Chapter 5

Induction of Decision Trees

5.1 Representing disjunctive concepts

Consider the problem domain described by the attribute-value language L (discussed in Chapter 2) and the following set of classified examples:

$E^+ = \{[red, circle], [blue, triangle], [blue, square]\}$
$E^- = \{[red, square], [red, triangle]\}$

The candidate elimination algorithm applied to these data cannot produce a correct hypothesis. This is because there exist positive examples, whose least generalization covers some negative examples. For example, there is no hypothesis covering both $[red, circle]$, and $[blue, square]$ and at the same time not covering the negative example $[red, square]$.

This problem is due to the very restricted language for the hypotheses we use in the candidate elimination algorithm. Clearly the above data require a concept description involving a *disjunction* between two subdomains in the hypothesis space.

A description which can cope with such data is the *decision tree*. A decision tree can be represented in various forms. Here is a decision tree for classification of the above training set shown in two ways:

1. *Tree.* Each node represents an attribute (e.g. color), and the branches from the node are the different choices of the attribute values. Clearly the branches represent disjunctive relations between attribute values (the color can be blue OR red, and both branches belong to the concept). The leaves of the tree actually represent the classification. Each one is marked YES or NO, depending on whether the particular choice of values along the path to this leaf specifies a positive or a negative example, correspondingly.

```
     color
       |
  ---------
  |         |
 blue      red
  |         |
 YES      shape
            |
    ----------------
   |        |       |
triangle  square  circle
   |        |       |
   NO       NO     YES
```

2. *Set of rules.* These rules actually represent the paths in the decision tree.

```
IF color = blue THEN YES
IF color = red AND shape = circle THEN YES
IF color = red AND shape = square THEN NO
IF color = red AND shape = triangle THEN NO
```

A natural question is "how can the decision tree generalize". The above tree is a typical example of generalization. The first left branch of the tree leads to a leaf, which is determined by fixing only one of the attributes (color). Thus we allow the other one (shape) to have any value and hence to cover a number of positive examples. Actually in the taxonomic language L thiswould be the concept $[blue, any\_shape]$.

5.2 Building a decision tree

There are many algorithms for building decision trees. Many of them refer to ID3 [Quinlan, 1986]. Actually it is a family of concept learning algorithms, called TDIDT (Top-Down Induction of Decision Trees), which originated from the Concept Learning System (CLS) of [Hunt, Marin and Stone, 1966].

The basic algorithm is the following [Dietterich et al, 1982]. Its input is a set of training instances E, and its output is a decision tree.

1. If all instances in E are positive, then create a YES node and halt. If all instances in E are negative, then create a NO node and halt. Otherwise, select (using some heuristic criterion) an attribute, A, with values $V_1...V_n$ and create the decision node

```
         A
         |
   -------------
  |   |  |      |
  |   |  |      |
 V1  V2  V3 ... Vn
```

2. Partition the training instances in E into subsets $E_1, E_2, ..., E_n$, according to the values of $A(\{V_1, V_2, ..., V_n\})$

3. Apply the algorithm recursively to each of the sets E_1, E_2 etc.

Here is an example how this algorithm works. Consider the following set E of 6 instances (the positive are marker with "+", and the negative – with "-"):

```
1. [red,circle] +
2. [red,square] -
3. [red,triangle] -
4. [blue,triangle] -
5. [blue,square] -
6. [blue,circle] -
```

Initially the set of instances E is just the complete sequence of instances. These are neither uniformly positive or uniformly negative so the algorithm selects an attribute A and creates a decision node. Assume that the *shape* attribute is chosen. It has possible values

$\{triangle, square, circle\}$. Therefore a decision node is created which has a branch corresponding to each value.

The set E is now partitioned into subsets E_1, E_2 etc. according to the possible values of "shape". Instances with $shape = triangle$ all end up in one subset, instances with $shape = circle$ all end up in another and so on.

The algorithm is now applied recursively to the subsets E_1, E_2 etc. Two of these now contain single instances. The set of instances with $shape = triangle$ is just $\{3\}$, while the set of instances with $shape = square$ is just $\{2\}$. Thus two NO nodes are created at the end of the corresponding branches.

The set of instances with $shape = circle$ is $\{1, 6\}$. It does not contain uniformly positive or negative instances so a new feature is selected on which to further split the instances. The only feature left now is *color*. Splitting the instances on this feature produces the final two leaves (a YES node and a NO node) and the algorithm terminates, having produced the following decision-tree:

```
          shape
            |
    ------------------
   |        |          |
triangle  square     circle
   |        |          |
   NO       NO       color
                       |
                  ---------
                  |         |
                 red      blue
                  |         |
                 YES        NO
```

5.3 Informaton-based heuristic for attribute selection

Clearly, we will want the algorithm to construct small, bushy trees, i.e. simple decision rules. However, the degree to which it will do so depends to an extent on how clever it is at selecting "good" attributes on which to split instances.

Selecting "good" attributes means giving priority to attributes which will best sort the instances out into uniform groups. So the question is, how can the algorithm be provided with a criterion, which will enable it distinguish (and select) this sort of attribute.

Several approaches have been explored. The most well-known is Quinlan's which involves calculating the *entropy* of the distribution of the positive/negative instances resulting from splitting on each of the remaining attributes and then using the attribute which achieves the lowest entropy distribution.

The entropy measure is based on the information theory of Shannon. According to this theory we can calculate the information content of the training set, and consequently, of any decision tree that covers this set of examples.

If we assume that all the instances in the training set E (the example above) occur with equal probability, then $p(YES) = 1/6$, $p(NO) = 5/6$. The information in E is:

$$I(E) = -\frac{1}{6}log_2(\frac{1}{6}) - \frac{5}{6}log_2(\frac{5}{6}) = 0.4308 + 0.2192 = 0.65$$

We want to find a measure of "goodness" (*information gain*) for each attribute chosen as a root of the current tree. This could be the total information in the tree minus the amount of information needed to complete the tree after choosing that attribute as a root. The amount of information needed to complete the tree is defined as the weighted average of the information in all its subtrees. The weighted average is computed by multiplying the information content of each subtree by the percentage of the examples present in that subtree and summing these products.

Assume that making attribute A, with n values, the root of the current tree, will partition the set of training examples E into subsets $E_1, ..., E_n$. Then, the information needed to complete that tree after making A the root is:

$$R(A) = \sum_{i=1}^{n} \frac{|E_i|}{|E|} I(E_i)$$

Then, the information gain of choosing attribute A is:

$$gain(A) = I(E) - R(A)$$

For the above example we can calculate the gain of choosing attributes *color* and *shape*. For *color* we have two subsets $C_1 = \{1, 2, 3\}$ and $C_2 = \{4, 5, 6\}$.

$$I(C_1) = -\frac{1}{3}log_2(\frac{1}{3}) - \frac{2}{3}log_2(\frac{2}{3}) = 0.5383 + 0.3840 = 0.9723$$

$$I(C_2) = -\frac{0}{3}log_2(\frac{0}{3}) - \frac{3}{3}log_2(\frac{3}{3}) = 0$$

$$gain(color) = I(E) - R(color) = 0.65 - (\frac{3}{6}0.9723 + \frac{3}{6}0) = 0.1638$$

For *shape* we have three subsets $S_1 = \{1, 6\}$, $S_2 = \{2, 5\}$ and $S_3 = \{3, 4\}$.

$$I(S_1) = -\frac{1}{2}log_2(\frac{1}{2}) - \frac{1}{2}log_2(\frac{1}{2}) = 1$$

Clearly $I(S_2) = 0$, and $I(S_3) = 0$, since they contain only one-class instances. Then

$$gain(shape) = I(E) - R(shape) = 0.65 - (\frac{2}{6}1 + 0 + 0) = 0.3166$$

Because *shape* provides grater information gain the algorithm will select it first to partition the tree (as it was shown in the examples above).

5.4 Learning multiple concepts

The algorithm described in Section 2 actually builds decision trees for classification of the training instances into two classes (YES – belonging to the concept, and NO – not belonging to the concept). This algorithm can be easily generalized to handle more than two classes (concepts) as follows:

1. If all instances in E belong to a single class, then create a node marked with the class name and halt. Otherwise, select an attribute A (e.g. using the information gain heuristic), with values $V_1...V_n$ and create the decision node

```
          A
          |
   -------------
  |   |  |     |
  |   |  |     |
 V1  V2  V3 ... Vn
```

2. Partition the training instances in E into subsets $E_1, E_2, ..., E_n$, according to the values of $A(\{V_1, V_2, ..., V_n\})$
3. Apply the algorithm recursively to each of the sets E_1, E_2 etc.

5.5 Learning from noisy data

In many situations the training data are imperfect. For example, the attribute or class values for some instances could be incorrect, because of errors. We call such data *noisy data*. In case of noise we usually abandon the requirement the hypothesis to cover all positive and none of the negative examples. So, we allow the learning system to misclassify some instances and we hope that the misclassified instances are those that contain errors.

Inducing decision trees from nosy data will cause basically two problems: first, the trees misclassify new data, and second, the trees tend to become very large and thus hard to understand and difficult to use.

Assume the following situation. At some step of the algorithm we have chosen an attribute A partitioning the current set S of 100 training instances into two classes – C_1 and C_2, where C_1 contains 99 instances and C_2 - one instance. Knowing that there is a noise in the training data, we can assume that all instances from S belong to class C_1. In this way we force the algorithm to stop further exploring the decision tree, i.e. we prune the subtree rooted at A. This technique is called *forward pruning*. There is another kind of pruning, called *post-pruning*, where first the whole tree is explored completely, and then the subtrees are estimated on their reliability with respect to possible errors. Then those of them with low estimates are pruned. Both techniques for pruning are based on probability estimates of the classification error in each node of the tree.