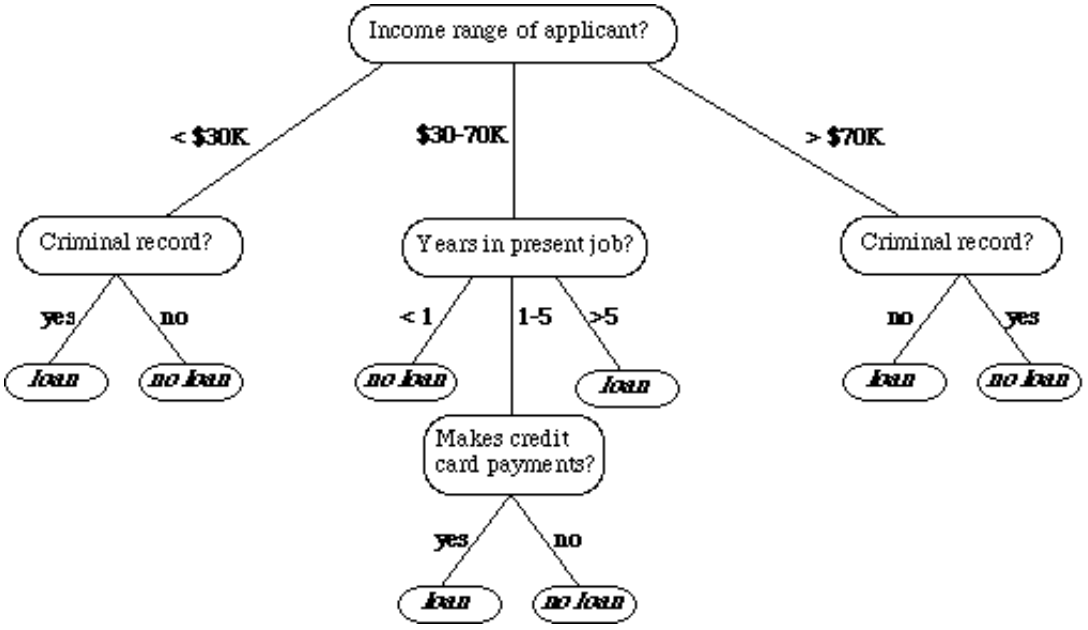
Induction of Decision Trees

Reference: Bratko sections 18.5, 18.6

Aim:
To describe an algorithm whose input is a collection of instances and their correct classification and whose output is a decision tree that can be used to classify each instance.
Keywords: attributes, backed-up error estimate, C4.5, C5, concept learning system (CLS), decision trees, entropy, expected error estimate, feature, ID3, instances, Laplace error estimate, pruning decision trees, splitting criterion in ID3, tree induction algorithm, windowing in ID3
Plan:
• What is a decision tree? • Building a decision tree • Which attribute should we split on? • Information theory and the splitting criterion • ID3 example & ID3 symbolic version • ID3 in iProlog • ID3 enhancements: windowing • Dealing with noisy data - expected error pruning

Decision Trees

- A decision tree is a tree in which each branch node represents a choice between a number of alternatives, and each leaf node represents a classification or *decision*.
- For example, we might have a decision tree to help a financial institution decide whether a person should be offered a loan:



- We wish to be able to induce a decision tree from a set of data about instances together with the decisions or classifications for those instances.

Example Instance Data

```
size:   small medium large
colour: red blue green
shape:  brick wedge sphere pillar

%% yes

medium  blue    brick
small   red     sphere
large   green   pillar
large   green   sphere

%% no

small   red     wedge
large   red     wedge
large   red     pillar
```

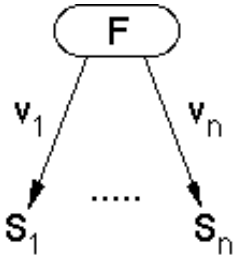
- In this example, there are 7 *instances*, described in terms of three *features* or *attributes* (size, colour, and shape), and the instances are classified into two *classes* `%% yes` and `%% no`.
- We shall now describe an algorithm for inducing a decision tree from such a collection of classified instances.
- Originally termed CLS (Concept Learning System) it has been successively enhanced.
- At the highest level of enhancement that we shall describe in these notes, the system is known as **ID3** - later versions include C4, C4.5 and See5/C5.0 (latest version release 1.20, May 2004).

Originator of the ID3 Algorithm

- ID3 and its successors have been developed by Ross Quinlan, who discovered it while working with Earl Hunt in the 1970s. He subsequently worked at Sydney Uni, Rand Corporation in California, UTS, back to Sydney Uni, and several years at UNSW. He now runs his own company, Rulequest (www.rulequest.com).
- You can find out more about ID3 and its later development in C4.5 in his book: Quinlan, J. R. *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, 1993, or via links at http://www.rulequest.com/Personal/

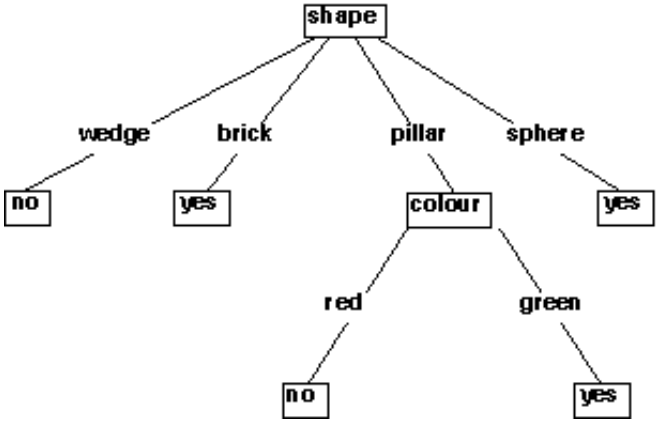
Photo of Ross Quinlan from his personal webpage at www.rulequest.com

Tree Induction Algorithm



- The algorithm operates over a set of training instances, *C*.
- If all instances in *C* are in class *P*, create a node *P* and stop, otherwise select a *feature* or *attribute* **F** and create a decision node.
- Partition the training instances in *C* into subsets according to the values of *V*.
- Apply the algorithm recursively to each of the subsets *C*.

Output of Tree Induction Algorithm



This can easily be expressed as a nested if-statement

```
if (shape == wedge)
        return no;
if (shape == brick)
        return yes;
if (shape == pillar)
{
        if (colour == red)
                return no;
        if (colour == green)
                return yes;
}
if (shape == sphere)
        return yes;
```

Choosing Attributes and ID3

- The order in which attributes are chosen determines how complicated the tree is.
- ID3 uses information theory to determine the most informative attribute.

- A measure of the information content of a message is the inverse of the probability of receiving the message:

$$information1(M) = 1/probability(M)$$

- Taking logs (base 2) makes information correspond to the number of bits required to encode a message:

$$information(M) = -log_2(probability(M))$$

Information

- The information content of a message should be related to the degree of surprise in receiving the message.
- Messages with a high probability of arrival are not as informative as messages with low probability.
- Learning aims to predict accurately, i.e. reduce surprise.
- Probabilities are multiplied to get the probability of two or more things both/all happening. Taking logarithms of the probabilities allows information to be added instead of multiplied.

Entropy

- Different messages have different probabilities of arrival.
- Overall level of uncertainty (termed entropy) is:

$$-\Sigma_i P_i \log_2 P_i$$

- Frequency can be used as a probability estimate.
- E.g. if there are 5 positive examples and 3 negative examples in a node the estimated probability of positive is 5/8 = 0.625.

Information and Learning

- We can think of learning as building many-to-one mappings between input and output.
- Learning tries to reduce the information content of the inputs by mapping them to fewer outputs.
- Hence we try to minimise entropy.
- The simplest mapping is to map everything to one output.
- We seek a trade-off between accuracy and simplicity.

Splitting Criterion

- Work out entropy based on distribution of classes.
- Trying splitting on each attribute.
- Work out expected information gain for each attribute.
- Choose best attribute.

Example

- Initial decision tree is one node with all examples.
- There are 4 positive examples and 3 negative examples
- i.e. probability of positive is 4/7 = 0.57; probability of negative is 3/7 = 0.43
- Entropy is: – (0.57 * log 0.57) – (0.43 * log 0.43) = 0.99
- Evaluate possible ways of splitting.

Example Part 2

- Try split on *size* which has three values: *large, medium* and *small.*
- There are four instances with size = large.
- There are two large positives examples and two large negative examples.
- The probability of positive is 0.5
- The entropy is: – (0.5 * log 0.5) – (0.5 * log 0.5) = 1

Example Part 3

- There is one small positive and one small negative
- Entropy is: – (0.5 * log 0.5) – (0.5 * log 0.5) = 1
- There is only one medium positive and no medium negatives, so entropy is 0.
- Expected information for a split on size is:

$$\left(1 \times \frac{4}{7}\right) + \left(1 \times \frac{2}{7}\right) + \left(0 \times \frac{1}{7}\right) = 0.86$$

Example Part 4

- The expected information gain is: 0.99 – 0.86 = 0.13
- Now try splitting on colour and shape.
- Colour has an information gain of 0.52
- Shape has an information gain of 0.7
- Therefore split on shape.
- Repeat for all subtree

Summary of Splitting Criterion

Some people learn best from an example; others like to see the most general formulation of an algorithm. If you are an "examples" person, don't let the following subscript-studded presentation panic you.

Assume there are k classes C_1, ... C_k ($k = 2$ in our example).

to decide which attribute to split on:

- **for** each attribute that has not already been used
  - Calculate the information gain that results from splitting on that attribute
  - Split on the attribute that gives the greatest information gain.

Summary of Splitting Criterion 2

to calculate the information gain from splitting N instances on attribute A:

- Calculate the entropy E of the current set of instances.
- **for** each value a_j of the attribute $A (j = 1, ..., r)$
  - Suppose that there are
    $J_{j,1}$ instances in class C_1,
    ...,
    $J_{j,k}$ instances in class C_k,
    for a total of J_j instances with $A = a_j$.
  - Let $q_{j,1} = J_{j,1}/J_j$, ..., $q_{j,k} = J_{j,k}/J_j$;
  - The entropy E_j associated with $A = a_j$ is
    $-q_{j,1} \log_2(q_{j,1}) ... -q_{j,k} \log_2(q_{j,k})$
- Now compute $E – (J_1/N)E_1 ... –(J_r/N)E_r$ - this is the information gain associated with a split on attribute A.

Summary of Splitting Criterion 3

to calculate the entropy E of the current set of instances

- Suppose that of the N instances classified to this node, I_1 belong to class C_1, ..., I_k belong to class C_k,
- Let $p_1 = I_1/N$, ..., $p_k = I_k/N$,
- Then the initial entropy E is $-p_1 \log_2(p_1) - p_2 \log_2(p_2) \ldots - p_k \log_2(p_k)$.

Using the iProlog Implementation of ID3

% cat example.data

```
table object(
                texture(smooth, wavy, rough),
                temperature(cold, cool, warm, hot),
                size(small, medium, large),
                class(yes, no)
)!

object(smooth, cold, large, yes).
object(smooth, cold, small, no).
object(smooth, cool, large, yes).
object(smooth, cool, small, yes).
object(smooth, hot, small, yes).
object(wavy, cold, medium, no).
object(wavy, hot, large, yes).
object(rough, cold, large, no).
object(rough, cool, large, yes).
object(rough, hot, small, no).
object(rough, warm, medium, yes).
```

Using the iProlog Implementation of ID3 - ctd

% prolog example.data

```
iProlog ML (21 March 2003)
: id(object)?
id0
: pp id0!

object(Texture, Temperature, Size, Class) :-
   (Temperature = cold ->
       (Texture = smooth ->
           (Size = small -> Class = no
           | Size = large -> Class = yes)
       | Texture = wavy -> Class = no
       | Texture = rough -> Class = no)
   | Temperature = cool -> Class = yes
   | Temperature = warm -> Class = yes
   | Temperature = hot ->
       (Texture = smooth -> Class = yes
       | Texture = wavy -> Class = yes
       | Texture = rough -> Class = no)).

:
```

Windowing

- ID3 can deal with very large data sets by performing induction on subsets or *windows* onto the data.
    1. Select a random subset of the whole set of training instances.
    2. Use the induction algorithm to form a rule to explain the current window.
    3. Scan through all of the training instances looking for exceptions to the rule.
    4. Add the exceptions to the window
- Repeat steps 2 to 4 until there are no exceptions left.

Noisy Data

- Frequently, training data contains "noise" - i.e. examples which are misclassified, or where one or more of the attribute values is wrong.
- In such cases, one is like to end up with a part of the decision tree which considers say 100 examples, of which 99 are in class C_1 and the other is apparently in class C_2 (because it is misclassified).
- If there are any unused attributes, we *might* be able to use them to elaborate the tree to take care of this one case, but the subtree we would be building would in fact be wrong, and would likely misclassify real data.
- Thus, particularly if we know there is noise in the training data, it may be wise to "prune" the decision tree to remove nodes which, statistically speaking, seem likely to arise from noise in the training data.
- A question to consider: *How fiercely should we prune?*

Expected Error Pruning

- Approximate expected error assuming that we prune at a particular node.
- Approximate backed-up error from children assuming we did not prune.
- If expected error is less than backed-up error, prune.

(Static) Expected Error

- If we prune a node, it becomes a leaf labelled, C.
- What will be the expected classification error at this leaf?

$$E(S) = \frac{N - n + k - 1}{N + k}$$

(This is called the *Laplace* error estimate - it is based on the assumption that the distribution of probabilities

that examples will belong to different classes is uniform.)

S is the set of examples in a node
k is the number of classes
N examples in S
C is the majority class in S
n out of N examples in S belong to C

Backed-up Error

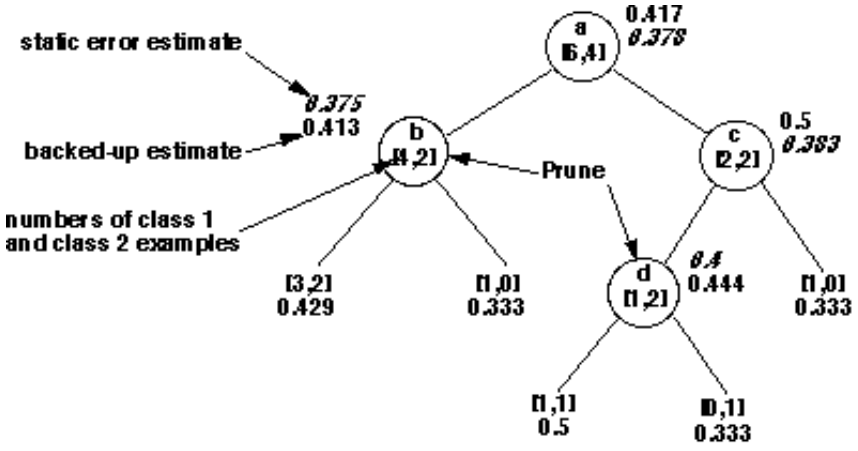
- For a non-leaf node
- Let children of *Node* be *Node1*, *Node2*, etc

$$BackedUpError(Node) = \Sigma_i P_i \times Error(Node_i)$$

- Probabilities can be estimated by relative frequencies of attribute values in sets of examples that fall into child nodes.

$$Error(Node) = \min(E(Node), BackedUpError(Node))$$

Pruning Example



Error Calculation for Pruning Example

- **Left child of *b* has class frequencies [3, 2]**

$$E = \frac{N - n + k - 1}{N + k} = \frac{5 - 3 + 2 - 1}{5 + 2} = 0.429$$

- **Right child has error of 0.333, calculated in the same way**

- **Static error estimate *E(b)* is 0.375, again calculated using the Laplace error estimate formula, with *N*=6, *n*=4, and *k*=2.**

- **Backed-up error is:**

$$BackedUpError(b) = (5/6)\times 0.429 + (1/6)\times 0.333 = 0.413$$

(5/6 and 1/6 because there are 4+2=6 examples handled by node *b*, of which 3+2=5 go to the left subtree and 1 to the right subtree.

- **Since backed-up estimate of 0.413 is greater than static estimate of 0.375, we prune the tree and use the static error of 0.375**

Summary: Induction of Decision Trees

- The ID3 family of decision tree induction algorithms use information theory to decide which attribute shared by a collection of instances to split the data on next.
- Attributes are chosen repeatedly in this way until a complete decision tree that classifies every input is obtained. If the data is noisy, some of the original instances may be misclassified.
- It may be possible to prune the decision tree in order to reduce classification errors in the presence of noisy data.
- The speed of this learning algorithm is reasonably high, as is the speed of the resulting decision tree classification system.
- Generalisation ability can be reasonable too.

Bill Wilson's contact info
Last modified 11 June 2008
UNSW's CRICOS Provider No. is 00098G