	Rissanen's code length for natural numbers (Rissanen, 1983)
m	the number of terms in the decision list
$\ell(T_i)$	code length for the term T_i
$\ell(S(T_i))$	code length for the example set covered by T_i

Code length for the example set $\boldsymbol{S}$ (Wallace & Patrick, 1993):

$$\ell(S) = \log(\#S + 1) + \log \binom{\#S}{\#S^+}$$

$\#S$ the number of examples in S
$\#S^+$ the number of S elements whose class is 1

Code length for the term $\boldsymbol{T}$:

$$\ell(T) = \sum_{j=1}^{\#A_{\text{used}}} \log\left(\binom{\#A}{\#j} + 1\right) + \sum_{j \in A_{\text{used}}} \ell(L_j)$$

$\#A$ the number of attribute vector elements
A_{used} the set of indices specifying literals used in the term T
$\#A_{\text{used}}$ the number of literals used in the term T
$\ell(L_j)$ code length for the literal L_j

Code length for the literal $\boldsymbol{L}$ *(for continuous attributes)*
The code lengths for the three types of literals are as follows:

$$\begin{aligned}
\ell(a^s < \theta_u) &= \log 3 + \ell(\theta_u), \\
\ell(\theta_l \leq a^s < \theta_u) &= \log 3 + \ell(\theta_l) + \ell(\theta_u), \\
\ell(\theta_l \leq a^s) &= \log 3 + \ell(\theta_l),
\end{aligned}$$

where $\ell(\theta_l)$ and $\ell(\theta_u)$ are code lengths for the thresholds. These lengths are determined as depicted in figure 14. For example, the half-way point between the minimum and maximum thresholds is encoded with one bit. The quarter-way point is encoded with three bits. Every time the precision doubles, two more bits are required to encode the threshold. Itoh's (1992) paper gives a full explanation of this.

Code length for the literal $\boldsymbol{L}$ *(for discrete attributes)*

$$\log(d - 1) + \log \binom{d - 1}{d'}$$

d size of the attribute's domain
d' the number of values appears to the right of the literal

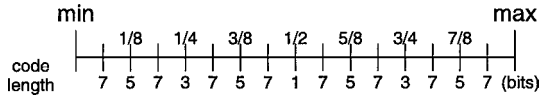


Figure 14. Code length for thresholds.

Appendix B: Attributes for dot patterns

We here describe the details of the candidate attributes shown in Table 1 of Section 5.1.1.

All but the third of the dot attributes should be fairly self-explanatory; they are the X and Y coordinates and the Euclidean distances to the k-th nearest neighbor (the value of parameter k will be introduced below) and to the nearest dot. In the attributes of the object pairs, the first attribute is simply the Euclidean distance between two dots, but the second and third attributes require the imposition of a total order on the dots. Let us call the dots in the object pair A and B. First, the dots are ordered according to their Euclidean distance from dot A, and the position of dot B in this order is found. Then, the dots are ordered according to their distance from dot B, and the position of dot A in this order is found. The second attribute of object pairs is then the smaller of these numerical positions, and the third is the larger.

The fourth attribute of the object pairs is related to the work of Wong and Lane (1983). They employed the following factor for clustering: $(\#O/2k)(V_k(A)+V_k(B))$, where k is an adjustable parameter. The function $V_k(A)$ gives the volume of a region centered on a dot A with radius r_k, the distance to the k-th nearest dot. In 2–dimensional space, this is simply $V_k(A) = (\pi/2){r_k}^2$. We employ Wong and Lane's factor as the fourth attribute of the dot pairs, with parameter k (also mentioned above in relation to the third attribute of the dots) set to the value of $2\log_e \#O$, as suggested in Wong amnd Lane (1983).

The fifth and sixth attributes of object pairs are related to Urquhart's work (Urquhart, 1982) on graph theoretical clustering. Urquhart proposes a clustering technique based on a *Gabriel graph* (GG) and a *relative neighborhood graph* (RNG). The GG is a graph having edges between two dots, A and B, if no other dot lies in the circular region that can be constructed between them, as shown in figure 15(a). The RNG is similar to this, except that the area considered between the two dots is the region described in figure 15(b). We adopt the number of dots in these two types of regions as the fifth and the sixth attributes of object pairs.

The seventh and eighth attributes are related to Zahn's pioneering work on graph theoretical clustering (Zahn, 1971) that employs a *minimal spanning tree* (MST). The MST is defined as a tree connecting all the dots in a given dot pattern for which the sum of the lengths of its constituent edges is minimal among all possible trees. There is only one path between any pair of two dots on an MST. For the dots A and B, we adopt the length of the longest edge on the MST path between them as the seventh attribute and the number of edges on this path as the eighth attribute.

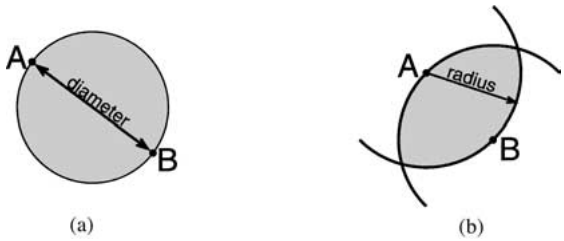


Figure 15. Dot-free regions in Gabriel graphs and relative neighborhood graphs. (a) Region for a Gabriel graph (b) Region for a relative neighbourhood graph.

Appendix C: Attributes for vector-data images

We here describes the details of the candidate attributes shown in Table 3 of Section 5.1.2.

The first four attributes of the line-segments are simply the X coordinate of the line-segment's mid-point, the Y coordinate of the mid-point, the difference between the X coordinates of the two end-points, and the difference between the Y coordinates of the two end-points.

All the other attributes of the line-segments are related to the notion of an *arc*: a series of connected line-segments that do not pass branching or terminal points. Branching points are defined as end-points to which three or more line-segments are connected. Terminal points are defined as end-points to which only one line-segment is connected. Four attributes are calculated from the arc involving the target line-segment. The first of these is the number of line-segments in the arc. The subsequent attributes are the standard deviation of the lengths and the angles of the line-segments in the arc. Finally, the sum of the lengths of the line-segments in the arc is also included as an attribute.

The attributes of object pairs for vector-data images also require some explanation. Let us call the two line-segments in the pair A and B. The first attribute, connection information, is then defined as

$$\begin{cases} x - 1 & \text{if } A \text{ and } B \text{ are directly connected,} \\ 0 & \text{otherwise,} \end{cases}$$

where x is the total number of line-segments connecting to the end-point that A and B have in common. The second attribute is the difference between the angles of the line-segments, regularized so as to range from 0° to 90°. The third attribute is the shortest distance that can join an end-point of A to an end-point of B. The fourth and fifth attributes are found by imposing a total order on all the line-segments. They are similar to the second and third attributes of object pairs for dot patterns, except that the minimum distance between end-points (as in the third property of pairs of line-segments) is used to construct the order. The sixth attribute is then the distance between the mid-points of A and B, and the seventh attribute is a Boolean "yes" or "no" to indicate whether the line-segments A and B belong to the same arc.

Acknowledgments

We thank Ian Frank, Shotaro Akaho, and Kazuo Miyashita for valuable advice.

References

Bensaid, A. M., Hall, L. O., Bezdek, J. C., & Clarke, L. P. (1996). Partially supervised clustering for image segmentation. *Pattern Recognition, 29:5*, 859–871.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth Inc.

Burset, M., & Guigó, R. (1996). Evaluation of gene structure prediction programs. *Genomics, 34*, 353–367.

Cheeseman, P., & Stutz, J. (1996). Bayesian classification (AutoClass): Theory and results. In U. M. Fayyad, G. Diatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 153–180). AAAI Press/The MIT Press, Chapt. 6.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society (B), 39:1*, 1–38.

Emde, W. (1994). Inductive learning of characteristic concept descriptions from small sets of classified examples. In *Proc. of European Conference of Macine Learning,* (pp. 103–121). [LNAI 784].

Everitt, B. S. (1993). *Cluster Analysis*. Edward Arnold, 3rd ed.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139–172.

Itoh, S. (1992). Application of MDL principle to pattern classification problems. *J. of Japanese Society for Artificial Intelligence, 7:4*, 608–614 (in Japanese).

Jain, A. K., & Dubes, R. C. (1988). *Algorithms for Clustering Data*. Prentice Hall.

Kamishima, T., Minoh, M., & Ikeda, K. (1995). Rule formulation based on inductive learning for extraction and classification of diagram symbols. *Transactions of The Information Processing Society of Japan, 36:3*, 614–626 (in Japanese).

McCallum, A., Nigam, K., & Ungar, L. H. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of ACM SIGKDD* (pp. 169–178).

Michalski, R. S. (1993). Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning, 11*, 111–151.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning, 5*, 71–99.

Pavlidis, T. (1992). Why progress in machine vision is so slow. *Pattern Recognition Letters, 13*, 221–225.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1*, 81–106.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation, 80*, 227–248.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *J. of the American Statistical Association, 66*, 846–850.

Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *The Annals of Statistics, 11:2*, 416–431.

Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Urquhart, R. (1982). Graph theoretical clustering based on limited neghbourhood sets. *Pattern Recognition, 15:3*, 173–187.

Wallace, C. S., & Patrick, J. D. (1993). Coding decision trees. *Machine Learning, 11*, 7–22.

Wong, M. A., & Lane, T. (1983). A *k*th nearest neighbour clustering procedure. *Journal of the Royal Statistical Society (B), 45:3*, 362–368.

Yamanishi, K. (1992). A learning criterion for stochastic rules. *Machine Learning, 9*, 165–203.

Yamanishi, K., & Han, T. (1992). Introduction to MDL from viewpoints of information theory. *J. of Japanese Society for Artificial Intelligence, 7:3*, 427–434 (in Japanese).

Zahn, C. T. (1971). Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. on Computers, 20:1*, 68–86.

Received July 11, 2000
Revised April 17, 2002
Accepted April 20, 2002
Final manuscript April 23, 2002