4.3 Distance-based Algorithms

Each item that is mapped to the same class may be thought of as more similar to the other items in that class than it is to the items found in other classes. Therefore, similarity measures may be used to identify the "alikeness" of different items in the database. The concept of similarity measure was introduced in Chapter 2 with respect to IR retrieval. Certainly, the concept is well known to anyone who has performed Internet searches using a search engine. In these cases, the set of web pages represent the whole database and these are divided into two classes: those that answer your query and those that do not. Those that answer your query should be more alike than those that do not answer your query. The similarity in this case is defined by the query you state, usually a keyword list. Thus, the retrieved pages are similar because they all contain the keyword list you have specified.

The idea of similarity measure can be abstracted and applied to more general classification problems. The difficulty lies in how the similarity measures are defined and applied to the items in the database. Since most similarity measures assume numeric values, they might be difficult to use for more general or abstract data types. A mapping from the attribute domain to a subset of integers may be used.

Using a similarity measure for classification where the classes are predefined is somewhat simpler than using a similarity measure for clustering where the classes are not known in advance. Again, think of the IR example, each IR query provides the class definition in the form of the IR query itself. So the classification problem then becomes one of determining similarity not among all tuples in the database but between each tuple and the query. This makes the problem of an $O(n)$ problem rather than an $O(n^2)$ problem.

4.3.1 Simple Approach

Using the IR approach, if we have a representative of each class, we can perform classification by assigning each tuple to the class to which it is most similar. We assume here that each tuple $t_i$ in the database is defined as a vector $(t_{i1}, t_{i2}, \ldots, t_{ik})$ of numeric values. The classification problem is then restated in Definition 4.2

**DEFINITION 4.2:** Given a database $D = \{t_1, t_2, \ldots, t_n\}$ of tuples where each tuple $t_i = \{t_{i1}, t_{i2}, \ldots, t_{ik}\}$ contains numeric values and a set of classes $C = \{C_1, \ldots, C_m\}$ where each class $C_j = \{C_{j1}, C_{j2}, \ldots, C_{jk}\}$ has numeric values, the classification problem is to assign each $t_i$ to the class $C_j$ such that $\text{sim}(t_i, C_j) \geq \text{sim}(t_i, C_l) \forall C_l \in C$ where $C \neq C_j$

To calculate these similarities measures, the representative vector for each must be determined. Referring to the three classes in Figure 4.1(a), we can determine a representative for each class by calculating the center of each region. Thus class A is represented by (4,7,5), class B by (2,2,5), and class C by (6,2,5). A simple classification technique then would be to place each item in
the class where it is most similar to the center of the class. The representative
for the class may be found in other ways. For example, in pattern recognition
problems, a predefined pattern can be used to represent each class. Once a simi-
larity measure is defined, each item to be classified will be compared to each
pre-defined pattern. The item will be placed in the class with the largest simi-
larity value. Algorithm 4.1 illustrates a straightforward distance-based approach
assuming that each class, $c_i$, is represented by its center or centroid. In the
algorithm we use to the center for its class. Since each tuple must be compared
to the center for a class and there are a fixed number of classes, the complexity
to classify one tuple is $O(n)$.

\begin{algorithm}
\textbf{ALGORITHM 4.1}
\begin{algorithmic}
\State \textbf{Input:}
\State $c_1,\ldots, c_m$ //centers for each class
\State $t$ //Input tuple to classify
\State \textbf{Output:}
\State $c$ //class to which $t$ is assigned
\State \textbf{Simple distance-based algorithm.}
\State dist = $\infty$
\For{$i := 1$ to $m$}
\If{$distsc(c_i, t) < \text{dist}$}
\State $c = i$;
\State dist = $\text{dist}(c_i, t)$;
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

Figure 4.9 illustrates the use of this approach to perform classification
using the data found in Figure 4.1. The three large dark circles are the class
representatives for the three classes. The dashed lines show the distance from
each item to the closest center.

4.3.2 K Nearest Neighbors

One common classification scheme based on the use of distance measures
is that of the K nearest neighbors (KNN). The KNN technique assumes that
the entire training set includes not only the data in the set but also the desired
classification for each item. In effect, the training data become the model.
When a classification for each item. In effect the training data become the
model. When a classification to be made for a new item, its distance to each
item in the training set must be determined. Only the $K$ closest entries in the
training set are considered further. The new item is then placed in the class that
contains the most items for this set of $K$ closest items. Figure 4.10 illustrates
the process used by KNN. Here the points in the training set are shown and
the three closest items in the training set are shown; $t$ will be placed in the
classification most of these are members.
Algorithm 4.2 outlines the user of the KNN algorithm. We use $T$ as the training data. Since each tuple to be classified must be compared to each data in the training data, if there are $q$ elements in the training set, this is $O(q)$ elements to be classified, this becomes an $O(nq)$ problem. Given that the area of a constant size, this can be viewed as a problem.

**ALGORITHM 4.2**

Input:

$T$ //Training data
$K$ //Number of neighbours
$t$ //Input tuple to classify

Output:

$c$ //Class to which $t$ is assigned

KNN algorithm:

//Algorithm to classify tuple using KNN
$N = \emptyset$;
//Find set of neighbors, $N$, for $t$
for each $d$ e $t$ do
    if $|N| \leq K$, then
\[ N = N \cup \{ d \}; \]

else
    
    if
        
        begin
            
            \[ N = N - \{ u \}; \]
            
            \[ N = N \cup \{ d \}; \]
        
    end
    
    //Find class for classification
    
    \[ c = \text{class to which the most } u \in N \text{ are classified}; \]

Example 4.6 illustrates this technique using the sample data from Table 4.1. The KNN technique is extremely sensitive to the value of K. A rule of thumb is that \( K \leq \sqrt{\text{number of training items}} \). For this example, the value is 3.46. Commercial algorithms often default the value of 10.

Example 4.6

Using the sample data from Table 4.1 and the output 1 classification as the training set output value, we classify the tuple (Pat, F, 1.6). Only the height is used for distance calculation so that both the Euclidean and Manhattan distance measures yield the same results; that is, the distance is simply the absolute value of the difference the values. Suppose that \( K = 5 \) is given. We then have the K nearest neighbor to the input tuple are \{ (Kristina, F, 1.6), (Kathy, F, 1.6), (Stepatanie, F, 1.7), (Dave, M, 1.7), (Wynette, F, 1.75) \}. Of these five items, four are classified as short and one as medium. Thus the KNN will classify Pat as short.

4.4 Decision Tree-Based Algorithms

The decision tree approach is most useful in classification problems. With this technique, a tree is constructed to model the classification process. Once the tree is built, it is applied to each tuple in the database and results in a classification for tuple. There are two basic steps in the technique: building the tree and applying the tree to the database. Most research has focused on how to build effective trees as the application process is straightforward.

The decision tree approach to classification is to divide the search space into angular regions. A tuple is classified based in the region into which it falls. A decision tree used in classification is contained in Definition 4.3. There are native definitions; for example, in a binary DT the nodes could be labelled predicates themselves and each arc would be labelled with yes or no.

Definition 4.3: Given a database \( D = \{ t_1, t_2, \ldots, t_n \} \) where \( t_n = (t_{i1}, t_{i2}, \ldots, t_{ik}) \) the database scheme contains the following attributes \( \{ A_1, \ldots, A_h \} \). Also is the set of classes \( C = \{ C_1, \ldots, C_m \} \). A decision tree (DT) or classification is a tree associated with \( D \) that has the following properties:

- Each internal node is labelled with an attribute, \( A_i \).
- Each arc is labelled with a predicate that can be applied to the attribute associated with the parent.
Each leaf node is labelled with a class, $C_j$.

Solving the classification problem using decision trees is a two-step process:

1. Decision tree induction: Construct a DT using training data.
2. For each $x_i \in D$, apply the DT to determine its class.

Based on our definition of the classification process, Definition 4.1 the construction of the DT represents the logic needed to perform the mapping. Thus, it is implicitly defined mapping. Using the DT shown in Figure 3.5 from Chapter 3, the classification sample data found in Table 4.1 is that shown in the column labelled Output2. A DT is relatively straightforward, we do not consider the second part of the further. Instead we focus on algorithms to construct decision trees. Several algorithms are surveyed in the following subsections.

There are many advantages to the use of DTs for classification. DTs are easy to use and efficient. Rules can be generated that are easy to interpret and understand. They scale well for large databases because tree size is independent of the database size. Each tuple in the database must be filtered through the tree. The time proportional to the height of the tree, which is fixed. Trees can construct data with many attributes.

Disadvantages also exist for DT algorithms. First, they do not easily handle continuous data. These attributes domains must be divided into categories to be hidden. The approach used is that domain space in divided into rectangular regions is seen in figure 4.1(a). Not all classifications problems are of this type. The data handled by DTs. handling missing data is difficult in Figure 2.4(a) in Chapter 2. Since the DT is constructed from the training data, overfitting occur. This can be overcome via tree pruning. Finally, correlations among attributes the database are ignored by the DT process.

ALGORITHM 4.3
Input:
$D$ //Training data
Output
$T$ //Decision tree
DTBuild algorithm:
//Simplistic algorithm to illustrate naive approach to building DT.
$T = \emptyset$;
Determine best splitting criterion;
$T = \text{Create a root node and label with splitting attribute};$
$T = \text{add arc to root node for each split predicate and label};$
for each arc do
  $D = \text{Database created by applying slitting predicate to } D;$
  if stopping point reached for this path, then
    $T' = \text{Create leaf node and label with appropriate class};$
  else
    $T' = \text{DTBuild(D)}$;
T = Add $T'$ to arc;
There have been many decision tree algorithms. We illustrate the tree building base in the simplistic DTBuild Algorithm 4.3. Attributes in the database schema that will be used to label nodes in the tree and around which the divisions will take place are the splitting attributes. The predicates by which the arcs in the tree are labelled are called splitting attributes. The predicates by which the arcs in the tree are labelled are called splitting predicates. In the decision trees shown in Figure 4.11, the splitting attributes are \{gender, height\}. The splitting predicates for gender are \{female, male\} while those for height include \{≤ 1.3m, 1.8m, 1.5m, 2m\}. The splitting predicates for height differ based on whether the tuple is for a male or a female. This recursive algorithm builds the tree in a top-down fashion by examining the training data. Using the initial training data, the best splitting attribute is chosen first. Algorithms differ in how they determine the best attribute and its best predicates to use for splitting. Once this has been determined the node and its arcs are created and added to the created tree. The algorithm continues recursively by adding new sub trees to each branching arc. The algorithm terminates when some stopping criteria is reached. Again each algorithm determines when to stop the tree differently. One simple approach would be stop when the tuples in the reduced training set all belong to the same class. This class is then used to label the leaf node created.

Note that the major factors in the performance of the DT building algorithm are the size of the training set and how the best splitting attribute is chosen. The following issues are faced by the most DT algorithms:

Choosing splitting attributes: Which attributes to use for splitting attributes impacts the performance applying the built DT. Some attributes are better than others. In the data shown in Table 4.1, the name attribute definitely should not be used and the gender may or may not be used and the gender may or may not be used. The choice of attributes involves not only an examination of the data in the training set but also the informed input.
Ordering of splitting attributes: The order in which the attributes are choosen is also important. In Figure 4.11(a) the gender attributes is chosen first. Alternatively the height attribute could be choosen first. As seen in Figure 4.11(b), in this case the height attribute must be examined a second time, requiring unnecessary comparisions.

Splits: Associated with the ordering of the attributes is the number of splits to take. With some attributes the domain is small, so the number of splits is obvious based on the domain (as with the gender attributes) However, if the domain is continuous or has a large number of values, the number of splits to use is not sufficient.

Tree Structure: To improve the performance of applying the tree to classification a balanced tree with the fewest levels is desirable. However in this case, more complicated comparisons with multiway branching may be needed. Some algorithms build only binary trees.

Stopping criteria: The criterion of the tree definitely stops when the training
data are perfectly classified. There may be situations when stopping earlier would be desirable to prevent the creation of larger trees. This is a trade-off between accuracy of classification and performance. In addition, stopping earlier may be performed to prevent overfitting. It is even conceivable that more levels than needed would be created in a tree if it is known that there are data distributions not represented in the training data.

Training data: The structure of the DT created depends on the training data. If the training data set is too small, then the generated tree might not be specific enough to work properly with more general data. If the training data set is too large then the created tree may overfit.

Pruning: Once a tree is constructed, some modifications to the tree might be needed to improve the performance of the tree during the classification phase. The pruning phase might remove redundant comparisons or remove subtrees to achieve better performance.

To illustrate some of these design decisions, Figure 4.1 shows four different decision trees that can be used to classify persons according to height. The first tree is a duplicate of that from Chapter 3. The first three trees in this figure all perform the same classification. However, they all perform it differently. Undeneath each tree is a table showing the logical divisions used by the associated tree for classification. A nice feature of Figure 4.11(a) is that it is balanced. The tree is of same depth for any path from the tree in (b) greater than that of any of the others, implying a slightly worse behavior when used for classification. However, all of these factors impact the time required to do the actual classification. These may not be crucial performance issues unless the database is extremely large. In that case, a balanced shorter tree would be desirable. The tree shown in Figure 4.11(d) does not represent the same classification logic as others.

The training data and the tree induction algorithm determine the tree shape. Thus, the best shaped tree that performs perfectly on the training set is desirable. Some algorithms create only binary trees. Binary trees are easily created, but tend to be deeper. The performance results when applying these types of trees for classification may be worse because more comparisons are needed. However, since these comparisons are simpler than those that require multway branches, the ultimate performance may be comparable.

The DT building algorithm may initially build the tree and then prune it for more effective classification. With pruning techniques, portions of the tree may be removed or combined to reduce the overall size of the tree. Portions of the tree relating to classification using an unimportant attribute may be removed. This sort of change with a node close to the root could ripple down to create major changes in the lower parts of values of the name attribute, all nodes labelled with that attribute would be real lower-level nodes would move up or be combined in some way. The approach this could become quite complicated. In the case of overfitting lower level subtrees preventing a tree from becoming too large. A second approach prunes the trees to be built.

The time and space complexity of DT algorithm depends on the size of
the data, \( q \); the number of attributes, \( h \); and the shape of the resulting tree. In the DT that is built may be quite deep and not bushy. As the tree is built for these nodes, each attribute will be examined to determine if it is the best. The time complexity to build the tree of \( O(h \log q) \). The time to classify a data of size \( n \) is based on the height of the tree. Assuming a height of \( O(\log q) \) this is \( O(n \log q) \).

In the following subsections we examine several popular DT approaches.

### 4.4.1 ID3

The ID3 techniques to building a decision tree is based on information theory and to minimize the expected number of comparisons. The basic idea of the induction algorithm is to ask questions whose answers provide the most information. That is the intuitive approach taken by adults when playing the Twenty Question game. The first question asked could be “is this thing alive?” The first question divides the search space into two large search areas, while the second performs little divisions of the space. The basic strategy is to choose splitting attributes with the highest information gain first. The information associated with an attribute value is related to the probability of looking at the Twenty Questions example, the child’s question divides the set into two sets. One set (Daddy) has an infinitesimal probability associated with it, while the question that adult makes divides the set into two sub sets with almost equal probability of occurring.

The concept used to quantify information is called Entropy. Entropy is the measure of amount of uncertainty or surprise or randomness in the set of data when all data in a set belong to a single class, there is no uncertainty. In this entropy is zero. The objective of decision tree classification is to iteratively take the given data into subsets where all elements in each final subset belong to the same class. In figure 4.12 (a, b and c) will help to explain the concept. Figure \( \log(1/p) \) as the probability \( p \) ranges from 0 to 1. This intuitively is amount of surprise based on the probability when \( p = 1 \) there is no surprise that means that if an event of probability of 1 and you are told that the event you would not be surprised. As \( p \) 0 the surprise increases. When we do the divide and conquer approach such as that used with decision trees, the decision multiple possibilities whose sum is 1. In the Twenty Questions game, the \( P \). To measure the information with this division, we must be able to combine the information associated with events. That is, we must be able to calculate the average information associated with the division. This can be performed by adding the two values together and which is expected information based on probability of an event. To determine the expected information associated with two events we add the individual values together. This function \( p \log(1/p) + (1-p) \log(1/(1-p)) \) is plotted in Figure 4.12(c). Note that the maximum occurs when the two possibilities are equal. This supports our intuitive idea that the more sophisticated questions posed by the adult are better than those posed by the child.
The formal definition of entropy is shown in the definition 4.4. The value of the entropy is between 0 and 1 and reaches a maximum when the possibilities are all the same.

**DEFINITION 4.4**

Given probabilities \( p_1, p_2, \ldots, p_s \) where \( \sum_{i=1}^{s} p_i = 1 \), entropy is defined as

\[
H(p_1, p_2, \ldots, p_s) = \sum_{i=1}^{s} p_i \log(1/p_i)
\]

Given a database state, \( D \), \( H(D) \) finds the amount of order in that state. When that state is split into \( s \) new states \( S = \{D_1, D_2, \ldots, D_s\} \), we can again look at the entropy of these states. Each step in ID3 chooses the state that orders splitting the most. A database state is completely ordered if all tuples in it are in the same class. ID3 chooses the splitting attribute with the highest gain in information, where gain is defined as the difference between how much information is needed to make a correct classification before the split versus how much information is needed after the split. Certainly, the split should reduce the information needed by the largest amount. This is calculated by determining the entropies of the original dataset and the weighted sum of the entropies from each of the individual datasets. The entropies of the split databases are weighted by the function of the dataset being placed in the division. The ID3 algorithm calculates the gain of a particular split by the following formula:

\[
\text{Gain}(D,S) = H(D) - \sum_{i=1}^{s} P(D_i)H(D_i)
\]

Example 4.7 and associated Figure 4.13 illustrate this process using the height example. In this example, six divisions of the possible ranges of heights are used. This division into ranges is needed when the domains of an attribute is continuous or consists of many possible values. While the choice of these divisions is somewhat arbitrary, a domain expert should be able to perform the task.
Example 4.7

The beginning state of the training data in Table 4.1 (with the output1 classification) is that (4/15) are short, (8/15) are medium, and (3/15) are tall. Thus, the entropy of the starting set is

$$4/15 \log(15/4) + 8/15 \log(15/8) + 3/15 \log(15/3) = 0.4384$$

Choosing the gender as the splitting attribute, there are nine tuples that are F and six that are M. The entropy of the subset that are F is

$$3/9 \log(9/3) + 6/9 \log(9/6) = 0.2764$$

whereas that for the M subset is

$$1/6 \log(6/1) + 2/6 \log(6/2) + 3/6 \log(6/3) = 0.43923$$

The ID3 algorithm must determine what the gain in information is by using this split. To do this, we calculate the weighted sum of these last two entropies to get

$$((9/15) 0.2764 ) + ((6/15) 0.4392) = 0.34152$$

The gain in entropy by using the gender attribute is thus

$$= 0.4384 - 0.34152 = 0.09688$$

Looking at the height attribute, we have two tuples that are 1.6, two 1.7, one is 1.75, two are 1.8, one is 1.88, two are 1.9, one is 1.95, one is 2, one is 2.1 and one is 2.2 Determining the split values for the height is not easy. Even though the training dataset has these 11 values, we know that there will be many more. Just as with continuous data, we divide into ranges:

$$(0,1.6],(1.6,1.7],(1.7,1.8],(1.8,1.9],(1.9,2.0],(2.0,\infty)$$

There are 2 tuples in the first division with entropy $$(2/2(0) + 0 +0) = 0.2$$ in $$(1.6,1.7]$$ with entropy $$(2/2(0) + 0 +0) = 0.3$$ in $$(1.7,1.8]$$ with entropy $$(0+3/3(0) + 0) = 0.4$$ in $$(1.8,1.9]$$ with entropy $$(0+4/4(0)+0) = 0.2$$ in $$(1.9,2.0]$$ with entropy $$(0+1/2)0.301)+1/2(0.301)) = 0.301$$ and two in the last with entropy $$(0+0+2/2(0)) = 0$$. All of these states are completely ordered and thus an entropy of 0 except for the (1.9,2.0] state. The gain in entropy by using the height attribute is thus

$$0.4384 - 2/15(0.301) = 0.3983$$

Thus, this has the greater gain, and we choose this over gender as the first splitting attribute. Within this division there are two males, one medium and one tall. This has occurred because this grouping too large. A further subdivision on height is needed and this generates DT seen in the figure.

Figure 4.13(a) illustrates a problem in that the tree has multiple splits with identical results. In addition, there is a subdivision of range (1.9,2.0], Figure 4.13(b) shows an optimized version of the tree.

C4.5 and C5.0

The decision tree algorithm C4.5 improves ID3 in the following ways:

Missing data: When the decision tree is built, missing data are simply ignored. That is, the gain ratio is calculated by looking only at the other records that have a value for the attribute. To classify a record with a missing variable
the value for that item can be predicted based on what is known about the attribute values for that record.

Continuous data: The basic idea is to divide the data into ranges based on the attribute values for that item that are found in the training sample.

Pruning: There are two primary pruning strategies proposed in C4.5:
- With subtree replacement, a subtree is replaced by a leaf node if this replacement results in an error rate close to that of the original tree. Subtree replacement works from the bottom of the tree up to the root.
- Another pruning strategy, called subtree raising, replaces a subtree by its most used subtree. Here a subtree is raised from its current location to a node higher up the tree. Again, we must determine the increase in error rate for this replacement.

Results: C4.5 allows classification via either decision tree or rules generated from them. In addition, some techniques to simplify complex rules are proposed. One approach is to replace the left-hand side of a rule by simpler version if all records in the training set are treated indetically. An otherwise type of rule can be used to indicate what should be done if no other rules apply.

Splitting: The ID3 approach favors attributes with many divisions and thus may tuple in the training set would be the best because there would be only one tuple for each division. An improvement can be made by taking into account the cardinality of each division. This approach uses the Gain Ratio as opposed to Gain. The Gain Ratio is defined as

$$\text{GainRatio}(D,S) = \frac{\text{Gain}(D,S)}{\text{H}(\frac{|D_1|}{|D|}, \ldots, \frac{|D_s|}{|D|})}$$

For splitting purposes, C4.5 uses the largest Gain Ratio that ensures a larger than average information gain. This is to compensate for the fact that the Gain Ratio value is skewed toward splits where the size of one subset is close to that of the starting one. Example 4.8 shows the calculation of Gain Ratio for the first split in Example 4.7.

Example 4.8
To calculate the Gain Ratio for the gender split, we first find the entropy associated with the split ignoring classes.

$$H(9/15, 6/15) = 9/15 \log(15/9) + 6/15 \log(15/6) = 0.292$$

This gives the Gain Ratio value for the gender attribute as

$$0.09688/0.292 = 0.332$$

The entropy for the split on height is


C5.0 is a commercial version of C4.5 now widely used in many data mining packages such as Clementine and RuleQuest. It is targeted toward use with large dataset. The DT induction is close to that of C4.5, but the rule generation is different. Unlike C4.5, the precise algorithms used for C5.0 have not been divulged. C5.0 does include improvements to generate rules. Results show that C5.0 improves on memory usage by about 90 percent, runs between 5.7 and 240 times faster than C4.5, and produces more accurate rules.
One major improvement to the accuracy of C5.0 is based on boosting. Boosting is an approach to combining different classifiers. While boosting normally increases that time that it takes to run a specific classifier, it does improve the accuracy. The error rate has been shown to be less than half of that found with C4.5 on some dataset. Boosting does not always help when the training data contains a lot of noise. Each item in the training set is assigned a weight. The weight indicates the importance of this item to the classification. A classifier is constructed for each combination of weights used. Thus, multiple classifiers are actually constructed. When C5.0 performs a classification, each classifier is assigned a vote, voting is performed, and the target tuple is assigned to the classification and regression trees (CART). is a technique that generates a binary decision with ID3 entropy is used to a measure to choose the best splitting attribute and worst. Unlike ID3, however, where a child is created for each subcategory only two are created. The splitting is performed around what is determined to be the best. At each step, an exhaustive search is used to determine the best split, where it is defined by

\[
\Phi(s/t) = 2P_L P_R \sum_{j=1}^{m} | P(C_j|t_L) - P(C_j|t_R) |
\]

formula is evaluated at the current node , t, and for each possible splitting attribute partition s. Here L and R are used to indicate the left and right subtrees of the node of the tree. P_L and P_R are the probability of the tuple in the training set will be the left or right side of the tree. This is defined as (tuples in subtree)/(tuples in training set). We assume that right branch is taken on equality. P(C_j|t_L) or P(C_j|t_R) is the probability tuple is in this class. C_j and in the left or right subtree. This is defined as the (class j in subtree) / (the target node) At each step, only one criterion is chosen as the best over all criteria. Example 4.9 shows its use with the height example with output1 results.

EXAMPLE 4.9

Step is to determine the split attribute criterion for the first split. We again know that there are six subranges to consider with the height attribute. Using this we have the potential split values of 1.6,1.7,1.8,1.9,2.0 We thus have a choice points, which yield the following goodness measures:

<table>
<thead>
<tr>
<th>(Gender)</th>
<th>(1.6)</th>
<th>(1.7)</th>
<th>(1.8)</th>
<th>(1.9)</th>
<th>(2.0)</th>
</tr>
</thead>
</table>

Illustrated with gender attribute, CART forces that an ordering for the cart to be used. CART handles missing data by simply ignoring that record in calculating goodness of a split on that attribute. The tree stops growing when no split will the performance. Note that even though it is best for the training data, it is best for all possible data to be added in the future. The CATR algorithm gains in pruning strategy, which we will not discuss here.
4.4 Scalable DT Techniques

We briefly examine some DT techniques that address creation of DT’s for large datasets.

The SPRINT (Scalable PaRallelizable INduction of decision Trees) algorithm addresses the scalability issue by ensuring that the CART technique can be applied regardless of availability of main memory. In addition, it can be easily parallelized. With SPRINT, a gini index is used to find the best split. Here gini for a database $D$ is defined as

$$gini(D) = 1 - \sum (p_j)^2$$

where $p_j$ is the frequency of class $C_j$, in $D$. The goodness of a split of a $D$ into subsets $D_1$ and $D_2$ is defined by

$$gini_{split}(D) = \frac{n_1}{n} (gini(D_1)) + \frac{n_2}{n} (gini(D_2))$$

The split with the best gini value is chosen. Unlike the earlier approaches, SPRINT does not need to sort the data by goodness value at each node during the DT induction process. With continuous data, the split point chosen to be the midpoint of the every pair of consecutive values from the training set.

By maintaining aggregate metadata concerning database attributes, the RainForest approach allows a choice called the attribute-value class label group. The table summarizes for an attribute the count of entries per class or attribute value grouping. This the AVC table summarizes the information needed to determine splitting attributes. The size of the table is not proportional to the size of the database or training set, but rather to the product of the number of classes, unique attribute values, and potential splitting attributes. This reduction in size facilitates the scaling of DT induction algorithms to extremely large training sets. During the tree-building phase, the training data are scanned, the AVC is built and the best splitting attribute is chosen. The algorithm continues by splitting the training data and constructing AVC for next node.