Decision Trees

Alex Sverdlov*

1 Introduction

A decision tree is a graphical representation of decision making that is easily interpretable by humans. Starting from the root, each node applies a test, deciding which of the child branches to take for subsequent tests. This continues recursively until a leaf node is reached—the label on the leaf node is the output.

In this scenario, the training (and subsequent input) data are often represented as a set of tuples (each tuple being a list of attribute values). Given such a tuple, the task is to predict the target label.

Learning decision trees is done by accepting a set of labeled tuples, and recursively deciding what test (on which attribute) to apply at each node. The goal is to induce a decision tree that has good accuracy on the training set, and generalizes well on verification set.

A related view of decision trees is to consider the volumes decision rules create—the tree recursively partitions the problem space into disjoint subspaces—with each subspace representing a particular label.

We can also view a decision tree as a collection of tests. Every path from the root node

*alex@theparticle.com

to the leaf represents a list of tests that need to be applied to classify an example with the leaf label. With this view (treating the tree as a bag of tests as opposed to a hierarchical tree), it is easier to spot undesirable tests and remove them.

In this paper, we review some key decision tree concepts.

2 Background

Most decision tree learning algorithms are variations on the top-down greedy search algorithm, with the most notable example being ID3 (Interactive Dichotomizer 3) by Quinlan [2]. Quinlan references Hunt's Concept Learning System (CLS) [1] as inspiration and a precursor to ID3.

Hunt's Concept Learning System was a divide and conquer scheme that could handle binary (positive and negative) target values, with the decision attribute being decided by a heuristic based on the largest number of positive cases.

ID3 improves on Hunt's approach by using an information theoretic measure of *information gain* to decide on the test attribute, and allowing for multi-valued target labels.

C4.5, also proposed by Quinlan [3], improves on ID3 with ability to handle numeric attributes and dealing with missing values. C4.5 also introduced a rule pruning mechanism—to avoid over-fitting the tree to the training data.

3 ID3

Starting with a set of tuples, ID3 algorithm calculates *information gain* on every attribute, and uses the highest information gain attribute to split the dataset. The algorithm then precedes recursively on each resulting tuple set.

Information gain measures the reduction in entropy of the target attribute minus the

entropy of the target attribute after the split on A:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where *Entropy* is a measure of bits it takes represent the target attribute and S_v is the set of tuples resulting from the split on value v of attribute A. The $|S_v|/|S|$ scales the split entropy by the number of elements in the set after the split on v.

$$Entropy(X) = -\sum_{x \in \mathcal{A}_x} P(x) \log_2 P(x)$$

The major limitation of ID3 is the inability to calculate information gain for a numeric attribute. In general, numerical attributes, especially those that function as keys, will have very high information gain, without any predictive power.

Running ID3 to conclusion (recursively until the entire dataset is exhausted), results in over fitting the training data. This can be avoided by stopping the algorithm early or later pruning branches.

3.1 C4.5

The ID3 method is not without problems, and C4.5 is essentially a set of adjustments to the basic ID3 algorithm to make it work better. For one, the *Gain* has a tendency of favoring unique identifiers. If we apply *Gain* on a database table, it will pick out all the keys, dates, ids, etc—none of which generalize.

When calculating which value to split on, C4.5 takes the number of distinct values into consideration. If a node will branch out a million different children, then we generally do

not want to use that attribute.

$$Split(S, A) = -\sum_{i=1}^{c} \frac{|S_v|}{|S|} \log_2 \frac{|S_i|}{|S|} \qquad GainRatio(S, A) = \frac{Gain(S, A)}{Split(S, A)}$$

Here, $S_1, ..., S_c$ are c subsets of examples resulting from partitioning S by c-valued attribute A.

C4.5 introduces a way of dealing with numerical values. If an attribute A is numeric, it has N distinct numeric values (the datasets are finite). Such an attribute presents $N - 1$ potential splits (we can split that attribute at any of the $N - 1$ values.

To efficiently calculate the information gain for each of the $N-1$ split points of a numerical attribute we need to sort the dataset on values of the attribute. Once the numeric attribute is sorted, it is feasible to calculate information gain using a single iteration over the data.

Missing values are addressed by calculating the ratio of non-missing values within the split, and weighing the tuple with the missing values according to the ratio of the non-missing values. For example, if a node is to be split in two, and has 2 negative values, and 3 positive values, and two missing values, then the two missing values will be weighted as 0.4 negative, and 0.6 positive.

Another major improvement C4.5 brings is pruning. We start out over-fitting the tree using ID3 algorithm, and convert the resulting tree into a bag of rules (each path from root to leaf becomes a conjunction). For each such conjunction, remove attributes if such a removal does not hurt the rule's classification performance (using validation tests). Sort the rules by their *estimated accuracy* (estimated by applying rule to either training samples or a different verification set), and apply them in order until a rule fits, and we are able to classify. This scheme avoids the dangers of over fitting and under fitting.

References

[1] Earl B Hunt. *Concept learning: An information processing problem.* Wiley, 1966.

[2] J. Ross Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[3] J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.