Induction of Decision Trees

Peter Waiganjo Wagacha

This notes are for ICS320 Foundations of Learning and Adaptive Systems

Institute of Computer Science
University of Nairobi
PO Box 30197, 00200 Nairobi.

9th May, 2003

Contents

1 Induction of Decision Trees

2 Introduction

Decision trees are powerful and popular tools for classification and prediction. Decision trees are attractive due to the fact that, in contrast to other machine learning techniques such as neural networks, they represent rules. Rules can readily be expressed so that humans can understand them or even directly used in a database access language like SQL so that records falling into a particular category may be retrieved.

In some applications, the accuracy of a classification or prediction is the only thing that matters. In such situations we do not necessarily care how or why the model works. In other situations, the ability to explain the reason for a decision, is crucial. In medical diagnosis tests are regularly performed. In many cases, there is often more than one test for a particular series of tests. Often one test may be more accurate than another. One test may also be more expensive than another. To the patient, all these may be irrelevant, but the doctor has to bear these intricacies in mind when interpreting the test results. In marketing one has to describe the customer segments to marketing professionals, so that they can utilize this knowledge in launching a successful marketing campaign. These domain experts must recognize and approve this discovered knowledge, and for this we need good descriptions. There are a variety of algorithms for building decision trees that share the desirable quality of interpretability. A well known and frequently used over the years is C4.5 developed by J. Ross Quinlan [Qui93] (or improved, but commercial version See5/C5.0).

3 What is a decision tree?

Decision tree is a classifier in the form of a tree structure (see Figure 1), where each node is either:

- a **leaf node** – indicates the value of the target attribute (class) of examples, or
- a **decision node** – specifies some test to be carried out on a single attribute-value, with one branch and sub-tree for each possible outcome of the test.

A decision tree can be used to classify an example by starting at the root of the tree and moving through it until a leaf node, which provides the classification of the instance.

Decision tree induction is a typical inductive approach to learn knowledge on classification. The key requirements to do mining with decision trees are:

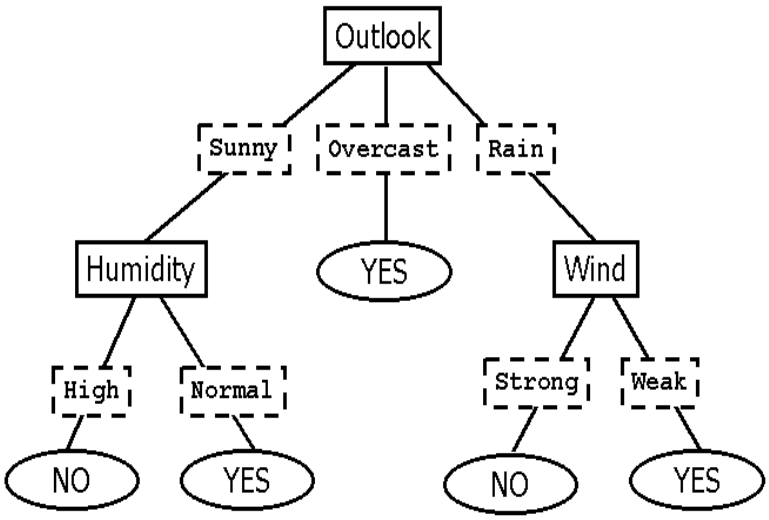

Figure 1: Decision tree: for conditions to play tennis.

- **Attribute-value description:** object or case must be expressible in terms of a fixed collection of properties or attributes. This means that we need to discretize continuous attributes, or this must have been provided in the algorithm.
- **Predefined classes (target attribute values):** The categories to which examples are to be assigned must have been established beforehand (supervised data).
- **Discrete classes:** A case does or does not belong to a particular class, and there must be more cases than classes.
- **Sufficient data:** Usually hundreds or even thousands of training cases.

4 Constructing decision trees

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees. Decision tree programs construct a decision tree T from a set of training cases.

J. Ross Quinlan originally developed ID3 at the University of Sydney. He first presented ID3 in 1975 in a book, Machine Learning, vol. 1, no. 1. ID3 is based on the Concept Learning System (CLS) algorithm.

Function ID3

```
Input:
    (R: a set of non-target attributes,
    C: the target attribute,
    S: a training set) returns a decision tree;

begin
  If S is empty, return a single node with value Failure;
  If S consists of records all with the same value for the
    target attribute, return a single leaf node with that value;
  If R is empty, then return a single node with the value of
    the most frequent of the values of the target attribute that are found
    in records of S; [in that case there may be be errors, examples
    that will be improperly classified];
  Let A be the attribute with largest
    Gain(A,S) among attributes in R;
  Let {a_j | j = 1,2,...,m} be the values of attribute A;
  Let {S_j | j = 1,2,...,m} be the subsets of S consisting
    respectively of records with value a_j for A;
  Return a tree with root labelled A and arcs labelled a_1, a_2, ..., a_m
    going respectively to the trees (ID3 (R-{A}, C, S_1),
    ID3 (R-{A}, C, S_2), ..., ID3 (R- {A}, C, S_m);
  Recursively apply ID3 to subsets S_j | j = 1,2,...,m until they are empty
end
```

Table 1: ID3 Decision Tree Algorithm

ID3 searches through the attributes of the training instances and extracts the attribute that best separates the given examples. If the attribute perfectly classifies the training sets then ID3 stops; otherwise it recursively operates on the m (where m = number of possible values of an attribute) partitioned subsets to get their "best" attribute. The algorithm uses a greedy search, that is, it picks the best attribute and never looks back to reconsider earlier choices. Note that ID3 may misclassify data.

The central focus of the decision tree growing algorithm is selecting which attribute to test at each node in the tree. For the selection of the attribute with the most inhomogeneous class distribution the algorithm uses the concept of *entropy*, which is explained next.

5 Which attribute is the best classifier?

The estimation criterion in the decision tree algorithm is the selection of an attribute to test at each decision node in the tree. The goal is to select the attribute that is most useful for classifying examples. A good quantitative measure of the worth of an attribute is a statistical property called *information gain* that measures how well a given attribute separates the training examples according to their target classification. This measure is used to select among the candidate attributes at each step while growing the tree.

6 Entropy

In order to define information gain precisely, we need to define a measure commonly used in information theory, called entropy, that characterizes the (im)purity of an arbitrary collection of examples. Given a set S, containing only positive and negative examples of some target concept (for a 2 class problem), the entropy of set S relative to this simple, binary classification is defined as:

$$Entropy(S) = -p_p log_2 p_p - p_n log_2 p_n \tag{1}$$

where p_p is the proportion of positive examples in S and p_n is the proportion of negative examples in S. In all calculations involving entropy we define $0log_0$ to be 0.

To illustrate, suppose S is a collection of 25 examples, including 15 positive and 10 negative examples [15+, 10-]. Then the entropy of S relative to this classification is

$$Entropy(S) = -(\frac{15}{25})log_2(\frac{15}{25}) - (\frac{10}{25})log_2(\frac{10}{25}) = 0.970 \tag{2}$$

Notice that the entropy is 0 if all members of S belong to the same class. For example, if all members are positive ($p_p = 1$), then p_n is 0, and Entropy(S) = -1$*$log$_2$(1) - 0$*$log$_2$0 = -1$*$0 - 0$*$log$_2$0 = 0. Note the entropy is 1 (at its maximum!) when the collection contains an equal number of positive and negative examples. If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1. Figure 2 shows the form of the entropy function relative to a binary classification, as p_+ varies between 0 and 1.

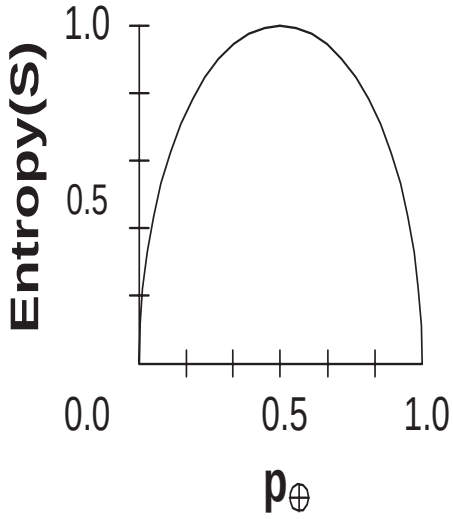

Figure 2: The entropy function relative to a binary classification, as the proportion of positive examples p_p varies between 0 and 1.

One interpretation of entropy from information theory is that it specifies the minimum number of bits of information needed to encode the classification of an arbitrary member of S (i.e., a member of S drawn at random with uniform probability). For example, if p$_p$ is 1, the receiver knows the drawn example will be positive, so no message need be sent, and the entropy is 0. On the other hand, if p$_p$ is 0.5, one bit is required to indicate whether the drawn example is positive or negative. If p_p is 0.8, then a collection of messages can be encoded using on average less than 1 bit per message by assigning shorter codes to collections of positive examples and longer codes to less likely negative examples.

Thus far we have discussed entropy in the special case where the target classification is binary. If the target attribute takes on c different values, then the entropy of S relative to this c-wise classification is defined as

$$Entropy(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{3}$$

where p_i is the proportion of S belonging to class i. Note the logarithm is still base 2 because entropy is a measure of the expected encoding length measured in bits. Note also that if the target attribute can take on c possible values, the maximum possible entropy is $log_2 c$.

7 Information gain

Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data. The measure we will use, called information gain, is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. More precisely, the information gain, $Gain(S, A)$ of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) = Entropy(S) - \sum_{v \in V(A)} Entropy \frac{\mid S_v \mid}{\mid S \mid}(S_v) \tag{4}$$

where $Values(A)$ is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v (i.e., $S_v = \{s \ \ S \mid A(s) = v\}$). Note the first term in the equation for $Gain$ is just the entropy of the original collection S and the second term is the expected value of the entropy after S is partitioned using attribute A. The expected entropy described by this second term is simply the sum of the entropies of each subset S_v, weighted by the fraction of examples $\mid S_v \mid / \mid S \mid$ that belong to S_v. $Gain(S, A)$ is therefore the expected reduction in entropy caused by knowing the value of attribute A. Put another way, $Gain(S, A)$ is the information provided about the target attribute value, given the value of some other attribute A. The value of $Gain(S, A)$ is the number of bits saved when encoding the target value of an arbitrary member of S, by knowing the value of attribute A.

The process of selecting a new attribute and partitioning the training examples is now repeated for each non-terminal descendant node, this time using only the training examples associated with that node. Attributes that have been incorporated higher in the tree are excluded, so that any given attribute can appear at most once along any path through the tree. This process continues for each new leaf node until either of two conditions is met:

1. every attribute has already been included along this path through the tree, or
2. the training examples associated with this leaf node all have the same target attribute value (i.e., their entropy is zero).

8 Incorporating Continuous-valued attributes

The initial definition of ID3 is restricted to attributes that take on a discrete set of values. First, the target attribute whose value is predicted by the learned tree must be discrete valued. Second, the attributes tested in the decision nodes of the tree must also be discrete valued. This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree. This can be accomplished by dynamically defining new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals. In particular, for an attribute A that is continuous-valued, the algorithm can dynamically create a new Boolean attribute A_c that is true if $A < c$ and false otherwise. The only question is how to select the best value for the threshold c. Clearly, we would like to pick a threshold, c, that produces the greatest information gain. By sorting the examples according to the continuous attribute A , then identifying adjacent examples that differ in their target classification, we can generate a set of candidate thresholds midway between the corresponding values of A. It can be shown that the value of c that maximizes information gain must always lie at such a boundary. These candidate thresholds can then be evaluated by computing the information gain associated with each. The information gain can then be computed for each of the candidate attributes, and the best can be selected. This dynamically created Boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

9 How Decision trees partition the space

There are a couple of ways through which this is done. These methods revolve around how to select attributes, which in turn dictates how the decision tree is grown.

10 Alternative measures for selecting attributes

There is a natural bias in the information gain measure that favours attribute with many values over those with few values. As an extreme example, consider an attribute that perfectly predicts the target function over the training data, that is; the test at the root creates as many links to a leaf as there are training examples. Thus, each training example is represented by a single leaf, after effecting the test at the root node of the tree. An example of such an attribute can be the (unique) timestamp an event occurred.

The problem with this decision tree is that it will be a poor predictor on subsequent examples, despite the fact that it has a very high information gain over the training data.

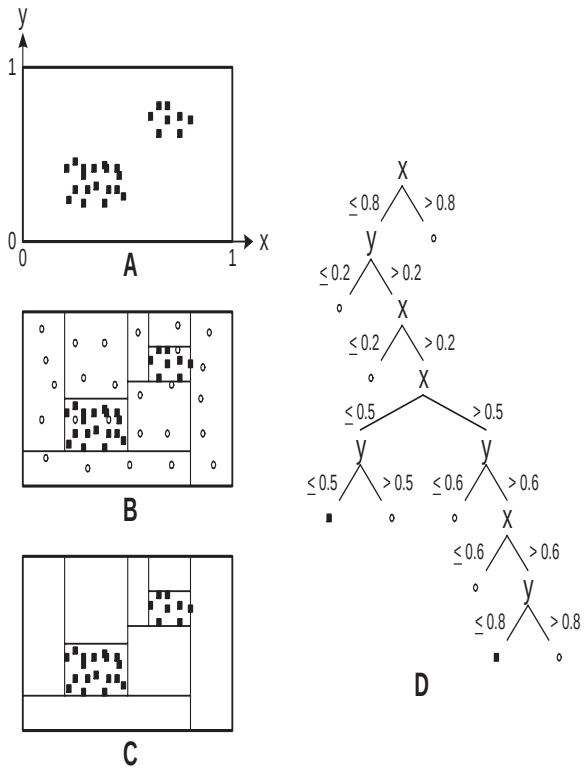

Figure 3: Decision tree partition

One way to avoid this difficulty is to select attributes based on other measure than information gain [MKS94].

10.1 Gain ratio

The gain ratio measure penalizes attributes such as timestamp by incorporating a term, called split information, that is sensitive to how broadly and uniformly the attribute splits the data:

$$SplitInformation(S, A) = -\sum_{i=1}^{c} \frac{S_i}{S} \log_2 \frac{S_i}{S} \tag{5}$$

where S_1 through S_c are the c subsets of examples resulting from partitioning S by the c-valued attribute A. Note that $SplitInformation$ is actually the entropy of S with respect to the values of attribute A. This is in contrast to the previous use of the entropy, in which the entropy of S was only considered with respect to the target attribute whose value is to be predicted by the learned tree.

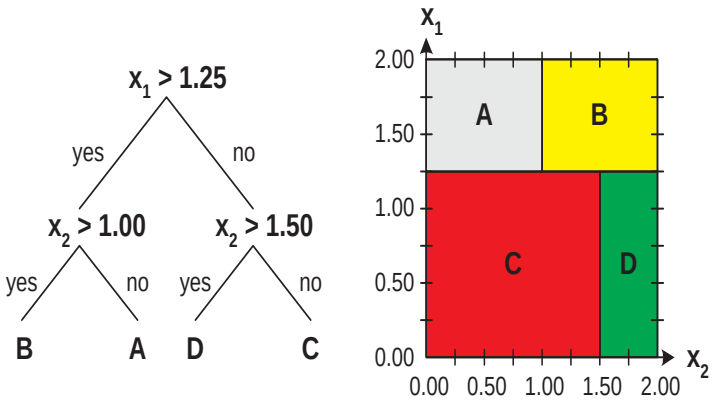

Figure 4: The left side of the figure shows a simple axis-parallel tree that uses two attributes. The right side shows the partitioning that this tree creates in the attribute space.

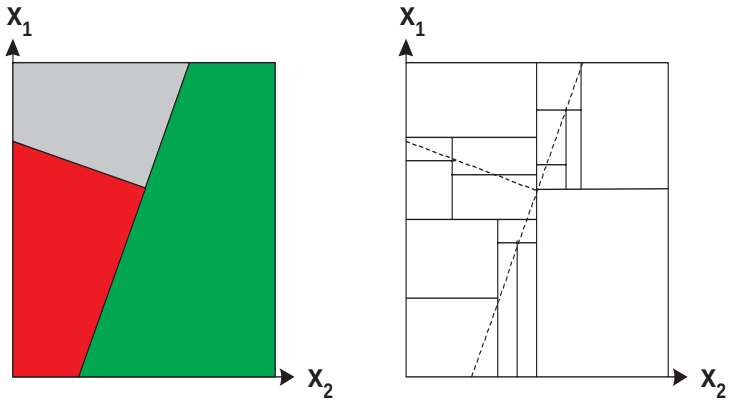

Figure 5: The left side shows a simple 2-D domain in which two oblique hyperplanes define the classes. The right side shows an approximation of the sort that an axis-parallel decision tree would have to create to model this domain.

The $GainRation$ measure is defined in terms of the earlier $Gain$ measure, as well as this $SplitInformation$, as follows

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)} \tag{6}$$

Notice that the $SplitInformation$ term discourages the selection of attributes with many uniformly distributed values. For example, consider a collection of n examples that are completely separated by attribute A. In this case, the $SplitInformation$ value will be $\log_2 n$. In contrast, a boolean attribute B that splits the same n examples in half will have $SplitInformation$ of 1. If attributes A and B produce the same information gain, then clearly B will score higher according to the $GainRatio$ measure.

10.2 Gini index

In each of the following methods, the set of examples S at the node to be split contains n instances that belong to one of k categories. Assume the example set S is split into two non-overlapping subsets S_L and S_R. L_j and R_j are the number of instances of category j in S_L and S_R respectively.

The $GiniIndex$ was proposed for decision trees by Breiman. The Gini Index as originally defined, measures the probability of mis-classification of a set of instances, rather than the impurity of a split. Therefore, a little variation of the original equation is provided:

$$GiniL = 1 - \sum_{i=1}^{k} (\frac{L_i}{|S_L|})^2 \tag{7}$$

$$GiniR = 1 - \sum_{i=1}^{k} (\frac{R_i}{|S_R|})^2 \tag{8}$$

$$Impurity = \frac{|S_L| \times GiniL + |S_R| \times GiniR}{n} \tag{9}$$

where $GiniL$ is the $Gini$ Index for the subset S_L and $GiniR$ is that for the subset S_R.

Twoing Value The Twoing value was first proposed by Breiman. This value is computed as follows

$$TwoingValue = \frac{|S_L|}{n} \times \frac{|S_R|}{n} \times (\sum_{i=1}^{k} |\frac{L_i}{|S_L|} - \frac{R_i}{|S_R|}|)^2 \tag{10}$$

The $TwoingValue$ is actually a goodness measure rather than an impurity measure. Therefore, the reciprocal of this value is often used as an indication of the impurity.

11 Strengths and Weaknesses of Tree methods

Below we have enumerated the strengths and weaknesses of decision trees methods.

11.1 Strengths

The strengths of decision tree methods are:

- Decision trees are able to generate understandable rules.
- Decision trees perform classification without requiring much computation.
- Decision trees are able to handle both continuous and categorical variables.
- Decision trees provide a clear indication of which fields are most important for prediction or classification.

11.2 Weaknesses

The weaknesses of decision tree methods are:

- Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
- Decision trees are prone to errors in classification problems with many classes and a relatively small number of training examples.
- Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.
- Decision trees do not treat non-rectangular classification regions well. Most decision-tree algorithms only examine a single attribute (feature) at a time. This leads to rectangular classification boxes that may not correspond well with the actual distribution of records in the decision space.

References

[KBM96] Miroslav Kubat, Ivan Bratko, and Ryszard Michalski. A review of machine learning methods. *Machine Laearning and Data Mining: Methods and Applications*, 1996. Editors: R.S. Michalski and I. Bratko and M. Kubat.

[MKS94] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo CA, 1993.

[RN95] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995. available at http://www.aima.cs.berkeley.edu.