Inductive Learning

Simplest way to describe it: learn a function (table-tree) from examples

- Ignores prior knowledge
- Assumes a deterministic, observable environment
- Assumes examples are given
- Assumes the "agent" wants to learn the function (for a reason so and so)

Calculating Information Gain

- The information gain is based on the decrease in entropy after a dataset is split on an attribute.
- Which attribute creates the most homogeneous branches?
  - First the entropy of the total dataset is calculated.
  - The dataset is then split on the different attributes.
  - The entropy for each branch is calculated. Then it is added proportionally, to get total entropy for the split.
  - The resulting entropy is subtracted from the entropy before the split.
- The result is the Information Gain, or decrease in entropy.
- The attribute that yields the largest IG is chosen for the decision node.

Calculating Information Gain

Given a set of examples S and an attribute A

- Let p_i be the probability that an arbitrary leaf in S belongs to class C_i, estimated by $|C_i|/|S|$
- Information needed (after using A to split S into v partitions) to classify S:

$$Info_A(S) = \sum_{i=1}^{v} \frac{|C_i|}{|S|} \times I(C_i)$$

- Expected information (entropy) needed to classify a leaf in S:

$$Info(S) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(S) - Info_A(S)$$

- The information is measured in bits.

The ID3 Algorithm

Given a set of examples, S

Described by a set of attributes A_i

Categorised into categories c_j

1. Choose the root node to be attribute A

Such that A scores highest information gain

Relative to S, i.e., gain(S,A) is the highest over all attributes

2. For each value v that A can take

Draw a branch and label each with corresponding v

The ID3 Algorithm

For each branch you've just drawn (for value v)
    If S_v only contains examples in category c
        Then put that category as a leaf node in the tree
Remove A from attributes which can be put into nodes
Replace S with S_v
Find new attribute A scoring best for Gain(S, A)
Start again at part 2

Remark: This is a greedy algorithm: (a form of hill climbing.)

Advantages of using ID3

- Understandable prediction rules are created from the training data.
- Builds the fastest tree.
- Builds a short tree.
- Only need to test enough attributes until all data is classified.
- Finding leaf nodes enables test data to be pruned, reducing number of tests.
- Whole dataset is searched to create tree.

Disadvantages of using ID3

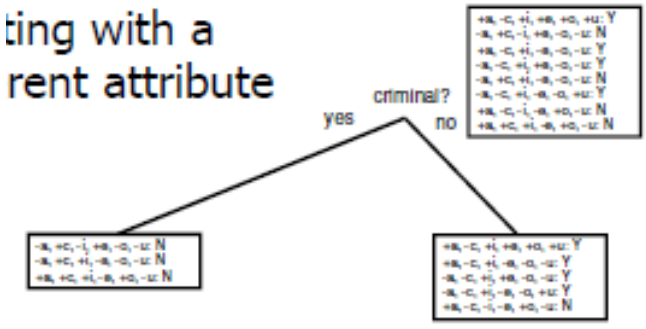
- Data may be over-fitted or over-classified, if a small sample is tested.
- Only one attribute at a time is tested for making a decision.
- Classifying continuous data may be computationally expensive, as many trees must be generated to see where to break the continuum.

Overfitting the DT

The depth of the tree is related to the generalization capability of the tree. If not carefully chosen it may lead to overfitting.

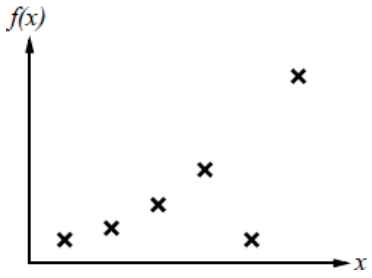
A tree **overfits** the data if we let it grow deep enough so that it begins to capture "adeviation" in the data that harm the predictive power on unseen examples;

Recall



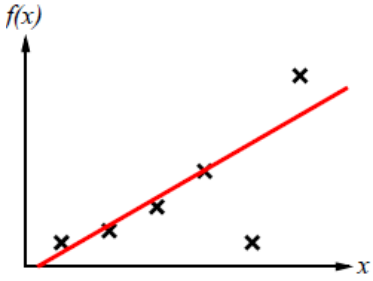
Overfitting in General

Given some points and finding a curve to fit them



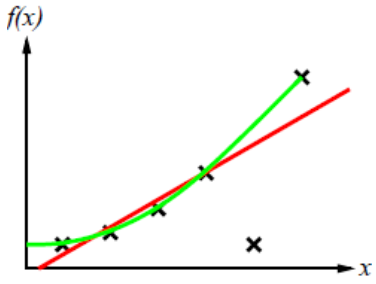
Overfitting in General

Given some points and finding a curve to fit them



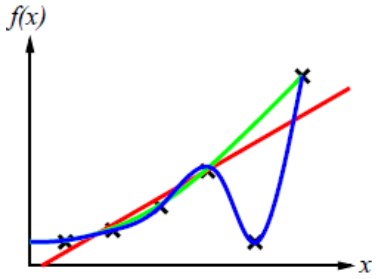
Overfitting in General

Given some points and finding a curve to fit them



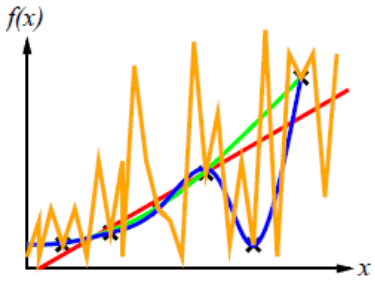
Overfitting in General

Given some points and finding a curve to fit them



Overfitting in General

Given some points and finding a curve to fit them



Ockham's Razor:

Maximize a combination of consistency and simplicity

Overfitting the DT

There are two main solutions to overfitting in a decision tree:

1) Stop the tree early before it begins to overfit the data

→ In practice this solution is hard to implement because it is not clear what is a stopping point.

2) Grow the tree until the algorithm stops even if the overfitting problem shows ,Then prune the tree.

→ This method has found great popularity in the machine learning community

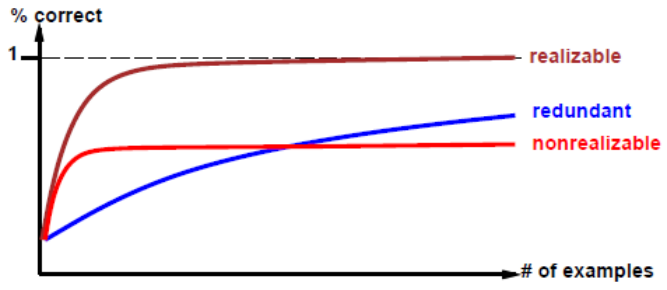
Decision Tree Pruning

common decision tree pruning algorithm depends on :

1- Considering all internal nodes in the tree

2- For each node, check if removing it (along with the subtree) and assigning most common class to it does not harm the accuracy of the data.

Performance Measure



Practical issues in DT

Practical issues while building a decision tree:

1) What happens when attribute values are missing?
2) Choosing a node (using the info gain concept)
2) How deep should the tree be?
3) How do we handle continuous attributes (So far we discussed only discrete)?
5) How do we improve the computational efficiency

Missing Data

Missing data: *In many domains, not all the attribute values will be known for every* example. The values might have gone unrecorded, or they might be too expensive to obtain. This gives rise to two problems:

→ First, given a complete decision tree, how should one classify an object that is missing one of the test attributes?

→ Second, how should one modify the information gain formula when some examples have unknown values for the attribute?

Multi-valued Attributes

Multivalued attributes*: When an attribute has many possible values, the information* gain measure gives an inappropriate indication of the attribute's usefulness.

In the extreme case, we could use an attribute, such as *Name, that has a different* value for every example. Then each subset of examples would be a singleton with a unique classification, so the information gain measure would have its highest value for this attribute. While, the attribute could be irrelevant or useless.

Continuous attributes

Continuous and integer-valued input attributes*:*

such as Height and Weight, have an infinite set of possible values. Rather than generate infinitely many branches, decision-tree learning algorithms typically find the

split point that gives the highest information gain.

For example, at a given node in the tree, it might be the case that testing on Weight > 160 gives the most information.

Efficient dynamic programming methods exist for finding good split points, but it is still by far the most expensive part of real-world decision tree learning applications.