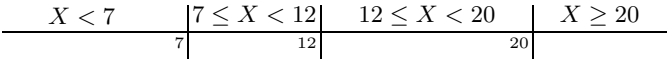
8
Continuous Attributes

8.1 Introduction

Many data mining algorithms, including the TDIDT tree generation algorithm, require all attributes to take categorical values. However, in the real world many attributes are naturally *continuous*, e.g. height, weight, length, temperature and speed. It is essential for a practical data mining system to be able to handle such attributes. In some cases the algorithms can be adapted for use with continuous attributes. In other cases, this is hard or impossible to do.

Although it would be possible to treat a continuous attribute as a categorical one with values 6.3, 7.2, 8.3, 9.2 etc., say, this is very unlikely to prove satisfactory in general. If the continuous attribute has a large number of different values in the training set, it is likely that any particular value will only occur a small number of times, perhaps only once, and rules that include tests for specific values such as $X = 7.2$ will probably be of very little value for prediction.

The standard approach is to split the values of a continuous attribute into a number of non-overlapping ranges. For example a continuous attribute X might be divided into the four ranges $X < 7$, $7 \leq X < 12$, $12 \leq X < 20$ and $X \geq 20$. This allows it to be treated as a categorical attribute with four possible values. In the figure below, the values 7, 12 and 20 are called *cut values* or *cut points*.



M. Bramer, *Principles of Data Mining*, Undergraduate Topics in Computer Science, DOI 10.1007/978-1-4471-7307-6_8

As further examples, an *age* attribute might be converted from a continuous numerical value into six ranges, corresponding to infant, child, young adult, adult, middle-aged and old, or a continuous attribute *height* might be replaced by a categorical one with values such as very short, short, medium, tall, very tall.

Converting a continuous attribute to one with a discrete set of values, i.e. a categorical attribute, is known as *discretisation*.

There are a number of possible approaches to discretising continuous attributes. Ideally the boundary points chosen for the ranges (the cut points) should reflect real properties of the domain being investigated, e.g. constant values in a physical or mathematical law. In practice it is very rarely possible to give principled reasons for choosing one set of ranges over another (for example where should the boundary be between tall and very tall or between medium and tall?) and the choice of ranges will generally have to be made pragmatically.

Suppose that we have a continuous attribute *length*, with values in the range from 0.3 to 6.6 inclusive. One possibility would be to divide these into three ranges of equal size, i.e.

$0.3 \leq length < 2.4$
$2.4 \leq length < 4.5$
$4.5 \leq length \leq 6.6$

This is known as the *equal width intervals* method. However there are obvious problems. Why choose three ranges, not four or two (or twelve)? More fundamentally it may be that some, or perhaps even many, of the values are in a narrow range such as 2.35 to 2.45. In this case any rule involving a test on $length < 2.4$ would include instances where *length* is say 2.39999 and exclude those where *length* is 2.40001. It is highly unlikely that there is any real difference between those values, especially if they were all measured imprecisely by different people at different times. On the other hand, if there were no values between say 2.3 and 2.5, a test such as $length < 2.4$ would probably be far more reasonable.

Another possibility would be to divide *length* into three ranges, this time so that there are the same number of instances in each of the three ranges. This might lead to a split such as

$0.3 \leq length < 2.385$
$2.385 \leq length < 3.0$
$3.0 \leq length \leq 6.6$

This is known as the *equal frequency intervals* method. It would seem to be preferable to the equal width intervals method given above but is still prone

to the same problem at cut points, e.g. why is a length of 2.99999 treated differently from one of 3.00001?

The problem with any method of discretising continuous attributes is that of over-sensitivity. Whichever cut points are chosen there will always be a potential problem with values that fall just below a cut point being treated differently from those that fall just above for no principled reason.

Ideally we would like to find 'gaps' in the range of values. If in the *length* example there were many values from 0.3 to 0.4 with the next smallest value being 2.2, a test such as $length < 1.0$ would avoid problems around the cut point, as there are no instances (in the training set) with values close to 1.0. The value 1.0 is obviously arbitrary and a different cut point, e.g. 1.5 could just as well have been chosen. Unfortunately the same gaps may not occur in unseen test data. If there were values such as 0.99, 1.05, 1.49 and 1.51 in the test data, whether the arbitrary choice of cut point was 1.0 or 1.5 could be of critical importance.

Although both the equal width intervals and the equal frequency intervals methods are reasonably effective, they both suffer from the fundamental weakness, as far as classification problems are concerned, that they take no account of the classifications when determining where to place the cut points, and other methods which do make use of the classifications are generally preferred. Two of these are described in Sections 8.3 and 8.4.

8.2 Local versus Global Discretisation

Some data mining algorithms, such as the TDIDT rule generation algorithm, can be adapted so that each continuous attribute is converted to a categorical attribute at each stage of the process (e.g. at each node of the decision tree). This is known as *local discretisation*.

An alternative approach is to use a *global discretisation* algorithm to convert each continuous attribute to a categorical one once and for all independently of any data mining algorithm that may subsequently be applied to the converted training set. For example, continuous attribute *Age* might be converted to categorical attribute *Age2*, with four values *A*, *B*, *C* and *D*, corresponding to ages in the intervals 0 to under 16, 16 to under 30, 30 to under 60 and 60 and over, respectively, with the three 'split values' 16, 30 and 60 determined *globally* from consideration of the training set as a whole. Although attractive in principle, finding an appropriate global discretisation is not necessarily easy to achieve in practice.

8.3 Adding Local Discretisation to TDIDT

The TDIDT algorithm is a widely used method of generating classification rules via the intermediate representation of a decision tree. (For definiteness in the description that follows we shall assume that the information gain attribute selection criterion is used, but this is not essential.) TDIDT can be extended to deal with continuous attributes in a number of ways. For example, at each node in the decision tree each continuous attribute can be converted to a categorical attribute with several values, by one of the methods described in Section 8.1 or otherwise.

An alternative approach is at each node to convert each continuous attribute to a number of alternative categorical attributes. For example if continuous attribute A has values -12.4, -2.4, 3.5, 6.7 and 8.5 (each possibly occurring several times) a test such as $A < 3.5$ splits the training data into two parts, those instances for which $A < 3.5$ and those for which $A \geq 3.5$. A test such as $A < 3.5$ can be considered as equivalent to a kind of categorical attribute with the two possible values true and false. We will use the phrase *pseudo-attribute* to describe it.

If a continuous attribute A has n distinct values $v_1, v_2, \ldots, v_n$ (in ascending numerical order) there are $n - 1$ possible corresponding pseudo-attributes (all binary), i.e. $A < v_2$, $A < v_3$, $\ldots$, $A < v_n$ (we omit $A < v_1$ as no values of A can be less than v_1, the smallest value).

We can imagine that for the part of the training set under consideration at each node all the continuous attribute columns are replaced by new columns for each pseudo-attribute derived from each continuous attribute. They would then be in competition for selection with each other and with any genuine categorical attributes. This imaginary replacement table will probably have far more columns than before but as all the attributes/pseudo-attributes are categorical it can be processed by the standard TDIDT algorithm to find the one with the largest corresponding information gain (or other measure).

If it turns out that one of the pseudo-attributes, say $Age < 27.3$, is selected at a given node, we can consider the continuous attribute *Age* as having been discretised into two intervals with cut point 27.3.

This is a local discretisation which does not (in the standard form of this method) lead to the continuous attribute itself being discarded. Hence there may be a further test such as $Age < 14.1$ at a lower level in the 'yes' branch descending from the test $Age < 27.3$.

The process described above may seem resource intensive but it is not as bad as it would first appear. We will come back to this point in Section 8.3.2,

but leaving it aside at the moment, we have an algorithm for incorporating local discretisation into TDIDT as follows.

At each node:

1. For each continuous attribute A
   a) Sort the instances into ascending numerical order.
   b) If there are n distinct values v_1, v_2, ..., v_n, calculate the values of information gain (or other measure) for each of the $n-1$ corresponding pseudo-attributes $A < v_2$, $A < v_3$, ..., $A < v_n$.
   c) Find which of the $n-1$ attribute values gives the largest value of information gain (or optimises some other measure). If this is v_i return the pseudo-attribute $A < v_i$, and the value of the corresponding measure.
2. Calculate the value of information gain (or other measure) for any categorical attributes.
3. Select the attribute or pseudo-attribute with the largest value of information gain (or which optimises some other measure).

8.3.1 Calculating the Information Gain of a Set of Pseudo-attributes

At any node of the evolving decision tree the entropy values (and hence the information gain values) of all the pseudo-attributes derived from a given continuous attribute can be calculated with a single pass through the training data. The same applies to any other measure that can be calculated using the frequency table method described in Chapter 6. There are three stages.

Stage 1

Before processing any continuous attributes at a node we first need to count the number of instances with each of the possible classifications in the part of the training set under consideration at the node. (These are the sums of the values in each row of a frequency table such as Figure 6.2.) These values do not depend on which attribute is subsequently processed and so only have to be counted once at each node of the tree.

Stage 2

We next work through the continuous attributes one by one. We will assume that a particular continuous attribute under consideration is named *Var* and that the aim is to find the largest value of a specified measure for all possible pseudo-attributes $Var < X$ where X is one of the values of *Var* in the part of the training set under consideration at the given node. We will call the values of attribute *Var* *candidate cut points*. We will call the largest value of measure *maxmeasure* and the value of X that gives that largest value the *cut point* for attribute *Var*.

Stage 3

Having found the value of *maxmeasure* (and the corresponding cut points) for all the continuous attributes, we next need to find the largest and then compare it with the values of the measure obtained for any categorical attributes to determine which attribute or pseudo-attribute to split on at the node.

To illustrate this process we will use the *golf* training set introduced in Chapter 4. For simplicity we will assume that we are at the root node of the decision tree but the same method can be applied (with a reduced training set of course) at any node of the tree.

We start by counting the number of instances with each of the possible classifications. Here there are 9 *play* and 5 *don't play*, making a total of 14.

We now need to process each of the continuous attributes in turn (Stage 2). There are two: *temperature* and *humidity*. We will illustrate the processing involved at Stage 2 using attribute *temperature*.

The first step is to sort the values of the attribute into ascending numerical order and create a table containing just two columns: one for the sorted attribute values and the other for the corresponding classification. We will call this the *sorted instances table*.

Figure 8.1 shows the result of this for our example. Note that temperature values 72 and 75 both occur twice. There are 12 distinct values 64, 65, ..., 85.

Temperature	Class
64	play
65	don't play
68	play
69	play
70	play
71	don't play
72	play
72	don't play
75	play
75	play
80	don't play
81	play
83	play
85	don't play

Figure 8.1 Sorted Instances Table for *golf* Dataset

The algorithm for processing the sorted instances table for continuous attribute *Var* is given in Figure 8.2. It is assumed that there are n instances and the rows in the sorted instances table are numbered from 1 to n. The attribute value corresponding to row i is denoted by value(i) and the corresponding class is denoted by class(i).

Essentially, we work through the table row by row from top to bottom, accumulating a count of the number of instances with each classification. As each row is processed its attribute value is compared with the value for the row below. If the latter value is larger it is treated as a candidate cut point and the value of the measure is computed using the frequency table method (the example that follows will show how this is done). Otherwise the attribute values must be the same and processing continues to the next row. After the last but one row has been processed, processing stops (the final row has nothing below it with which to compare).

The algorithm returns two values: *maxmeasure* and *cutvalue*, which are respectively the largest value of the measure that can be obtained for a pseudo-attribute derived from attribute *Var* and the corresponding cut value.

Algorithm for Processing a Sorted Instances Table
Set count of all classes to zero

Set maxmeasure to a value less than the smallest
possible value of the measure used

for $i = 1$ to $n - 1$ {
  increase count of class(i) by 1
  if value(i) < value($i + 1$){
    (a) Construct a frequency table for pseudo-attribute
        $Var <$ value($i + 1$)
    (b) Calculate the value of *measure*
    (c) If measure > maxmeasure {
        maxmeasure = measure
        cutvalue = value($i + 1$)
    }
  }
}

Figure 8.2 Algorithm for Processing a Sorted Instances Table

Returning to the *golf* training set and continuous attribute temperature, we start with the first instance, which has temperature 64 and class *play*. We increase the count for class *play* to 1. The count for class *don't play* is zero. The value of temperature is less than that for the next instance so we construct a frequency table for the pseudo-attribute *temperature* < 65 (Figure 8.3(a)).

Class	*temperature* < 65	*temperature* ≥ 65	Class total
play	1 *	8	**9**
don't play	0 *	5	**5**
Column sum	1	13	**14**

Figure 8.3(a) Frequency Table for *golf* Example

In this and the other frequency tables in this section the counts of play and don't play in the 'temperature < xxx' column are marked with an asterisk. The entries in the final column are fixed (the same for all attributes) and are shown in bold. All the other entries are calculated from these by simple addition and subtraction. Once the frequency table has been constructed, the values of

measures such as Information Gain and Gain Ratio can be calculated from it, as described in Chapter 6.

Figure 8.3(b) shows the frequency table resulting after the next row of the sorted instances table has been examined. The counts are now *play* = 1, *don't play* = 1.

Class	*temperature* < 68	*temperature* ≥ 68	Class total
play	1 *	8	**9**
don't play	1 *	4	**5**
Column sum	2	12	**14**

Figure 8.3(b) Frequency Table for *golf* Example

The value of Information Gain (or the other measures) can again be calculated from this table. The important point here is how easily this second frequency table can be derived from the first. Only the *don't play* row has changed by moving just one instance from the 'greater than or equal to' column to the 'less than' column.

We proceed in this way processing rows 3, 4, 5 and 6 and generating a new frequency table (and hence a new value of measure) for each one. When we come to the seventh row (*temperature* = 72) we note that the value of temperature for the next instance is the same as for the current one (both 72), so we do not create a new frequency table but instead go on to row 8. As the value of temperature for this is different from that for the next instance we construct a frequency table for the latter value, i.e. for pseudo-attribute *temperature* < 75 (Figure 8.3(c)).

Class	*temperature* < 75	*temperature* ≥ 75	Class total
play	5 *	4	**9**
don't play	3 *	2	**5**
Column sum	8	6	**14**

Figure 8.3(c) Frequency Table for *golf* Example

We go on in this way until we have processed row 13 (out of 14). This ensures that frequency tables are constructed for all the distinct values of temperature except the first. There are 11 of these candidate cut values, corresponding to pseudo-attributes *temperature* < 65, *temperature* < 68, ..., *temperature* < 85.

The value of this method is that the 11 frequency tables are generated from each other one by one, by a single pass through the sorted instances table.

At each stage it is only necessary to update the relevant count of instances in the appropriate class to move from one frequency table to the next. Having duplicated attribute values is a complication, but it is easily overcome.

8.3.2 Computational Efficiency

This section looks at three efficiency issues associated with the method described in Section 8.3.1.

(a) *Sorting continuous values into ascending numerical order*

This is the principal overhead on the use of the method and thus the principal limitation on the maximum size of training set that can be handled. This is also true of almost any other conceivable method of discretising continuous attributes. For this algorithm it has to be carried out once for each continuous attribute at each node of the decision tree.

It is important to use an efficient method of sorting, especially if the number of instances is large. The one most commonly used is probably Quicksort, descriptions of which are readily available from books (and websites) about sorting. Its most important feature is that the number of operations required is approximately a constant multiple of $n \times \log_2 n$, where n is the number of instances. We say it *varies as* $n \times \log_2 n$. This may not seem important but there are other sorting algorithms that vary as n^2 (or worse) and the difference is considerable.

Figure 8.4 shows the values of $n \times \log_2 n$ and n^2 for different values of n. It is clear from the table that a good choice of sorting algorithm is essential.

n	$n \times \log_2 n$	n^2
100	664	10,000
500	4,483	250,000
1,000	9,966	1,000,000
10,000	132,877	100,000,000
100,000	1,660,964	10,000,000,000
1,000,000	19,931,569	1,000,000,000,000

Figure 8.4 Comparison of Values of $n \log_2 n$ and n^2

The difference between the values in the second and third columns of this table is considerable. Taking the final row for illustration, if we imagine a sorting

task for 1,000,000 items (not a huge number) that takes 19,931,569 steps and assume that each step takes just one microsecond to perform, the time required would be 19.9 seconds. If we used an alternative method to perform the same task that takes 1,000,000,000,000 steps, each lasting a microsecond, the time would increase to over 11.5 days.

(b) *Calculating the measure value for each frequency table*

For any given continuous attribute, generating the frequency tables takes just one pass through the training data. The number of such tables is the same as the number of cut values, i.e. the number of distinct attribute values (ignoring the first). Each table comprises just $2 \times 2 = 4$ entries in its main body plus two column sums. Processing many of these small tables should be reasonably manageable.

(c) *Number of candidate cut points*

As the method was described in Section 8.3.1 the number of candidate cut points is always the same as the number of distinct values of the attribute (ignoring the first). For a large training set the number of distinct values may also be large. One possibility is to reduce the number of candidate cut points by making use of class information.

Figure 8.5 is the sorted instances table for the *golf* training set and attribute *temperature*, previously shown in Section 8.3.1, with the eleven cut values indicated with asterisks (where there are repeated attribute values only the last occurrence is treated as a cut value).

We can reduce this number by applying the rule 'only include attribute values for which the class value is different from that for the previous attribute value'. Thus attribute value 65 is included because the corresponding class value (don't play) is different from the class corresponding to temperature 64, which is play. Attribute value 69 is excluded because the corresponding class (play) is the same as that for attribute value 68. Figure 8.6 shows the result of applying this rule.

The instances with temperature value 65, 68, 71, 81 and 85 are included. Instances with value 69, 70 and 83 are excluded.

However, repeated attribute values lead to complications. Should 72, 75 and 80 be included or excluded? We cannot apply the rule 'only include attribute values for which the class value is different from that for the previous attribute value' to the two instances with attribute value 72 because one of their class values (don't play) is the same as for the previous attribute value and the other (play) is not. Even though both instances with temperature 75 have class play,

Temperature	Class
64	play
65 *	don't play
68 *	play
69 *	play
70 *	play
71 *	don't play
72	play
72 *	don't play
75	play
75 *	play
80 *	don't play
81 *	play
83 *	play
85 *	don't play

Figure 8.5 Sorted Instances with Candidate Cut Values

Temperature	Class
64	play
65 *	don't play
68 *	play
69	play
70	play
71 *	don't play
72	play
72 ?	don't play
75	play
75 ?	play
80 ?	don't play
81 *	play
83	play
85 *	don't play

Figure 8.6 Sorted Instances with Candidate Cut Values (revised)

we still cannot apply the rule. Which of the instances for the previous attribute value, 72, would we use? It seems reasonable to include 80, as the class for both occurrences of 75 is play, but what if they were a combination of play and don't play?

There are other combinations that can occur, but in practice none of this need cause us any problems. It does no harm to examine more candidate cut points than the bare minimum and a simple amended rule is: 'only include attribute values for which the class value is different from that for the previous attribute value, together with any attribute which occurs more than once and the attribute immediately following it'.

This gives the final version of the table shown in Figure 8.7, with eight candidate cut values.

Temperature	Class
64	play
65 *	don't play
68 *	play
69	play
70	play
71 *	don't play
72	play
72 *	don't play
75	play
75 *	play
80 *	don't play
81 *	play
83	play
85 *	don't play

Figure 8.7 Sorted Instances with Candidate Cut Values (final)

8.4 Using the ChiMerge Algorithm for Global Discretisation

ChiMerge is a well-known algorithm for global discretisation introduced by Randy Kerber, an American researcher [1]. It uses a statistical technique for discretising each continuous attribute separately.

The first step in discretising a continuous attribute is to sort its values into ascending numerical order, with the corresponding classifications sorted into the same order.

The next step is to construct a frequency table giving the number of occurrences of each distinct value of the attribute for each possible classification. It then uses the distribution of the values of the attribute within the different classes to generate a set of intervals that are considered statistically distinct at a given level of significance.

As an example, suppose that A is a continuous attribute in a training set with 60 instances and three possible classifications $c1$, $c2$ and $c3$. A possible distribution of the values of A arranged in ascending numerical order is shown in Figure 8.8. The aim is to combine the values of A into a number of ranges. Note that some of the attribute values occur just once, whilst others occur several times.

Value of A	Observed frequency for class			Total
	$c1$	$c2$	$c3$	
1.3	1	0	4	5
1.4	0	1	0	1
1.8	1	1	1	3
2.4	6	0	2	8
6.5	3	2	4	9
8.7	6	0	1	7
12.1	7	2	3	12
29.4	0	0	1	1
56.2	2	4	0	6
87.1	0	1	3	4
89.0	1	1	2	4

Figure 8.8 ChiMerge: Initial Frequency Table

Each row can be interpreted not just as corresponding to a single attribute value but as representing an *interval*, i.e. a range of values starting at the value given and continuing up to but excluding the value given in the row below. Thus the row labelled 1.3 corresponds to the interval $1.3 \leq A < 1.4$. We can regard the values 1.3, 1.4 etc. as *interval labels*, with each label being used to indicate the lowest number in the range of values included in that interval. The final row corresponds to all values of A from 89.0 upwards.

The initial frequency table could be augmented by an additional column showing the interval corresponding to each row (Figure 8.9).

Value of A	Interval	Observed frequency for class			Total
		$c1$	$c2$	$c3$	
1.3	$1.3 \leq A < 1.4$	1	0	4	5
1.4	$1.4 \leq A < 1.8$	0	1	0	1
1.8	$1.8 \leq A < 2.4$	1	1	1	3
2.4	$2.4 \leq A < 6.5$	6	0	2	8
6.5	$6.5 \leq A < 8.7$	3	2	4	9
8.7	$8.7 \leq A < 12.1$	6	0	1	7
12.1	$12.1 \leq A < 29.4$	7	2	3	12
29.4	$29.4 \leq A < 56.2$	0	0	1	1
56.2	$56.2 \leq A < 87.1$	2	4	0	6
87.1	$87.1 \leq A < 89.0$	0	1	3	4
89.0	$89.0 \leq A$	1	1	2	4

Figure 8.9 ChiMerge: Initial Frequency Table with Intervals Added

In practice, the 'Interval' column is generally omitted as it is implied by the entries in the Value column.

Starting with the initial frequency table, ChiMerge systematically applies statistical tests to combine pairs of adjacent intervals until it arrives at a set of intervals that are considered statistically different at a given level of significance.

ChiMerge tests the following hypothesis for each pair of adjacent rows in turn.

Hypothesis
The class is independent of which of the two adjacent intervals an instance belongs to.

If the hypothesis is confirmed, there is no advantage in treating the intervals separately and they are merged. If not, they remain separate.

ChiMerge works through the frequency table from top to bottom, examining each pair of adjacent rows (intervals) in turn in order to determine whether the relative class frequencies of the two intervals are significantly different. If not, the two intervals are considered to be similar enough to justify merging them into a single interval.

The statistical test applied is the χ^2 test, pronounced (and often written) as the 'Chi square' test. χ is a Greek letter, which is written as Chi in the Roman alphabet. It is pronounced like 'sky', without the initial 's'.

For each pair of adjacent rows a *contingency table* is constructed giving the observed frequencies of each combination of the two variables A and 'class'. For

the adjacent intervals labelled 8.7 and 12.1 in Figure 8.8 the contingency table is shown below as Figure 8.10(a).

Value of A	Observed frequency for class			Total observed
	$c1$	$c2$	$c3$	
8.7	6	0	1	7
12.1	7	2	3	12
Total	13	2	4	19

Figure 8.10(a) Observed Frequencies for Two Adjacent Rows of Figure 8.8

The 'row sum' figures in the right-hand column and the 'column sum' figures in the bottom row are called 'marginal totals'. They correspond respectively to the number of instances for each value of A (i.e. with their value of attribute A in the corresponding interval) and the number of instances in each class for both intervals combined. The grand total (19 instances in this case) is given in the bottom right-hand corner of the table.

The contingency table is used to calculate the value of a variable called χ^2 (or 'the χ^2 statistic' or 'the Chi-square statistic'), using a method that will be described in Section 8.4.1. This value is then compared with a *threshold value* T, which depends on the number of classes and the level of statistical significance required. The threshold will be described further in Section 8.4.2. For the current example we will use a significance level of 90% (explained below). As there are three classes this gives a threshold value of 4.61.

The significance of the threshold is that if we assume that the classification is independent of which of the two adjacent intervals an instance belongs to, there is a 90% probability that χ^2 will be less than 4.61.

If χ^2 is less than 4.61 it is taken as supporting the hypothesis of independence at the *90% significance level* and the two intervals are merged. On the other hand, if the value of χ^2 is greater than 4.61 we deduce that the class and interval are not independent, again at the 90% significance level, and the two intervals are left unchanged.

8.4.1 Calculating the Expected Values and χ^2

For a given pair of adjacent rows (intervals) the value of χ^2 is calculated using the 'observed' and 'expected' frequency values for each combination of class and row. For this example there are three classes so there are six such combinations. In each case, the observed frequency value, denoted by O, is the frequency that

actually occurred. The expected value E is the frequency value that would be expected to occur by chance given the assumption of independence.

If the row is i and the class is j, then let the total number of instances in row i be $rowsum_i$ and let the total number of occurrences of class j be $colsum_j$. Let the grand total number of instances for the two rows combined be sum. Assuming the hypothesis that the class is independent of which of the two rows an instance belongs to is true, we can calculate the expected number of instances in row i for class j as follows. There are a total of $colsum_j$ occurrences of class j in the two intervals combined, so class j occurs a proportion of $colsum_j/sum$ of the time. As there are $rowsum_i$ instances in row i, we would expect $rowsum_i \times colsum_j/sum$ occurrences of class j in row i.

To calculate this value for any combination of row and class, we just have to take the product of the corresponding row sum and column sum divided by the grand total of the observed values for the two rows.

For the adjacent intervals labelled 8.7 and 12.1 in Figure 8.8 the six values of O and E (one pair of values for each class/row combination) are given in Figure 8.10(b).

Value of A	Frequency for class						Total observed
	$c1$		$c2$		$c3$		
	O	E	O	E	O	E	
8.7	6	4.79	0	0.74	1	1.47	7
12.1	7	8.21	2	1.26	3	2.53	12
Total	13		2		4		19

Figure 8.10(b) Observed and Expected Values for Two Adjacent Rows of Figure 8.8

The O values are taken from Figure 8.8 or Figure 8.10(a). The E values are calculated from the row and column sums. Thus for row 8.7 and class $c1$, the expected value E is $13 \times 7/19 = 4.79$.

Having calculated the value of O and E for all six combinations of class and row, the next step is to calculate the value of $(O - E)^2/E$ for each of the six combinations. These are shown in the Val columns in Figure 8.11.

The value of χ^2 is then the sum of the six values of $(O - E)^2/E$. For the pair of rows shown in Figure 8.11 the value of χ^2 is 1.89.

If the independence hypothesis is correct the observed and expected values O and E would ideally be the same and χ^2 would be zero. A small value of χ^2 would also support the hypothesis, but the larger the value of χ^2 the more reason there is to suspect that the hypothesis may be false. When χ^2 exceeds

Value of A	Frequency for class									Total observed
	$c1$			$c2$			$c3$			
	O	E	Val*	O	E	Val*	O	E	Val*	
8.7	6	4.79	0.31	0	0.74	0.74	1	1.47	0.15	7
12.1	7	8.21	0.18	2	1.26	0.43	3	2.53	0.09	12
Total	13			2			4			19

* Val columns give the value of $(O - E)^2/E$

Figure 8.11 O, E and Val values for two adjacent rows of Figure 8.8

the threshold value we consider that it is so unlikely for this to have occurred by chance that the hypothesis is rejected.

The value of χ^2 is calculated for each adjacent pair of rows (intervals). When doing this, a small but important technical detail is that an adjustment has to be made to the calculation for any value of E less than 0.5. In this case the denominator in the calculation of $(O - E)^2/E$ is changed to 0.5.

The results for the initial frequency table are summarised in Figure 8.12(a).

Value of A	Frequency for class			Total	Value of χ^2
	$c1$	$c2$	$c3$		
1.3	1	0	4	5	3.11
1.4	0	1	0	1	1.08
1.8	1	1	1	3	2.44
2.4	6	0	2	8	3.62
6.5	3	2	4	9	4.62
8.7	6	0	1	7	1.89
12.1	7	2	3	12	1.73
29.4	0	0	1	1	3.20
56.2	2	4	0	6	6.67
87.1	0	1	3	4	1.20
89.0	1	1	2	4	
Total	27	12	21	60	

Figure 8.12(a) Initial Frequency Table with χ^2 Values Added

In each case, the χ^2 value given in a row is the value for the pair of adjacent intervals comprising that row and the one below. No χ^2 value is calculated for the final interval, because there is not one below it. As the table has 11 intervals there are 10 χ^2 values.

ChiMerge selects the smallest value of χ^2, in this case 1.08, corresponding to the intervals labelled 1.4 and 1.8 and compares it with the threshold value, which in this case is 4.61.

The value 1.08 is less than the threshold value so the independence hypothesis is supported and the two intervals are merged. The combined interval is labelled 1.4, i.e. the smaller of the two previous labels.

This gives us a new frequency table, which is shown in Figure 8.12(b). There is one fewer row than before.

Value of A	Frequency for class			Total
	$c1$	$c2$	$c3$	
1.3	1	0	4	5
1.4	1	2	1	4
2.4	6	0	2	8
6.5	3	2	4	9
8.7	6	0	1	7
12.1	7	2	3	12
29.4	0	0	1	1
56.2	2	4	0	6
87.1	0	1	3	4
89.0	1	1	2	4

Figure 8.12(b) ChiMerge: Revised Frequency Table

The χ^2 values are now calculated for the revised frequency table. Note that the only values that can have changed from those previously calculated are those for the two pairs of adjacent intervals of which the newly merged interval (1.4) is one. These values are shown in bold in Figure 8.12(c).

Now the smallest value of χ^2 is 1.20, which again is below the threshold value of 4.61. So intervals 87.1 and 89.0 are merged.

ChiMerge proceeds iteratively in this way, merging two intervals at each stage until a minimum χ^2 value is reached which is greater than the threshold, indicating that an irreducible set of intervals has been reached. The final table is shown as Figure 8.12(d).

The χ^2 values for the two remaining pairs of intervals are greater than the threshold value. Hence no further merging of intervals is possible and the discretisation is complete. Continuous attribute A can be replaced by a categorical attribute with just three values, corresponding to the ranges (for the 90% significance level):

Value of A	Frequency for class			Total	Value of χ^2
	$c1$	$c2$	$c3$		
1.3	1	0	4	5	**3.74**
1.4	1	2	1	4	**5.14**
2.4	6	0	2	8	3.62
6.5	3	2	4	9	4.62
8.7	6	0	1	7	1.89
12.1	7	2	3	12	1.73
29.4	0	0	1	1	3.20
56.2	2	4	0	6	6.67
87.1	0	1	3	4	1.20
89.0	1	1	2	4	
Total	27	12	21	60	

Figure 8.12(c) Revised Frequency Table with χ^2 Values Added

Value of A	Frequency for class			Total	Value of χ^2
	$c1$	$c2$	$c3$		
1.3	24	6	16	46	10.40
56.2	2	4	0	6	5.83
87.1	1	2	5	8	
Total	27	12	21	60	

Figure 8.12(d) Final Frequency Table

$$\begin{array}{lcl} 1.3 & \leq & A < 56.2 \\ 56.2 & \leq & A < 87.1 \\ A & \geq & 87.1 \end{array}$$

A possible problem with using these ranges for classification purposes is that for an unseen instance there might be a value of A that is substantially less than 1.3 (the smallest value of A for the training data) or substantially greater than 87.1. (Although the final interval is given as $A \geq 87.1$ the largest value of A for the training data was just 89.0.) In such a case we would need to decide whether to treat such a low or high value of A as belonging to either the first or last of the ranges as appropriate or to treat the unseen instance as unclassifiable.

8.4.2 Finding the Threshold Value

Threshold values for the χ^2 test can be found in statistical tables. The value depends on two factors:

1. The significance level. 90% is a commonly used significance level. Other commonly used levels are 95% and 99%. The higher the significance level, the higher the threshold value and the more likely it is that the hypothesis of independence will be supported and thus that the adjacent intervals will be merged.
2. The number of *degrees of freedom* of the contingency table. A full explanation of this is outside the scope of this book, but the general idea is as follows. If we have a contingency table such as Figure 8.10(a) with 2 rows and 3 columns, how many of the $2 \times 3 = 6$ cells in the main body of the table can we fill independently given the marginal totals (row and column sums)? The answer to this is just 2. If we put two numbers in the $c1$ and $c2$ columns of the first row ($A = 8.7$), the value in the $c3$ column of that row is determined by the row sum value. Once all three values in the first row are fixed, those in the second row ($A = 12.1$) are determined by the three column sum values.

In the general case of a contingency table with N rows and M columns the number of independent values in the main body of the table is $(N-1) \times (M-1)$. For the ChiMerge algorithm the number of rows is always two and the number of columns is the same as the number of classes, so the number of degrees of freedom is $(2-1) \times (\text{number of classes} - 1) = \text{number of classes} - 1$, which in this example is 2. The larger the number of degrees of freedom is, the higher the threshold value.

For 2 degrees of freedom and a 90% significance level, the χ^2 threshold value is 4.61. Some other values are given in Figure 8.13 below.

Choosing a higher significance level will increase the threshold value and thus may make the merging process continue for longer, resulting in categorical attributes with fewer and fewer intervals.

8.4.3 Setting *minIntervals* and *maxIntervals*

A problem with the ChiMerge algorithm is that the result may be a large number of intervals or, at the other extreme, just one interval. For a large training set an attribute may have many thousands of distinct values and the method may produce a categorical attribute with hundreds or even thousands of values. This is likely to be of little or no practical value. On the other hand,

Degrees of freedom	90% Significance level	95% Significance level	99% Significance level
1	2.71	3.84	6.64
2	4.61	5.99	9.21
3	6.25	7.82	11.34
4	7.78	9.49	13.28
5	9.24	11.07	15.09
6	10.65	12.59	16.81
7	12.02	14.07	18.48
8	13.36	15.51	20.09
9	14.68	16.92	21.67
10	15.99	18.31	23.21
11	17.28	19.68	24.72
12	18.55	21.03	26.22
13	19.81	22.36	27.69
14	21.06	23.69	29.14
15	22.31	25.00	30.58
16	23.54	26.30	32.00
17	24.77	27.59	33.41
18	25.99	28.87	34.80
19	27.20	30.14	36.19
20	28.41	31.41	37.57
21	29.62	32.67	38.93
22	30.81	33.92	40.29
23	32.01	35.17	41.64
24	33.20	36.42	42.98
25	34.38	37.65	44.31
26	35.56	38.89	45.64
27	36.74	40.11	46.96
28	37.92	41.34	48.28
29	39.09	42.56	49.59
30	40.26	43.77	50.89

Figure 8.13 χ^2 Threshold Values

if the intervals are eventually merged into just one that would suggest that the attribute value is independent of the classification and the attribute would best be deleted. Both a large and a small number of intervals can simply reflect setting the significance level too low or too high.

Kerber [1] proposed setting two values, *minIntervals* and *maxIntervals*. This form of the algorithm always merges the pair of intervals with the lowest value of χ^2 as long as the number of intervals is more than *maxIntervals*. After that the pair of intervals with the smallest value of χ^2 is merged at each stage until *either* a χ^2 value is reached that is greater than the threshold value *or* the number of intervals is reduced to *minIntervals*. In either of those cases the algorithm stops. Although this is difficult to justify in terms of the statistical theory behind the χ^2 test it can be very useful in practice to give a manageable number of categorical values. Reasonable settings for *minIntervals* and *maxIntervals* might be 2 or 3 and 20, respectively.

8.4.4 The ChiMerge Algorithm: Summary

With the above extension, the ChiMerge algorithm is summarised in Figure 8.14.

8.4.5 The ChiMerge Algorithm: Comments

The ChiMerge algorithm works quite well in practice despite some theoretical problems relating to the statistical technique used, which will not be discussed here (Kerber's paper [1] gives further details). A serious weakness is that the method discretises each attribute independently of the values of the others, even though the classifications are clearly not determined by the values of just a single attribute.

Sorting the values of each continuous attribute into order can be a significant processing overhead for a large dataset. However this is likely to be an overhead for any method of discretisation, not just ChiMerge. In the case of ChiMerge it needs to be performed only once for each continuous attribute.

1. Set values of *minIntervals* and *maxIntervals* ($2 \leq minIntervals \leq maxIntervals$).
2. Decide on a significance level (say 90%). Using this and the number of degrees of freedom (i.e. number of classes -1) look up the threshold value to use.
3. For each continuous attribute in turn:
(a) Sort the values of the attribute into ascending numerical order.
(b) Create a frequency table containing one row for each distinct attribute value and one column for each class. Label each row with the corresponding attribute value. Enter the number of occurrences of each attribute value/class combination in the training set in the cells of the table.
(c) If (number of rows = *minIntervals*) then stop, otherwise go on to next step.
(d) For each pair of adjacent rows in the frequency table in turn:
For each combination of row and class:

(i) calculate O, the observed frequency value for that combination

(ii) calculate E, the expected frequency value for that combination, from the product of the row and column sums divided by the total number of occurrences in the two rows combined

(iii) calculate the value of $(O-E)^2/E$ *

Add the values of $(O-E)^2/E$ to give χ^2 for that pair of adjacent rows.
(e) Find the pair of adjacent rows with the lowest value of χ^2.
(f) If the lowest value of χ^2 is less than the threshold value OR (number of rows $>$ *maxIntervals*), merge the two rows, setting the attribute value label for the merged row to that of the first of the two constituent rows, reduce the number of rows by one and go back to step (c). Otherwise stop.

---

* If $E < 0.5$, replace E in the denominator of this formula by 0.5.

Figure 8.14 The ChiMerge Algorithm

8.5 Comparing Global and Local Discretisation for Tree Induction

This section describes an experiment aimed at comparing the effectiveness of using the local discretisation method for TDIDT described in Section 8.3 with that of using ChiMerge for global discretisation of continuous attributes followed by using TDIDT for rule generation, with all attributes now categorical.

For convenience information gain will be used for attribute selection throughout.

Seven datasets are used for the experiment, all taken from the UCI Repository. Basic information about each dataset is given in Figure 8.15.

Dataset	Instances	Attributes		Classes
		Categ.	Contin.	
glass	214	0	9	7
hepatitis	155	13	6	2
hypo	2514	22	7	5
iris	150	0	4	3
labor-ne	40	8	8	2
pima-indians	768	0	8	2
sick-euthyroid	3163	18	7	2

Figure 8.15 Datasets Used in ChiMerge Experiments

The version of ChiMerge used is a re-implementation by the present author of Kerber's original algorithm.

The value of each set of classification rules can be measured by the number of rules generated and the percentage of instances that they correctly classify. The methodology chosen for these experiments is *10-fold cross-validation*. First the training set is divided into 10 groups of instances of equal size. TDIDT is then run 10 times with a different 10% of the instances omitted from the rule generation process for each run and used subsequently as an unseen test set. Each run produces a percentage of correct classifications over the unseen test set and a number of rules. These figures are then combined to give an average number of rules and the percentage of correct classifications. The 'default to largest class' strategy is used throughout.

Figure 8.16 shows the results of applying TDIDT directly to all the datasets, compared with first using ChiMerge to discretise all the continuous attributes globally (90% significance level).

The percentage of correct classifications for the global discretisation approach is comparable with those achieved by local discretisation. However, local discretisation seems to produce an appreciably smaller number of rules, at least for these datasets. This is particularly the case for the *pima-indians* and *sick-euthyroid* datasets.

On the other hand, the global discretisation approach has the considerable advantage that the data only has to be discretised once and can then be used as the input to any data mining algorithm that accepts categorical attributes, not only TDIDT.

Dataset	Local discretisation		Global discretisation	
	Number of rules	Correct %	Number of of rules	Correct %
glass	38.3	69.6	88.2	72.0
hepatitis	18.9	81.3	42.0	81.9
hypo	14.2	99.5	46.7	98.7
iris	8.5	95.3	15.1	94.7
labor-ne	4.8	85.0	7.6	85.0
pima-indians	121.9	69.8	328.0	74.0
sick-euthyroid	72.7	96.6	265.1	96.6

Figure 8.16 TDIDT with Information Gain. Local Discretisation v Global Discretisation by ChiMerge (90% significance level). Results from 10-fold Cross-validation

8.6 Chapter Summary

This chapter looks at the question of how to convert a continuous attribute to a categorical one, a process known as *discretisation*. This is important as many data mining algorithms, including TDIDT, require all attributes to take categorical values.

Two different types of discretisation are distinguished, known as local and global discretisation. The process of extending the TDIDT algorithm by adding local discretisation of continuous attributes is illustrated in detail, followed by a description of the ChiMerge algorithm for global discretisation. The effectiveness of the two methods is compared for the TDIDT algorithm for a number of datasets.

8.7 Self-assessment Exercises for Chapter 8

1. Using the amended form of the rule given in Section 8.3.2, what are the candidate cut points for the continuous attribute *humidity* in the *golf* training set given in Chapter 4?
2. Starting at Figure 8.12(c) and the resulting merger of intervals 87.1 and 89.0, find the next pair of intervals to be merged.

Reference

[1] Kerber, R. (1992). ChiMerge: discretization of numeric attributes. In *Proceedings of the 10th national conference on artificial intelligence* (pp. 123–128). Menlo Park: AAAI Press.