Chapter 3 — Decision Tree Learning — Part 2 — Issues in decision tree learning

Pralhad Teggi · Follow

16 min read · Feb 15, 2020

Practical issues in learning decision trees include

- determining how deeply to grow the decision tree,
- handling continuous attributes,
- choosing an appropriate attribute selection measure,
- handling training data with missing attribute values,
- handling attributes with differing costs, and
- improving computational efficiency.

Below we discuss each of these issues and extensions to the basic ID3 algorithm that address them. ID3 has itself been extended to address most of these issues, with the resulting system renamed C4.5.

1. Avoiding Overfitting the Data

When we are designing a machine learning model, a model is said to be a good machine learning model, if it generalizes any new input data from the problem domain in a proper way. This helps us to make predictions in the future data, that data model has never seen. Now, suppose we want to check how well our machine learning model learns and generalizes to the new data. For that we have overfitting and underfitting, which are majorly responsible for the poor performances of the machine learning algorithms.

Underfitting

A machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data. Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data. In such cases the rules of the machine learning model are too easy and flexible to be applied on such a minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and also reducing the features by feature selection.

Overfitting

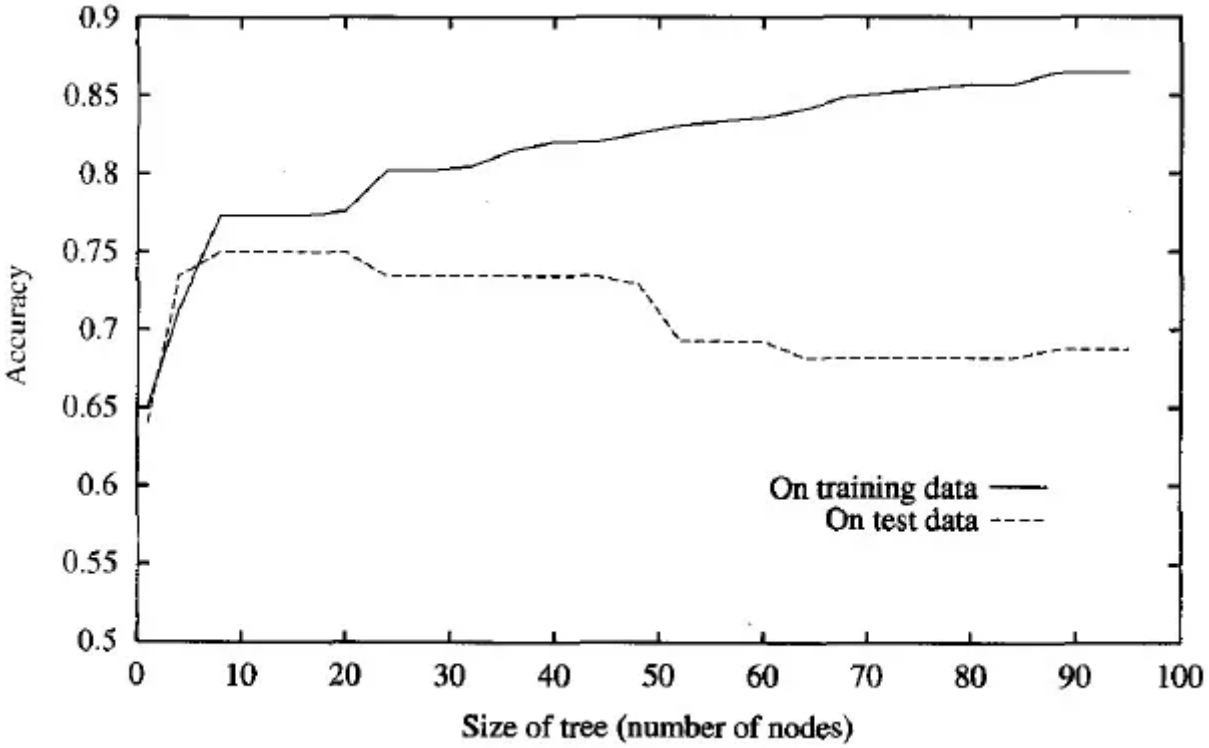
A machine learning algorithm is said to be overfitted, when we train it with a lot of data. When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set. Then the model does not categorize the data correctly, because of too much of details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

Coming to our ID3 algorithm, it grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that overfit the training examples.

Definition — Overfit: Given a hypothesis space H, a hypothesis h ∈ H is said to overfit the training data if there exists some alternative hypothesis h' ∈ H, such that h has

smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

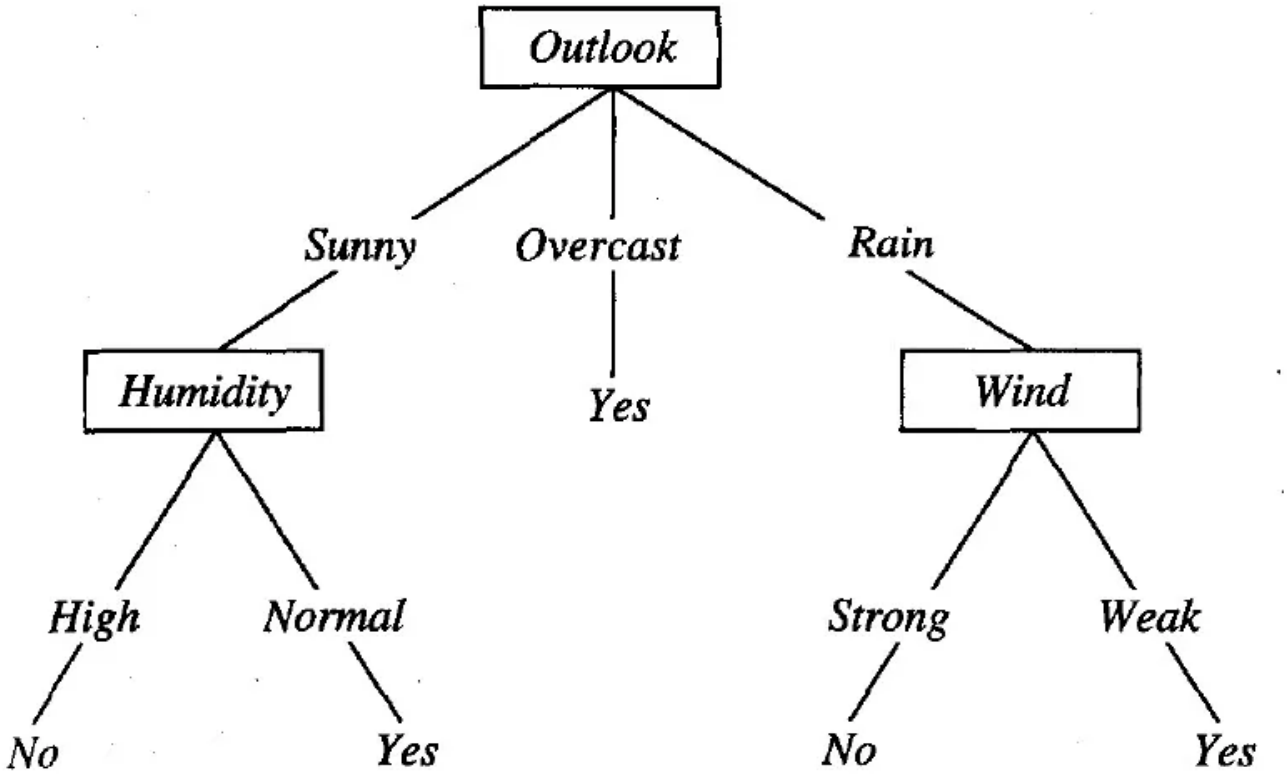
The below figure illustrates the impact of overfitting in a typical application of decision tree learning. In this case, the ID3 algorithm is applied to the task of learning which medical patients have a form of diabetes.



The horizontal axis of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed.
The vertical axis indicates the accuracy of predictions made by the tree.
The solid line shows the accuracy of the decision tree over the training examples, whereas the broken line shows accuracy measured over an independent set of test examples (not included in the training set).

Lets try to understand the effect of adding the following positive training example, incorrectly labeled as negative, to the training examples Table.

<Sunny, Hot, Normal, Strong, ->, Example is noisy because the correct label is +.
Given the original error-free data, ID3 produces the decision tree shown in Figure.



However, the addition of this incorrect example will now cause ID3 to construct a more complex tree. In particular, the new example will be sorted into the second leaf node from the left in the learned tree of above Figure, along with the previous positive examples D9 and D11.
Because the new example is labeled as a negative example, ID3 will search for further refinements to the tree below this node. The result is that ID3 will output a decision tree (h) that is more complex than the original tree from above figure (h'). Of course, h will fit the collection of training examples perfectly, whereas the simpler h' will not. However, given that the new decision node is simply a consequence of fitting the noisy training example, we expect h to outperform h' over subsequent data drawn from the same instance distribution.
The above example illustrates how random noise in the training examples can lead to overfitting.

In fact, **overfitting is possible even when the training data are noise-free,** especially when small numbers of examples are associated with leaf nodes. In this case, it is quite possible for coincidental regularities to occur, in which some attribute happens to partition the examples very well, despite being unrelated to the actual target function. Whenever such coincidental regularities exist, there is a risk of overfitting.

Avoiding Overfitting —

There are several approaches to avoiding overfitting in decision tree learning. These can be grouped into two classes:
- **Pre-pruning (avoidance):** Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- **Post-pruning (recovery):** Allow the tree to overfit the data, and then post-prune the tree

Although the first of these approaches might seem more direct, the second approach of post-pruning overfit trees has been found to be more successful in practice. This is due to the difficulty in the first approach of estimating precisely when to stop growing the tree. Regardless of whether the correct tree size is found by stopping early or by post-pruning, a key question is what criterion is to be used to determine the correct final tree size.

Criterion used to determine the correct final tree size

- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree
- Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set
- Use measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is called the Minimum Description Length
- MDL — Minimize : size(tree) + size (misclassifications(tree))

The first of the above approaches is the most common and is often referred to as a training and validation set approach. We discuss the two main variants of this approach below. In this approach, the available data are separated into two sets of examples: a training set, which is used to form the learned hypothesis, and a separate validation set, which is used to evaluate the accuracy of this hypothesis over subsequent data and, in particular, to evaluate the impact of pruning this hypothesis. The motivation is this: Even though the learner may be misled by random errors and coincidental regularities within the training set, the validation set is unlikely to exhibit the same random fluctuations. Therefore, the validation set can be expected to provide a safety check against overfitting the spurious

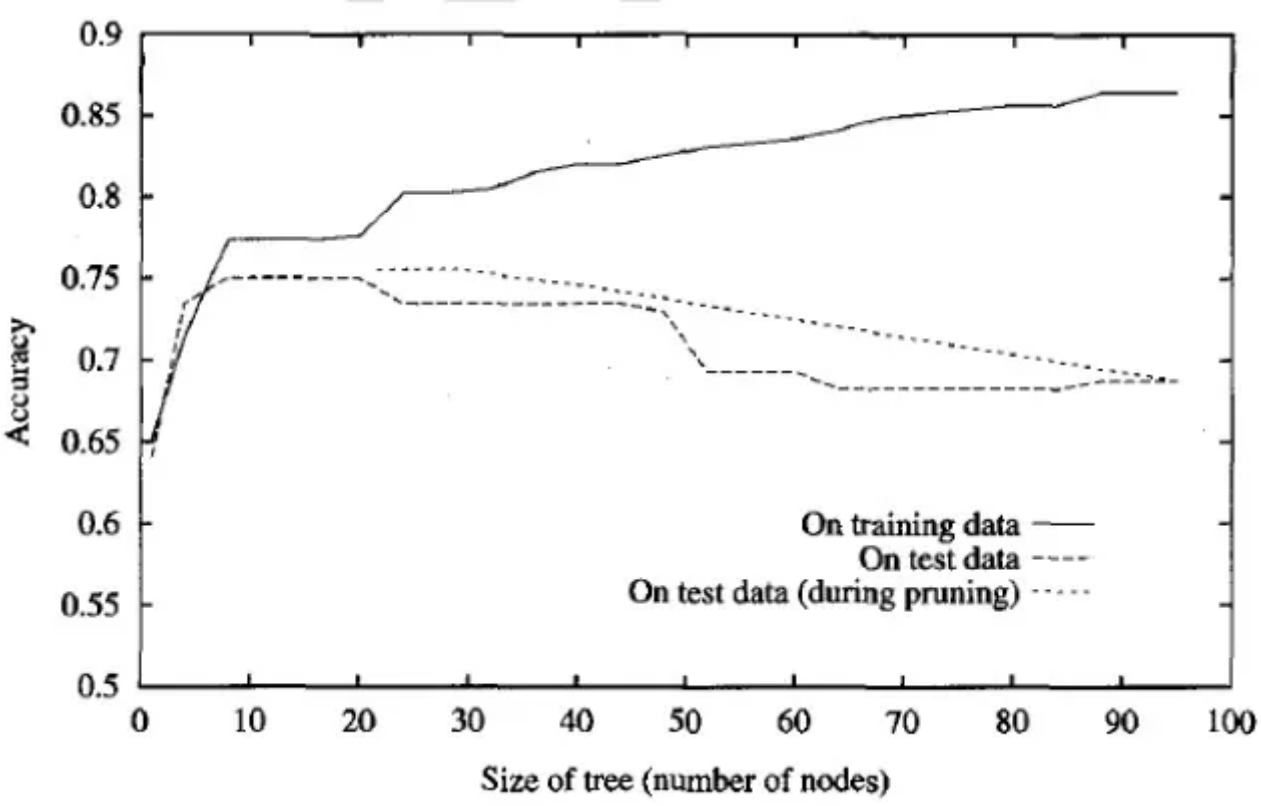
characteristics of the training set. Of course, it is important that the validation set be large enough to itself provide a statistically significant sample of the instances. One common heuristic is to withhold one-third of the available examples for the validation set, using the other two-thirds for training.

1. Reduced Error Pruning

How exactly might we use a validation set to prevent overfitting? One approach, called reduced-error pruning (Quinlan 1987), is to consider each of the decision nodes in the tree to be candidates for pruning.

- Reduced-error pruning, is to consider each of the decision nodes in the tree to be candidates for pruning
- Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.
- Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set

The impact of reduced-error pruning on the accuracy of the decision tree is illustrated in the below figure .



As earlier figure accuracy vs size of the tree, the accuracy of the tree is shown measured over both training examples and test examples.

- The additional line in figure shows accuracy over the test examples as the tree is pruned. When pruning begins, the tree is at its maximum size and lowest accuracy over the test set. As pruning proceeds, the number of nodes is reduced and accuracy over the test set increases.
- The available data has been split into three subsets: the training examples, the validation examples used for pruning the tree, and a set of test examples used to provide an unbiased estimate of accuracy over future unseen examples. The plot shows accuracy over the training and test sets.

Using a separate set of data to guide pruning is an effective approach provided a large amount of data is available. One common heuristic is: the training set constitutes 60% of all data, the validation set 20%, and the test set 20%. The major drawback of this approach is that when data is limited, withholding part of it for the validation set reduces even further the number of examples available for training.

The following section presents an alternative approach to pruning that has been found useful in many practical situations where data is limited. Many additional techniques have been proposed as well, involving partitioning the available data several different times in multiple ways, then averaging the results.

2. Rule Post-Pruning

Rule post-pruning involves the following steps:

- Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.
- Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.
- Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
- Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

To illustrate, consider again the decision tree shown in above figure. In rule post-pruning, one rule is generated for each leaf node in the tree. Each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition) and the classification at the leaf node becomes the rule consequent (postcondition). For example, the leftmost path of the tree in figure is translated into the rule

IF (Outlook = Sunny) ^ (Humidity = High)
THEN PlayTennis = No

Next, each such rule is pruned by removing any antecedent, or precondition, whose removal does not worsen its estimated accuracy. Given the above rule, rule post-pruning would consider removing the preconditions

(Outlook = Sunny) and (Humidity = High)

- It would select whichever of these pruning steps produced the greatest improvement in estimated rule accuracy, then consider pruning the second precondition as a further pruning step.
- No pruning step is performed if it reduces the estimated rule accuracy.

There are three main advantages by converting the decision tree to rules before pruning

- Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree

node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path.

- Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves. Thus, it avoid messy bookkeeping issues such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.
- Converting to rules improves readability. Rules are often easier for to understand.

2. Incorporating Continuous-Valued Attributes

Our initial definition of ID3 is restricted to attributes that take on a discrete set of values.

1. The target attribute whose value is predicted by learned tree must be discrete valued.

2. 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. For an attribute A that is continuous-valued, the algorithm can dynamically create a new boolean attribute A, that is true if A < c and false otherwise. The only question is how to select the best value for the threshold c.

Illustration: Suppose we wish to include the continuous-valued attribute Temperature. Suppose further that the training examples associated with a particular node in the decision tree have the following values for Temperature and the target attribute PlayTennis.

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

What threshold-based boolean attribute should be defined based on Temperature?

- *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.

- In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes: (48 + 60)/2, and (80 + 90)/2.

- The information gain can then be computed for each of the candidate attributes, Temperature >54, and Temperature >85 and the best can be selected (Temperature >54)

- This dynamically created boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

3. Alternative Measures for Selecting Attributes

There is a natural bias in the information gain measure that favors attributes with many values over those with few values.

- As an extreme example, consider the attribute Date, which has a very large number of possible values. What is wrong with the attribute Date? Simply put, it has so many possible values that it is bound to separate the training examples into very small subsets. Because of this, it will have a very high information gain relative to the training examples.

- How ever, having very high information gain, its a very poor predictor of the target function over unseen instances.

Alternate measure-1

One alternative measure that has been used successfully is the gain ratio (Quinlan 1986). The gain ratio measure penalizes attributes such as Date by incorporating a term, called split information that is sensitive to how broadly and uniformly the attribute splits the data:

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

where S1 through Sc, 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 our previous uses of entropy, in which we considered only the entropy of S with respect to the target attribute whose value is to be predicted by the learned tree.

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

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

The SplitInformation term discourages the selection of attributes with many uniformly distributed values (e.g., Date).

One practical issue that arises in using GainRatio in place of Gain to select attributes is that the denominator can be zero or very small when |Si| ≈ |S| for one of the Si. This either makes the GainRatio undefined or very large for attributes that happen to have the same value for nearly all members of S. To avoid selecting attributes purely on this basis, we can adopt some heuristic such as first calculating the Gain of each attribute, then applying the GainRatio test only considering those attributes with above average Gain (Quinlan 1986).

Alternate measure-2

An alternative to the GainRatio, designed to directly address the above difficulty is a distance-based measure introduced by Lopez de Mantaras in 1991. This measure is based on defining a distance metric between partitions of the data. Each attribute is evaluated based on the distance between the data partition it creates and the perfect partition (i.e., the partition that perfectly classifies the training data). The attribute whose partition is closest to the perfect partition is chosen. It is not biased toward attributes with large numbers of values, and the predictive accuracy of the induced trees is not significantly different from that obtained with the Gain and Gain Ratio measures. However, this distance measure avoids the practical difficulties

associated with the GainRatio measure, and in his it produces significantly smaller trees in the case of data sets whose attributes have very different numbers of values.

4. Handling Missing Attribute Values

In certain cases, the available data may be missing values for some attributes. For example, in a medical domain in which we wish to predict patient outcome based on various laboratory tests, it may be that the Blood-Test-Result is available only for a subset of the patients. In such cases, it is common to estimate the missing attribute value based on other examples for which this attribute has a known value.

Consider the situation in which Gain(S, A) is to be calculated at node n in the decision tree to evaluate whether the attribute A is the best attribute to test at this decision node. Suppose that (x, c(x)) is one of the training examples in S and that the value A(x) is unknown.

Method-1

One strategy for dealing with the missing attribute value is to assign it the value that is most common among training examples at node n. Alternatively, we might assign it the most common value among examples at node n that have the classification c(x). The elaborated training example using this estimated value for A(x) can then be used directly by the existing decision tree learning algorithm.

Method-2

A second, more complex procedure is to assign a probability to each of the possible values of A. These probabilities can be estimated again based on the observed frequencies of the various values for A among the examples at node n.

For example, given a boolean attribute A, if node n contains six known examples with A = 1 and four with A = 0, then we would say the probability that A(x) = 1 is 0.6, and the probability that A(x) = 0 is 0.4.

A fractional 0.6 of instance x is now distributed down the branch for A = 1, and a fractional 0.4 of x down the other tree branch. These fractional examples are used for the purpose of computing information Gain and can be further subdivided at subsequent branches of the tree if a second missing attribute value must be tested. This same fractioning of examples can also be applied after learning, to classify new instances whose attribute values are unknown. In this case, the classification of the new instance is simply the most probable classification, computed by summing the

weights of the instance fragments classified in different ways at the leaf nodes of the tree.

This method for handling missing attribute values is used in C4.5

5. Handling Attributes with Differing Costs

In some learning tasks the instance attributes may have associated costs. For example, in learning to classify medical diseases we might describe patients in terms of attributes such as Temperature, BiopsyResult, Pulse, BloodTestResults, etc. These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort.

In such tasks, we would prefer decision trees that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classifications.

ID3 can be modified to consider attribute costs by introducing a cost term into the attribute selection measure. For example, we might divide the Gain by the cost of the attribute, so that lower-cost attributes would be preferred. While such cost-sensitive measures do not guarantee finding an optimal cost-sensitive decision tree, they do bias the search in favor of low-cost attributes.

Method-1

Tan and Schlimmer (1990) and Tan (1993) describe one such approach and apply it to a robot perception task in which the robot must learn to classify different objects according to how they can be grasped by the robot's manipulator. In this case the attributes correspond to different sensor readings obtained by a movable sonar on the robot. Attribute cost is measured by the number of seconds required to obtain the attribute value by positioning and operating the sonar. They demonstrate that more efficient recognition strategies are learned, without sacrificing classification accuracy, by replacing the information gain attribute selection measure by the following measure

$$\frac{Gain^2(S, A)}{Cost(A)}$$

Method-2

Nunez (1988) describes a related approach and its application to learning medical diagnosis rules. Here the attributes are different symptoms and laboratory tests with differing costs. His system uses a somewhat different attribute selection measure, where w ∈ [0, 1] is a constant that determines the relative importance of cost versus information gain.

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

Thanks for Reading .....

Machine Learning | Decision Tree | Tree Pruning | Tom Mitchell

Follow

Written by Pralhad Teggi

157 Followers

Working in Micro Focus, Bangalore, India (14+ Years). Research interests include data science and machine learning.

More from Pralhad Teggi