8

Discretization Methods

In this Chapter, we introduce unsupervised and supervised methods for discretization of continuous data attributes. Discretization is one of the most important, and often required, preprocessing methods. Its overall goal is to reduce the complexity of the data for further data mining tasks.

1. Why Discretize Data Attributes?

Often data are described by attributes that assume **continuous** values. If the number of continuous values of the attributes/features is huge, model building for such data can be difficult and/or highly inefficient. Moreover, many data mining algorithms, as described in Part 4 of this book, operate only in discrete search/attribute spaces. Examples of the latter are decision trees and rule algorithms, for which discretization is a necessary preprocessing step, not just a tool for reducing the data and the subsequently generated model complexity.

The goal of **discretization** is to reduce the number of values a continuous attribute assumes by grouping them into a number, n, of intervals (bins).

Two key problems associated with discretization are how to choose the number of intervals (bins), and how to decide on their width. Discretization can be performed with or without taking class information (if available) into account. Analogously to unsupervised vs. supervised learning methods (which require class information), discretization algorithms are divided into two main categories: **unsupervised** and **supervised**. If class information exists (i.e., if we have access to training data) a discretization algorithm should take advantage of it, especially if the subsequently used learning algorithm for model building is supervised. In this case, a discretization algorithm should maximize the interdependence between the attribute values and the class labels. An additional benefit of using class information in the discretization process is that it minimizes the (original) information loss. Figure 8.1 illustrates a trivial case for one attribute, using the same width for all intervals. The top of Figure 8.1 shows a case where a user decided to choose $n = 2$ intervals, without using information about class membership of the data points. The bottom of Figure 8.1 shows the grouping of attribute values into $n = 4$ intervals, while taking into account the class information; the four intervals better discretize the data for a subsequent classification into two classes, represented by white and black ovals. In the first case, by choosing just two intervals, the user made it more difficult for a classifier to distinguish between the classes (since the instances from different classes were grouped into the same intervals). With real data, it would be unusual to have such nicely distributed attribute values; most often there would be a mixture of data points from several classes in each interval, as illustrated in Figure 8.2.

Discretization of continuous attributes is most often performed one attribute at a time, independent of other attributes. This approach is known as **static attribute discretization**. On the other end of the spectrum is **dynamic attribute discretization**, where all attributes are discretized

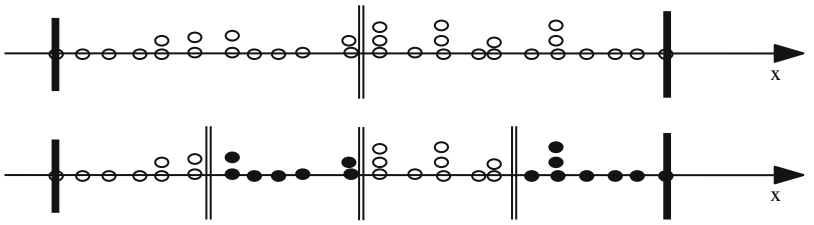

Figure 8.1. Deciding on the number of intervals without using (top) and after using (bottom) class information.

simultaneously while taking into account the interdependencies among them. Still another way to look at discretization algorithms is to consider whether the partitions produced by them apply only to localized regions of the example/instance space. In this case, they are called **local** (like the one performed by the C4.5 algorithm, where not all features are discretized). On the other hand, when all attributes are discretized, they produce $n_1 \bullet n_2 \bullet \ldots n_d$ regions, where n_i is the number of intervals of the i^{th} attribute; such methods are called **global**.

Discretization transforms continuous attribute/feature values into a finite number of intervals and associates with each interval a **discrete** value. Discrete values can be **nominal** (e.g. white, red) or **numerical** (e.g., 1, 2, 3), although for coding purposes nominal values are always converted into numerical ones. Any discretization process consists of two basic steps:

– first, the number of discrete intervals needs to be chosen. This is usually done by the user, although a few discretization algorithms are able to do it on their own. There also exist some heuristic rules that help the user to decide on the number of intervals.
– second, the width (boundary) of each interval (given the range of values of an attribute) must be determined, which is often done by a discretization algorithm itself.

Since all discretization algorithms but one supervised algorithm (outlined later in the Chapter) are *static* and all but one (*k*-means) are *global* we shall use only the following categorization to describe them:

– **unsupervised** (or class-blind), which discretize attributes without taking into account class information
– **supervised** (or class-aware), which discretize attributes while using interdependence between the known class labels and the attribute values

As we will learn in Part 4 of this book, most of the model-building algorithms are strongly influenced by the number of intervals into which the attributes are divided. One such effect is efficiency of model-building, and another the ability of the model to generalize on new data. The larger the number of intervals, the larger the chance for a model to overfit the data (see Chapter15).

To summarize: on the one hand, a discretization algorithm, used as a preprocessing step, should generate as few discrete intervals as possible. On the other hand, however, too few intervals may hide information about the relationship between the class variable (if known) and the interval variable. The latter situation is especially true when the attribute values are not distributed

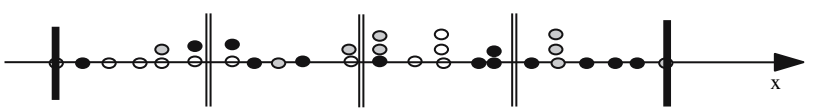

Figure 8.2. Distribution of points belonging to three categories over attribute X.

evenly, in which case a large amount of important original information may be lost. In practice, discretization algorithms search for a number of discrete intervals as a tradeoff between these two goals.

2. Unsupervised Discretization Algorithms

Unsupervised discretization algorithms are the simplest to use and implement. They only require the user to specify the number of intervals and/or how many data points should be included in any given interval.

The following heuristic is often used to choose intervals: *the number of intervals for each attribute should not be smaller than the number of classes (if known)*. The other popular heuristic is to choose the number of intervals, n_{Fi}, for each attribute, $F_{i,}$ $(i = 1, \ldots, n$, where n is the number of attributes), as follows:

$$n_{Fi} = M/(3^*C)$$

where M is the number of training examples and C is the number of known categories.

A description of two unsupervised algorithms follows. We assume that, in general, the user supplies a set of numbers, representing the number of intervals into which each attribute/feature is to be discretized:

$$N = \{n_{F1}, \ldots, n_{Fi}, \ldots, n_{Fn})$$

2.1. Equal-Width Discretization

This algorithm first finds the minimum and maximum values for each feature, F_i, and then divides this range into a number, n_{Fi}, of user-specified, equal-width intervals.

2.2. Equal-Frequency Discretization

This algorithm determines the minimum and maximum values of the attribute, sorts all values in ascending order, and divides the range into a user-defined number of intervals, in such a way that every interval contains the same number of sorted values.

3. Supervised Discretization Algorithms

As we will learn in Part 4 of this book (particularly in Chapter 12), the goal of machine learning algorithms is to generate models of the data that well approximate concepts/hypotheses represented (described) by the data. In other words, their goal is to discover relation between the **class variable** (such as the conclusion (THEN part) of a production rule) and the attribute/feature variable (such as the condition (IF part) of a rule). In a similar manner a supervised discretization problem can be formalized in view of the class-attribute interdependence, as described below.

3.1. Information-Theoretic Algorithms

Many supervised discretization algorithms have their origins in **information theory**. A supervised learning task requires having a training dataset consisting of M examples, where each example belongs to only one of the S classes. Let F indicate a continuous attribute/feature. We say that there exists a discretization scheme D on F that discretizes the continuous attribute F into n discrete intervals, bounded by the pairs of numbers:

$$\mathrm{D} : \{[\mathrm{d}_0, \mathrm{d}_1], (\mathrm{d}_1, \mathrm{d}_2], \ldots, (\mathrm{d}_{\mathrm{n}-1}, \mathrm{d}_{\mathrm{n}}]\} \quad (1)$$

where d_0 is the minimal value and d_n is the maximal value of attribute F, and the values in Equation (1) are arranged in ascending order.

These values constitute the boundary set for discretization D:

$$\{d_0, d_1, d_2, \ldots, d_{n-1}, d_n\}$$

Each value of attribute F can be assigned into only one of the n intervals defined in Equation (1). The membership of each value, within a certain interval, for attribute F may change with a change of the discretization D. The class variable and the discretization variable of attribute F are treated as random variables defining a two-dimensional frequency matrix, called the **quanta matrix**, as shown in Table 8.1.

In the table, q_{ir} is the total number of continuous values belonging to the i^{th} class that are within interval $(d_{r-1}, d_r]$. M_{i+} is the total number of objects belonging to the i^{th} class, and M_{+r} is the total number of continuous values of attribute F that are within the interval $(d_{r-1}, d_r]$, for $i = 1, 2 \ldots, S$ and $r = 1, 2, \ldots, n$.

Example: Let us assume there are three classes, four intervals, and 33 examples, distributed as illustrated in Figure 8.2. For these data we construct the corresponding quanta matrix, shown in Table 8.2.

The values shown in Table 8.2 have been calculated as follows. Total number of values:

$$M = \sum_{r=1}^{L_j} q_{+r} = \sum_{i=1}^{c} q_{i+}$$

$$M = 8 + 7 + 10 + 8 = 33$$

$$M = 11 + 9 + 13 = 33$$

Number of values in the First interval:

$$M_{+r} = \sum_{i=1}^{c} q_{ir}$$

$$M_{+\text{first}} = 5 + 1 + 2 = 8$$

Table 8.1. Two-dimensional quanta matrix for attribute F and discretization scheme D.

Class	**Interval**					**Class Total**
	$[d_0, d_1]$	...	$(d_{r-1}, d_r]$	...	$(d_{n-1}, d_n]$	
C_1	q_{11}	...	q_{1r}	...	q_{1n}	M_{1+}
:	:	...	:	...	:	:
C_i	q_{i1}	...	q_{ir}	...	q_{in}	M_{i+}
:	:	...	:	...	:	:
C_S	q_{S1}	...	q_{Sr}	...	q_{Sn}	M_{S+}
Interval Total	M_{+1}	...	M_{+r}	...	M_{+n}	M

Table 8.2. Quanta matrix corresponding to Figure 8.2.

Classes	**Intervals**				**Total**
	First	**Second**	**Third**	**Fourth**	
White	**5**	**2**	**4**	**0**	11
Grey	**1**	**2**	**2**	**4**	9
Black	**2**	**3**	**4**	**4**	13
Total	8	7	10	8	33

Number of values in the White class:

$$M_{i+} = \sum_{r=1}^{n} q_{ir}$$

$$M_{\text{white}+} = 5+2+4+0 = 11$$

We now calculate several statistics from the above quanta matrix. The estimated joint probability of the occurrence that attribute F values are within interval $D_r = (d_{r-1}, d_r]$ and belong to class C_i is calculated as follows:

$$p_{ir} = p(C_i, D_r|F) = \frac{q_{ir}}{M} \tag{2}$$

(For the example: $p_{\text{white,first}} = 5/33 = 0.24$)

The estimated class marginal probability that attribute F values belong to class C_i, p_{i+}, and the estimated interval marginal probability that attribute F values are within the interval $D_r = (d_{r-1}, d_r]$ p_{+r}, are:

$$p_{i+} = p(C_i) = \frac{M_{i+}}{M} \tag{3}$$

(For the example: $p_{\text{white}+} = 11/33$)

$$p_{+r} = p(D_r|F) = \frac{M_{+r}}{M} \tag{4}$$

(For the example: $p_{+\text{first}} = 8/33$)

The **Class-Attribute Mutual Information** between the class variable C and the discretization variable D for attribute F given the 2-D frequency matrix is defined as

$$I(C, D|F) = \sum_{i=1}^{S}\sum_{r=1}^{n} p_{ir} \log_2 \frac{p_{ir}}{p_{i+}p_{+r}} \tag{5}$$

(For the example: $I(C, D : vj) = 5/33^* \log((5/33)/(11/33^*8/33)) + \ldots + 4/33^* \log((4/33)/(13/33)^*8/33))$

Similarly, the **Class-Attribute Information** and **Shannon's entropy** are defined, respectively, as

$$INFO(C, D|F) = \sum_{i=1}^{S}\sum_{r=1}^{n} p_{ir} \log_2 \frac{p_{+r}}{p_{ir}} \tag{6}$$

(For the example: $\text{INFO(C, D : vj)} = 5/33^* \log((8/33)/(5/33)) + \ldots + 4/33^* \log((8/33)/(4/33))$

$$H(C, D|F) = \sum_{i=1}^{S}\sum_{r=1}^{n} p_{ir} \log_2 \frac{1}{p_{ir}} \tag{7}$$

(For the example: $\text{H(c : vj)} = 5/33^* \log(1/(5/33)) + \ldots + 4/33^* \log(1/(4/33))$

Given Equations (5), (6) and (7), the **Class-Attribute Interdependence Redundancy** criterion (CAIR, or R) and the **Class-Attribute Interdependence Uncertainty** criterion (CAIU, or U) are defined as

$$R(C, D|F) = \frac{I(C, D|F)}{H(C, D|F)} \tag{8}$$

$$U(C, D|F) = \frac{INFO(C, D|F)}{H(C, D|F)} \tag{9}$$

The CAIR criterion measures the interdependence between classes (the larger its value, the better correlated are the class label and the discrete intervals) and the discretized attribute. CAIR is independent of the number of class labels and of the number of unique values of the continuous attribute. The same holds true for the CAIU criterion, but with a reverse relationship. The CAIR criterion is used in a CADD algorithm, described later, that uses a user-specified number of intervals, initializes the discretization intervals using a maximum entropy discretization method, and uses the significance test for selection of a proper confidence interval, all of which add to its complexity. The CAIM algorithm, on the otherhand, described in detail below, avoids these disadvantages.

3.1.1. CAIM: Class-Attribute Interdependency Maximization Algorithm

The **Class-Attribute Interdependency Maximization algorithm** works in a top-down manner: it divides one of the existing intervals into two new intervals, using a criterion function that results in achieving the optimal class-attribute interdependency after the split. It starts with the entire interval $[d_o, d_n]$ and maximizes interdependence between the continuous attribute and its class labels in order to automatically generate a small number of discrete intervals. The discretization criterion used in the CAIM algorithm is described next.

The CAIM criterion (note that the criterion's name is the same as the algorithm's name) measures dependency between the class variable C and the discretization variable D for attribute F, for a given quanta matrix, and is defined as follows:

$$CAIM(C, D|F) = \frac{\sum_{r=1}^{n} \frac{\max_r^2}{M_{+r}}}{n} \tag{10}$$

where n is the number of intervals, r iterates through all intervals, i.e., $r = 1, 2, \ldots, n$, max_r is the maximum value among all q_{ir} values (maximum in the r^{th} column of the quanta matrix), $i = 1, 2, \ldots, S$, and M_{+r} is the total number of continuous values of attribute F that are within the interval $(d_{r-1}, d_r]$.

The CAIM criterion is a heuristic measure defined to quantify the interdependence between classes and the discretized attribute. It has the following properties:

- the larger the value of the CAIM, the higher the interdependence between the class labels and the intervals. The larger the number of values belonging to class C_i within a particular interval (if the number of values belonging to C_i within the interval is the largest, then C_i is called the *leading class*), the higher the interdependence between C_i and the interval. The goal of maximizing the interdependence can be translated into the goal of achieving the largest possible number of values that belong to a leading class, within all intervals. CAIM maximizes the number of values belonging to a leading class by using the max_i operation. CAIM achieves the highest value when all values within a particular interval belong to the same class, for all intervals; then $max_r = M_{ri}$ and CAIM $= M/n$.
- CAIM assumes real values in the interval $[0, M]$, where M is the number of values of attribute F.
- CAIM generates a discretization scheme where each interval potentially has the majority of its values grouped within a single class label
- the squared max_i value is divided by the M_{ri} in order to account for the (negative) impact that values belonging to classes other than the leading class have on the discretization scheme. The more such values, the bigger the value of M_{ri}, which decreases the value of the CAIM criterion.
- scales the max_r^2 value to avoid overflow error during calculations. To do so, we calculate $\frac{\max_r^2}{M_{ri}}$ as $\frac{\max_r}{M_{ri}} \max_r$

– since CAIM favors discretization schemes with smaller numbers of intervals the summed value is divided by the number of intervals n.

The value of the CAIM criterion is calculated with a single pass over the quanta matrix. The optimal discretization scheme can be found by searching over the space of all possible discretization schemes to find the one with the highest value of the CAIM criterion. Since such a search for the globally optimal CAIM value would be highly combinatorial, the algorithm uses a greedy approach to search for an approximation of the optimal value by finding locally maximal values of the criterion. Although this approach does not guarantee finding the global maximum it is computationally inexpensive.

The CAIM algorithm, like all discretization algorithms, consists of two steps: (1) initialization of the candidate interval boundaries and the corresponding initial discretization scheme, and (2) the consecutive additions of a new boundary that results in the locally highest value of the CAIM criterion. The pseudocode of the CAIM algorithm follows.

Given: Data consisting of M examples, S classes, and continuous attributes F_i
For every F_i *DO*:

Step1.
1.1 find the maximum (d_n) and minimum (d_0) values of F_i
1.2 form a set of all distinct values of F_i in ascending order, and initialize all possible interval boundaries B with minimum, maximum and all the midpoints of all the adjacent pairs in the set
1.3 set the initial discretization scheme as $D: \{[d_0, d_n]\}$, set GlobalCAIM=0

Step2.
2.1 initialize $k = 1$
2.2 tentatively add an inner boundary, which is not already in D, from B, and calculate the corresponding CAIM value
2.3 after all the tentative additions have been tried accept the one with the highest value of CAIM
2.4 if (CAIM > GlobalCAIM or $k < S$), then update D with the boundary accepted in step 2.3 and set GlobalCAIM=CAIM; else terminate
2.5 set $k = k + 1$ and go to step 2.2

Result: Discretization scheme D.

The algorithm starts with a single interval that covers all values of an attribute and then divides it iteratively. From all possible division points that are tried (with replacement) in Step 2.2., it chooses the division boundary that gives the highest value of the CAIM criterion. The algorithm uses the heuristic that every discretized attribute should have at least the number of intervals equal to the number of classes.

The running time of the algorithm is log-linear, namely, O (M log (M)). Tables 8.4 and 8.5 illustrate the performance of the algorithm on several datasets. All but the *smo* dataset can be obtained from the UC Irvine ML repository at http://www.ics.uci.edu/~mlearn/MLRepository.html, while the *smo* can be obtained from the StatLib at http://lib.stat.cmu.edu. A summary description of the eight datasets is shown in Table 8.3.

A comparison of the results achieved by the CAIM algorithm and six other unsupervised and supervised discretization algorithms (described later in this Chapter), on the eight datasets, is shown in Table 8.4. The goodness of discretization is evaluated by three measures: (a) the CAIR criterion value, (b) the number of generated intervals, and (c) the execution time.

As mentioned above, one of the goals of data preprocessing is to make it easier for the subsequently used machine learning algorithm to generate a model of the data. In Tables 8.5 and 8.6 we

Table 8.3. Properties of datasets used for comparing discretization algorithms.

Properties	**Datasets**							
	Iris	**sat**	**thy**	**wav**	**ion**	**smo**	**Hea**	**pid**
# of classes	3	6	3	3	2	3	2	2
# of examples	150	6435	7200	3600	351	2855	270	768
# of training / testing examples	10 × cross-validation	10 × cross-validation	10 × cross-validation	10 × cross-validation	10 × cross-validation	10 × cross-validation	10 × cross-validation	10 × cross-validation
# of attributes	4	36	21	21	34	13	13	8
# of continuous attributes	4	36	6	21	32	2	6	8

Table 8.4. Comparison of discretization algorithms using eight datasets (bolded entries indicate the best results).

Criterion	Discretization Method	Dataset															
		iris	**std**	**sat**	**std**	**thy**	**std**	**wav**	**std**	**ion**	**std**	**smo**	**std**	**hea**	**std**	**pid**	**std**
CAIR mean value through all intervals	Equal Width	0.40	0.01	0.24	0	0.071	0	0.068	0	0.098	0	0.011	0	0.087	0	0.058	0
	Equal Frequency	0.41	0.01	0.24	0	0.038	0	0.064	0	0.095	0	0.010	0	0.079	0	0.052	0
	Paterson-Niblett	0.35	0.01	0.21	0	0.144	0.01	**0.141**	0	0.192	0	0.012	0	0.088	0	0.052	0
	Maximum Entropy	0.30	0.01	0.21	0	0.032	0	0.062	0	0.100	0	0.011	0	0.081	0	0.048	0
	CADD	0.51	0.01	**0.26**	0	0.026	0	0.068	0	0.130	0	**0.015**	0	0.098	0.01	0.057	0
	IEM	0.52	0.01	0.22	0	0.141	0.01	0.112	0	**0.193**	0.01	0.000	0	0.118	0.02	0.079	0.01
	CAIM	**0.54**	0.01	**0.26**	0	**0.170**	0.01	0.130	0	0.168	0	0.010	0	**0.138**	0.01	**0.084**	0
# of intervals	Equal Width	16	0	252	0	126	0.48	630	0	640	0	22	0.48	56	0	106	0
	Equal Frequency	16	0	252	0	126	0.48	630	0	640	0	22	0.48	56	0	106	0
	Paterson-Niblett	48	0	432	0	45	0.79	252	0	384	0	17	0.52	48	0.53	62	0.48
	Maximum Entropy	16	0	252	0	125	0.52	630	0	572	6.70	22	0.48	56	0.42	97	0.32
	CADD	16	0.71	246	1.26	84	3.48	628	1.43	536	10.26	22	0.48	55	0.32	96	0.92
	IEM	**12**	0.48	430	4.88	28	1.60	91	1.50	113	17.69	**2**	0	**10**	0.48	17	1.27
	CAIM	**12**	0	**216**	0	**18**	0	**63**	0	**64**	0	6	0	12	0	**16**	0

Table 8.5. Comparison of accuracies achieved by the CLIP4 and C5.0 algorithms (bolded values indicate the best results).

Algorithm	Discretization Method	Datasets															
		iris		sat		thy		wav		ion		smo		hea		pid	
		acc	std	acc	std	acc	std	acc	std	acc	std	acc	std	acc	std	acc	std
CLIP4	Equal Width	88.0	**6.9**	**77.5**	**2.8**	91.7	1.9	68.2	2.2	86.9	6.4	68.6	2.0	64.5	10.1	65.5	6.5
	Equal Frequency	91.2	7.6	76.3	3.4	95.7	2.4	65.4	2.9	81.0	3.7	68.9	2.8	72.6	8.2	63.3	6.3
	Paterson-Niblett	87.3	8.6	75.6	3.9	97.4	0.6	60.9	5.2	**93.7**	3.9	68.9	2.7	68.5	13.6	72.7	5.1
	Maximum Entropy	90.0	6.5	76.4	2.7	97.3	0.9	63.5	2.9	82.9	4.8	68.7	2.8	62.6	9.8	63.4	5.1
	CADD	**93.3**	4.4	**77.5**	2.6	70.1	13.9	61.5	3.4	88.8	3.1	68.8	2.5	72.2	11.4	65.5	4.2
	IEM	92.7	4.9	77.2	2.7	**98.8**	0.5	75.2	1.7	92.4	6.9	66.9	2.6	75.2	8.6	72.2	4.2
	CAIM	92.7	8.0	76.4	2.0	97.9	0.4	**76.0**	1.9	92.7	3.9	**69.8**	4.0	**79.3**	5.0	**72.9**	3.7
C5.0	Equal Width	94.7	5.3	86.0	1.6	95.0	1.1	57.7	8.2	85.5	6.4	69.2	5.4	74.7	5.2	70.8	2.8
	Equal Frequency	94.0	5.8	85.1	1.5	97.6	1.2	57.5	7.9	81.0	12.4	70.1	1.7	69.3	5.7	70.3	5.4
	Paterson-Niblett	94.0	4.9	83.0	1.0	97.8	0.4	74.8	5.6	85.0	8.1	70.1	3.2	**79.9**	7.1	71.7	4.4
	Maximum Entropy	93.3	6.3	85.2	1.5	97.7	0.6	55.5	6.2	86.5	8.8	70.2	3.9	73.3	7.6	66.4	5.9
	CADD	93.3	5.4	86.1	0.9	93.5	0.8	56.9	2.1	77.5	11.9	70.2	4.7	73.6	10.6	71.8	2.2
	IEM	**95.3**	4.5	84.6	1.1	99.4	0.2	**76.6**	2.1	**92.6**	2.9	69.7	1.6	73.4	8.9	**75.8**	4.3
	CAIM	**95.3**	4.5	86.2	1.7	98.9	0.4	72.7	4.2	89.0	5.2	**70.3**	2.9	76.3	8.9	74.6	4.0
	Built-in	92.7	9.4	**86.4**	1.7	**99.8**	0.4	72.6	3.6	87.0	9.5	70.1	1.3	76.8	9.9	73.7	4.9

Table 8.6. Comparison of the number of rules/leaves generated by the CLIP4 and C5.0 algorithms (bolded values indicate the best results).

Algorithm	Discretization Method	Datasets															
		iris		sat		thy		wav		ion		smo		pid		hea	
		#	std	#	std	#	std	#	std	#	std	#	std	#	std	#	std
CLIP4	Equal Width	4.2	0.4	47.9	1.2	**7.0**	0.0	**14.0**	0.0	**1.1**	0.3	20.0	0.0	7.3	0.5	7.0	0.5
	Equal Frequency	4.9	0.6	47.4	0.8	**7.0**	0.0	**14.0**	0.0	1.9	0.3	19.9	0.3	7.2	0.4	6.1	0.7
	Paterson-Niblett	5.2	0.4	42.7	0.8	**7.0**	0.0	**14.0**	0.0	2.0	0.0	19.3	0.7	**1.4**	0.5	7.0	1.1
	Maximum Entropy	6.5	0.7	47.1	0.9	**7.0**	0.0	**14.0**	0.0	2.1	0.3	19.8	0.6	7.0	0.0	**6.0**	0.7
	CADD	4.4	0.7	45.9	1.5	**7.0**	0.0	**14.0**	0.0	2.0	0.0	20.0	0.0	7.1	0.3	6.8	0.6
	IEM	4.0	0.5	44.7	0.9	**7.0**	0.0	**14.0**	0.0	2.1	0.7	18.9	0.6	3.6	0.5	8.3	0.5
	CAIM	**3.6**	0.5	45.6	0.7	**7.0**	0.0	**14.0**	0.0	1.9	0.3	**18.5**	0.5	1.9	0.3	7.6	0.5
C5.0	Equal Width	6.0	0.0	348.5	18.1	31.8	2.5	69.8	20.3	32.7	2.9	**1.0**	0.0	249.7	11.4	66.9	5.6
	Equal Frequency	4.2	0.6	367.0	14.1	56.4	4.8	56.3	10.6	36.5	6.5	**1.0**	0.0	303.4	7.8	82.3	0.6
	Paterson-Niblett	11.8	0.4	243.4	7.8	15.9	2.3	**41.3**	8.1	18.2	2.1	**1.0**	0.0	58.6	3.5	58.0	3.5
	Maximum Entropy	6.0	0.0	390.7	21.9	42.0	0.8	63.1	8.5	32.6	2.4	**1.0**	0.0	306.5	11.6	70.8	8.6
	CADD	4.0	0.0	346.6	12.0	35.7	2.9	72.5	15.7	24.6	5.1	**1.0**	0.0	249.7	15.9	73.2	5.8
	IEM	**3.2**	0.6	466.9	22.0	34.1	3.0	270.1	19.0	12.9	3.0	**1.0**	0.0	**11.5**	2.4	**16.2**	2.0
	CAIM	**3.2**	0.6	332.2	16.1	**10.9**	1.4	58.2	5.6	**7.7**	1.3	**1.0**	0.0	20.0	2.4	31.8	2.9
	Built-in	3.8	0.4	287.7	16.6	11.2	1.3	46.2	4.1	11.1	2.0	1.4	1.3	35.0	9.3	33.3	2.5

show the classification results of two machine learning algorithms, achieved after the data were discretized by different discretization algorithms. The algorithms are a decision tree C5.0 and a rule algorithm CLIP4, which are described in Chapter 12. The classification results are compared using two measures: the accuracy (Table 8.5) and the number of the generated rules (Table 8.6).

Note: Since the decision tree algorithm has a built-in front-end discretization algorithm, and thus is able to generate models from continuous attributes, we show its performance while it generated rules from raw data against the results achieved when discretized data were used.

As we see in Table 8.5 the best accuracy, on average, for the two machine learning algorithms is achieved for data discretized with the CAIM algorithm. Table 8.6 compares the number of rules generated by the two machine learning algorithms and indicates that the CAIM algorithm, by generating discretization schemes with small numbers of intervals, reduces the number of rules generated by the CLIP4 algorithm, and the size of the generated decision trees.

3.1.2. χ^2

A supervised learning χ^2 test (**Chi^2 test**) is used in the CAIR/CADD algorithm, but can also be used on its own for discretization purposes.

In the χ^2 test, we use the decision attribute, so it is a supervised discretization method. Let us note that any interval **Boundary Point** (BP) divides the feature values from the range $[a, b]$, into two parts, namely, the left boundary point $L_{BP} = [a, \text{BP}]$ and the right boundary point $R_{BP} = (\text{BP}, b]$; thus $n = 2$. Using statistics, we can measure the degree of independence between the partition defined by the decision attribute and defined by the interval BP. For that purpose, we use the χ^2 test as follows:

$$\chi^2 = \sum_{r=1}^{2} \sum_{i=1}^{C} \frac{(q_{ir} - E_{ir})^2}{E_{ir}}$$

where E_{ir} is the expected frequency of feature F_{ir} :

$$E_{ir} = \frac{q_{+r} \quad q_{i+}}{M}$$

If either q_{+r} or q_{i+} is zero, then E_{ir} is set to 0.1. It holds that if the partitions defined by a decision attribute and by an interval boundary point BP are independent, then

$$P(q_{i+}) = P(q_{i+}|L_{BP}) = P(q_{i+}|R_{BP})$$

for any class, which means that $q_{ir} = E_{ir}$ for any $r \in [1, 2]$ and $i \in [1, \ldots, C]$, and $\chi^2 = 0$. Conversely, if an interval boundary point properly separates objects from different classes the value of the χ^2 test for this particular BP should be very high. This observation leads to the following heuristic: *retain interval boundaries with corresponding high values of the* χ^2 *test and delete those with small corresponding values*.

Example: Let us calculate χ^2 test values for data shown in the contingency tables shown in Table 8.7.

In Table 8.7 the boundary points BP1 through BP4 separate 14 data points for one feature (the points belong to two classes) into two intervals each (Int.1 and Int.2) in four different ways. The χ^2 values corresponding to the partitions shown in Table 8.7 are (from top left to bottom right) 2.26, 7.02, 0.0, and 14.0, respectively.

In order to use the χ^2 test for discretization, the following procedure can be used. First, the user chooses the χ^2 value, based on the desired significance level and degree of freedom using statistical tables. Then, the χ^2 values are calculated for all adjacent intervals. Next, the smallest of the calculated χ^2 values is found and compared with the user-chosen value: if it is smaller,

Table 8.7. Four hypothetical scenarios for discretizing an attribute.

	BP1				BP2		
Class	**Int.1**	**Int.2**	**Total**	**Class**	**Int.1**	**Int.2**	**Total**
C1	4	4	8	C1	7	1	8
C2	1	5	6	C2	1	5	6
Total	5	9	14	Total	8	6	14

	BP3				BP4		
Class	**Int.1**	**Int.2**	**Total**	**Class**	**Int.1**	**Int.2**	**Total**
C1	4	4	8	C1	8	0	8
C2	3	3	6	C2	0	6	6
Total	7	7	14	Total	8	6	14

then the two corresponding intervals are merged; otherwise, they are kept. The process is repeated until the smallest value found is larger than the user-specified value. The procedure needs to be repeated for each feature separately. There are also several other ways, not covered here, in which the χ^2 test can be used for discretization of features.

3.1.3. Maximum Entropy Discretization

Let us recall that in order to design an effective discretization scheme, one needs to choose:

– *Initial discretization*. One way of doing this is to start with only one interval bounded by the minimum and the maximum values of the attribute. The optimal interval scheme is found by successively adding the candidate boundary points. On the other hand, a search can begin with all the boundary points as candidates for the optimal interval scheme, and then their number can be reduced by elimination. The easiest way to chooe the candidate boundary points, however, is to take all the midpoints between any two nearby values of a continuous feature; alternatively, the user may use some heuristic to specify their number.
– *Criteria for a discretization scheme*. As mentioned above, CAIM, CAIR, and CAIU criteria are good measures of the interdependence between the class variable and the interval variable, and thus all can be used as discretization criteria.

Let T be the set of all possible discretization schemes, with their corresponding quanta matrices. The goal of the maximum entropy discretization is to find a $t^* \in T$ such that

$$H(t^*) \geq H(t) \quad \forall t \in T$$

where H is **Shannon's entropy** as defined before. This method is intended to ensure maximum entropy with minimum loss of information. To calculate the maximum entropy (ME) for one row in the quanta matrix (one class), a discretization scheme must be found that makes the quanta values as even as possible. However, for a general multiclass problem, discretization based on maximum entropy for the quanta matrix can be highly combinatorial. To avoid this situation, the problem of maximizing the total entropy of the quanta matrix is approximated by maximizing the marginal entropy. Then boundary improvement (by successive local perturbation) is performed to maximize the total entropy of the quanta matrix. The ME algorithm after Ching et al. is presented below.

Given: A training data set consisting of M examples and C classes.
For each feature, DO:

1. Initial selection of the interval boundaries:

   a) Calculate the heuristic number of intervals $= M/(3^*C)$
   b) Set the initial boundary so that the sums of the rows for each column in the quanta matrix distribute as evenly as possible to maximize the marginal entropy

2. Local improvement of the interval boundaries

   a) Boundary adjustments are made in increments of the ordered, observed unique feature values to both the lower boundary and the upper boundary for each interval
   b) Accept the new boundary if the total entropy is increased by such an adjustment
   c) Repeat the above until no improvement can be achieved

Result: Final interval boundaries for each feature.

The ME discretization seeks the discretization scheme that preserves the information about a given data set. However, since it hides information about the class-feature interdependence, it is not very helpful for subsequently used ML algorithms, which have to find the class-feature interdependence on their own.

3.1.4. Class-Attribute Interdependence Redundancy Discretization (CAIR)

In order to overcome the ME problem of not retaining the interdependence relationship between the class variable and attribute variable, the CAIR measure can be used as a criterion for the discretization algorithm. Since the problem of maximizing CAIR, if we are to find an optimal number of intervals, is highly combinatorial, a heuristic is used to find a local optimization solution, as outlined in the following pseudocode.

Given: A training data set consisting of M examples and C classes.

1. Interval initialization

   a) Sort in an increasing order the continuous-valued attributes
   b) Calculate the number of intervals from the heuristic $(M/(3^*C))$
   c) Perform ME discretization on the sorted values to obtain initial intervals
   d) Form the quanta matrix corresponding to those initial intervals

2. Interval improvement

   a) Based on the existing boundaries, tentatively eliminate each inner boundary in turn, and calculate its corresponding CAIR
   a) After all tentative elimination has been tried on each existing boundary, accept the one with the largest corresponding value of CAIR
   b) Update the boundaries, and repeat this step until there is no increase of CAIR

3. Interval elimination
   At this step the statistically insignificant (redundant) intervals are consolidated. This is done by using the χ^2 test in the following manner:

   a) Perform the following test:

$$R(C:F_j) \geq \frac{\chi^2}{2 \cdot L \cdot H(C:F_j)}$$

   where χ^2 is the χ^2 value at a certain significance level specified by the user, L is the total number of values in two adjacent intervals, and H is the entropy for these intervals

b) If the test is significant at the specified level, then perform the same test for the next pair of intervals
c) If not, one of the intervals from the pair is redundant, and they are consolidated into one by eliminating the separating boundary

Result: Final interval boundaries for each feature.

The CAIR criterion was used in the CADD algorithm to select a minimum number of intervals without significantly reducing CA interdependence information. The problems with CADD are that it uses the heuristic in selecting the initial number of intervals, that the use of ME to initialize the interval boundaries may result in a very poor discretization scheme, although this outcome is remedied by the boundary perturbation scheme at step 2, and that the confidence level for the χ^2 test needs to be specified by the user. All these factors increase the computational complexity of the algorithm.

3.2. Other Supervised Discretization Algorithms

Below we describe two conceptually simple supervised discretization algorithms that are normally used for other purposes. One is a clustering algorithm and the other a decision tree. Later we outline a dynamic supervised discretization algorithm.

3.2.1. K-means clustering for discretization

The ***K*-means algorithm**, the widely used clustering algorithm, is based on the minimization of a performance index defined as the sum of the squared distances of all vectors, in a cluster, to its center. The K-means algorithm is described in detail in Chapter 9. Below, we restate it expressly for the purpose of discretization of an attribute, namely, for finding the number and the boundaries of intervals.

Given: **A t**raining data set consisting of M examples and C classes, and a user-defined number of intervals n_{Fi} for feature F_i

1. Do class c_j for $(j = 1, \ldots, C)$
2. Choose $K = n_{Fi}$ as the initial number of cluster centers. Initially, the first K values of the feature can be selected as the cluster centers.
3. Distribute the values of the feature among the K cluster centers, based on the minimal distance criterion. As a result, feature values will cluster around the updated K cluster centers.
4. Compute K new cluster centers such that for each cluster the sum of the squared distances from all points in the same cluster to the new cluster center is minimized
5. Check whether the updated K cluster centers are the same as the previous ones, if yes go to step 1; otherwise, go to Step 3

Result: The final boundaries for the single feature that consist of the minimum value of the feature, midpoints between any two nearby cluster prototypes for all classes, and the maximum value of the feature.

The outcome of the algorithm, in an ideal case, is illustrated, for one feature, in Figure 8.3. The behavior of the K-means algorithm is strongly influenced by the number of cluster centers (which the user must choose), the choice of initial cluster centers, the order in which the samples are taken, and geometric properties of the data. In practice, before specifying the number of intervals for each feature, the user might draw the frequency plot for each attribute so that the "guessed" number of intervals (clusters) is as correct as possible. In Figure 8.3 we have correctly "guessed" the number of clusters to be 6, and therefore the result is good. Otherwise, we might have gotten a number of intervals that did not correspond to the true number of clusters present

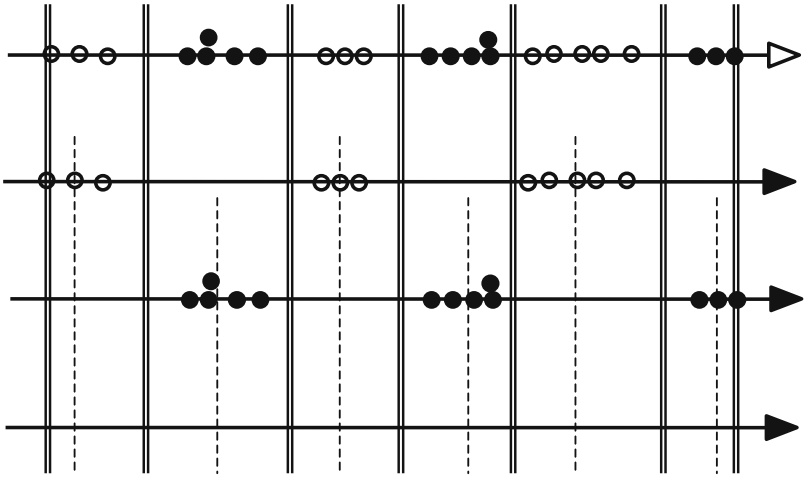

Figure 8.3. Illustration of the K-means discretization algorithm.

in a given attribute (what we are in fact doing is one-dimensional clustering). The problem of choosing the correct number of clusters is inherent to all clustering algorithms. To avoid it, one can cluster the feature values into several different numbers of intervals (clusters) and then calculate some measure of the goodness of clustering (see the discussion of cluster validity measures in Chapter 9). Then, one can choose the number of intervals that corresponds to the optimal value of the used cluster validity measure.

3.2.2. One-level Decision Tree Discretization

In order to better understand the one-level decision tree algorithm the reader is encouraged to read about **decision trees** in Chapter 12. The one-level decision tree algorithm by Holte can be used for feature discretization. It is known as **One-Rule Discretizer**, or **1RD**. It greedily divides the feature range into a number of intervals, using the constraint that each interval must include at least the user-specified minimum number of continuous values (statistics tells us that this number should not be less than 5). The 1RD algorithm starts with initial partition into the intervals, each containing the minimum number of values, and then moves the initial partition boundaries, by adding the feature values, so that each interval contains a strong majority of objects from one decision class. The process is illustrated in Figure 8.4.

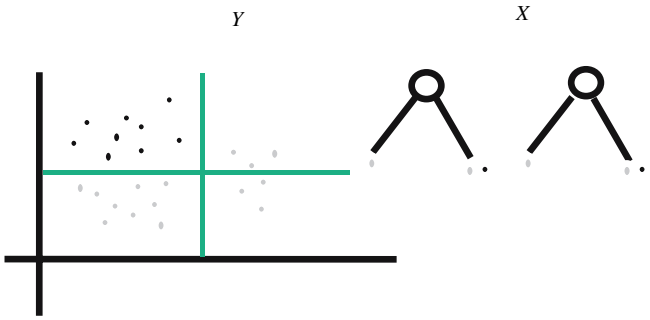

Figure 8.4. Illustration of the 1RD algorithm for 2D data. On the right, we see a discretization using features X and Y, respectively.

3.2.3. Dynamic Attribute Discretization

Few algorithms for dynamic attribute selection exist, and their applicability to large data sets is limited because of the high computational costs. In spite of this situation we outline below one **supervised dynamic discretization** algorithm. Discretization performed by a supervised dynamic discretization algorithm, in an ideal case, may essentially eliminate the need for the subsequent design of a classifier, as illustrated in Figure 8.5, which shows the hypothetical outcome of the algorithm.

From the outcome of such discretization, we may design a classifier (although one that does not recognize all the training instances correctly) in terms of the following rules, using both features:

IF $x_1 = 1$ AND $x_2 = I$	THEN class = MINUS (covers 10 minuses)
IF $x_1 = 2$ AND $x_2 = II$	THEN class = PLUS (covers 10 pluses)
IF $x_1 = 2$ AND $x_2 = III$	THEN class = MINUS (covers 5 minuses)
IF $x_1 = 2$ AND $x_2 = I$	THEN class = MINUS MAJORITY CLASS (covers 3 minuses & 2 pluses)
IF $x_1 = 1$ AND $x_2 = II$	THEN class = PLUS MAJORITY CLASS (covers 2 pluses & 1 minus)

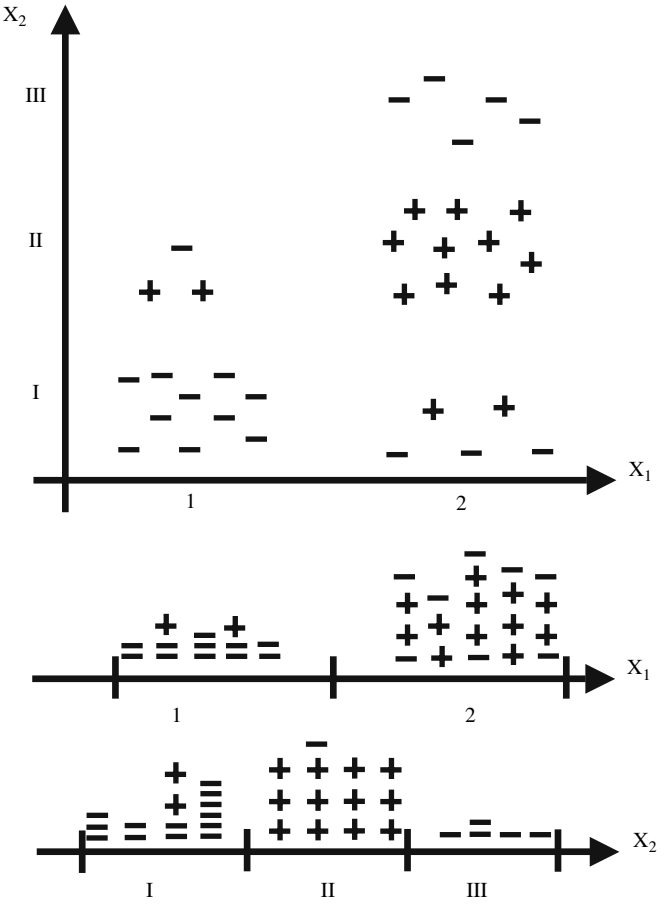

Figure 8.5. Discretization of two attributes performed in a supervised dynamic way. Top: training data instances (their class memberships). Bottom: discretization of the first attribute into two intervals and of the second attribute also into two intervals.

We note that the discretized feature x_2 alone can be used to design a reasonably accurate classifier if the following rules are specified:

IF $x_2 = I$ THEN class = MINUS MAJORITY CLASS
(covers 10 minuses & 2 pluses)
IF $x_2 = II$ THEN class = PLUS MAJORITY CLASS
(covers 10 pluses & 1 minus)
IF $x_2 = III$ THEN class = MINUS (covers 5 minuses)

By defining the above, simpler classifier, we have in fact used the most discriminatory feature (see Chapter 7), identified by the dynamic discretization algorithm. In specifying both classifiers, we have used the terms PLUS/MINUS MAJORITY CLASS to indicate that the corresponding rules do not cover only one class but a mixture of both. As we will see in Chapter 12, to avoid overfitting of the training data, we accept such rules for large data sets.

Below, after Gama et al., we outline one such supervised dynamic discretization algorithm. It uses training data to build hypotheses for possible discretizations of all continuous attributes into a number of intervals for each attribute. Then it uses test data to validate/choose the best hypothesis. The algorithm uses an A* search to determine the number of intervals for each attribute.

The size of the search space for the A* algorithm is $r_1 \bullet r_2 \bullet \ldots r_d$ regions, where r_i is the number of continuous values for the i^{th} attribute. A state is defined as a collection of vectors $(n_1, n_2 \ldots n_d)$, where n_i is the number of intervals for attribute i. The most general discretization would be into one interval for each attribute, i.e., $(1, 1, \ldots 1)$, and the most specific would be to use all original continuous values for all attributes, i.e. $(r_1, r_2 \ldots r_d)$. The goal of the search is to find the optimal value for each attribute, according to some objective function. The objective function for the A* algorithm is defined as $o(n) = g(n) + d(n)$, where n denotes a state. The function g measures the distance to the goal state, and function d measures the depth of the search tree, or distance from the initial state. The function g is defined using the test data set, that is defined in terms of the error committed on the test data using the current discretization scheme for all attributes. For example, for four attributes, it could be (2,3,5,4). The function d is defined as

$$d(n) = \frac{1}{c} \sum_{i=1}^{l} \log(n_i)$$

where c is a user-specified scaling constant used to control the influence of function d on the objective function. The objective function favors smaller over larger number of intervals for each attribute.

Before one can calculate the value of function g (the error on test data), two operations must be performed for a current candidate hypothesis, say, (2,3,5,4):

– both the training and test data must be discretized into 2, 3, 5, and 4 intervals, respectively
– to achieve this outcome one can use any of the discretization algorithms described in this Chapter (we know the number of intervals for each attribute).

Once we have discretized both data sets, we compute the error on the test data. If it is smaller than the current best value, we accept the new discretization. Using our example, if the current state is (2,3,5,4) and we decide to increase the number of intervals for each attribute by one (our initial state was (1,1,1,1)), and the current node has four children nodes ((3,3,5,4), (2,4,5,4), etc.), the decision of which of these nodes to expand will depend on the value of the objective function: only the node with the smallest value will be expanded.

The stopping criteria for the algorithm are as follows: if the value of the objective function is zero (no error on test data), then stop: optimal discretization has been found. The other criterion is that if after a few iterations (say, 3) there is no improvement of the objective function value, then increase the number of intervals by one. If, again, after a few iterations there is no improvement, then stop: optimal discretization has been found.

From the above outline, we note that this supervised dynamic discretization algorithm, although interesting, is computationally expensive. That is why dynamic algorithms are not used in practice.

3.2.4. Paterson and Niblett Algorithm

Finally, we briefly describe an algorithm that is incorporated into C4.5 and C5.0 algorithms as a preprocessing step, used before building a decision tree, to discretize continuous data. For feature F_i that has a number, r, of continuous values, first the training examples are sorted on the values of feature F_i. Then they are split, on threshold R, into two intervals: those for which $F_i \leq R$ and those for which $F_i > R$. To find the threshold R, all the r-1 possible divisions on feature F_i are examined. For each possible value of R, the **information gain** is computed, and the threshold corresponding to the largest value of information gain, or **gain ratio**, is chosen as the threshold to discretize this feature (see Chapter 12 for information about decision trees).

4. Summary and Bibliographical Notes

In this Chapter we introduced two **unsupervised** and several **supervised discretization algorithms** [1, 2, 4, 5, 6, 8, 9, 10, 11]. We also outlined one **dynamic discretization algorithm** [7] and performed comparison between several state-of-the-art algorithms [3, 10]. Discretization algorithms are one of the most important preprocessing tools. Depending on the goal of preprocessing and the type of data, we use either one type of algorithm or another. If the goal is to build a model of the data in a situation where training data are available, one should use a supervised discretization algorithm. If the data are very large then one is constrained to using only the simplest algorithms. The most important topics discussed in this Chapter are the notion of the **quanta matrix** and the **information-theoretic approach** to discretization.

References

1. Cios, K.J., Pedrycz, W., and Swiniarski, R. 1998. *Data Mining Methods for Knowledge Discovery*, Kluwer
2. Ching, J.Y., Wong, A.K.C., and Chan, K.C.C. 1995. Class-dependent discretization for inductive learning from continuous and mixed-mode data. *IEEE Transactions on PAMI*, 17:641–651
3. Cios, K.J., and Kurgan, L. 2004. CLIP4: Hybrid inductive machine learning algorithm that generates inequality rules. *Information Sciences*, 163(1–3): 37–83
4. Doughherty, J., Kohavi, R., and Sahami, M. 1995. Supervised and unsupervised discretization of continuous features. In: Machine Learning: Proceedings of the 12th International Conference, Prieditis, A., and Russell S. (Eds.)
5. Fayyad, U.M., and Irani, K.B. 1992. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102
6. Fayyad, U.M., and Irani, K.B. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. In: *Proceedings of the 13th Inernationa lJoint Conference on Artificial Intelligence*, Morgan-Kaufmann, 1022–1027
7. Gama, J., Torgo, L., and Soares, C. 1998. Dynamic discretization of continuous attributes. In: *Proceedings of the Sixth Ibero-American Conference on Artificial Intelligence*, 160–169
8. Holte, R.C. 1993. Very simple classification rules perform well on most commonly used data sets. *Machine Learning*, 11:63–90
9. Kerber, R. 1992. Chimerge: discretization of numeric attributes. In: *Proceedings of the 10th National Conference on Artificial Intelligence*, MIT Press, 123–128

10. Kurgan, L., and Cios, K.J., 2004. CAIM discretization algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 16(2):145–153
11. Paterson, A., and Niblett, T.B. 1987. *ACLS Manual*. Edinburgh Intelligent Terminals, Ltd.

5. Exercises

1. What are the basic categories of discretization algorithms?
2. What is a quanta matrix?
3. Implement and run a one-level decision tree to discretize the iris and pid data sets (see Table 8.3).
4. Implement and run the CAIM algorithm on the iris and pid data sets (see Table 8.3).
5. Implement and run the dynamic discretization algorithm on the iris and pid data sets (see Table 8.3).