Chapter 3
Decision Tree Learning

3.1 INTRODUCTION

- Decision trees learning is a method for inductive inference (reaching a general conclusion from specific examples).
- It's a method for approximating discrete-values functions.
- Learned trees can also re-represented as sets of if-then rules.

3.2 DECISION TREES REPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node specifies a test of some attribute and each branch corresponds to one of the possible values for this attribute.
- An instance is classified by the following steps:
  1. Start at the root node.
  2. Test the attribute specified by this node.
  3. Move down the tree branch corresponding to the value of the attribute in the given example.
  4. Repeat for the subtree rooted at the new node.
- Decision trees represent a disjunction of conjunctions on constraints in the attribute values of instances.

3.3 APPROPRIATE PROBLEMS FOR DECISION TREES LEARNING

- Instances are represented by attribute-value pairs.
- The target function has discrete output values.
- Disjunctive descriptions maybe required.
- The training data may contain errors.
- The training data may contain missing attribute values.

3.4 <u>THE BASIC DECISION TREE LEARNING ALGORITHM</u>

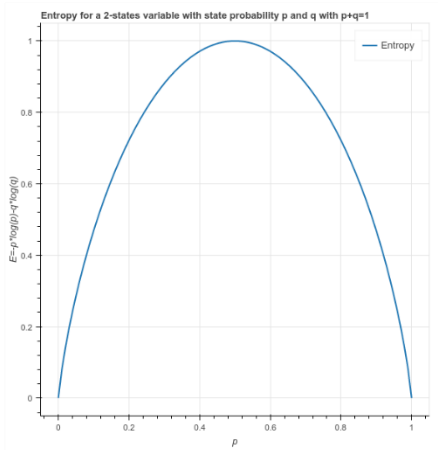
- The core decision tree algorithm employs a top-down, greedy search through the space of possible decision trees called the ID3 algorithm.
- ID3 learns decision trees by constructing them top-down:
  1. Each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples.
  2. The best attribute is selected and used as a test at the root node.
  3. A descendent of root node is created for each possible value of this attribute.
  4. The training examples are sorted to the appropriate descendent node.
  5. The entire process is then repeated for each descendent node.

  NOTE: The algorithm never backtracks to reconsider earlier choices.
- Which attribute is the best classifier?
  - Information gain: A measure of how well a given attribute separates the training examples according to their target information.
  - ID3 uses information gain to select among the candidate attributes at each step of the tree.
- Information Gain:
  - Entropy: The impurity of an arbitrary collection of examples.

$$Entropy\ (S) = -p_{+} \log_2 p_{+} - p_{-} \log_2 p_{-}$$

P₊ → The proportion of positive examples in *S*.

P₋ → The proportion of negative examples in *S*.



1. Entropy is 0 if all members of *S* belong to the same class.
2. Entropy is 1 if the collection contains an equal number of positive and negative examples.
3. Entropy is between 0 and 1 if the collection contains an unequal number of positive and negative examples.
4. More generally, if the target attribute can take on *c* different values, then the entropy *S* will be:

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

NOTE: The logarithm is base 2 because entropy is a measure of the expected encoding length measured in bits.

- The Information Gain is the expected reduction in entropy caused by partitioning the examples according to a specific attribute.

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

Values(A) → The set of all possible values for attribute *A*.

S_v → The subset of *S* for which attribute *A* has value *v*.

The second term in the equation denotes the sum of entropies of each subset S_v, weighted by the fraction of examples $\frac{|S_v|}{|S|}$ that belongs to S_v.

$Gain(S, A)$ is therefore the expected reduction in entropy caused by knowing the attribute *A*.

3.5 HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- The core decision tree algorithm employs a top-down, greedy search through the space of possible decision trees called the ID3 algorithm.
  - The hypothesis space searched by ID3 is a set of possible decision trees.
  - ID3 performs a simple-to-complex, hill-climbing search through this hypothesis space, beginning with an empty tree, then considering progressively more complex hypotheses in search of a tree the correctly fits the training data.
  - The Information Gain function guides ID3 to the best decision tree.
- ID3 basics:
  - Hypothesis space is complete: The target function is surely in this hypothesis space.
  - ID3 always maintains (outputs) a single hypothesis: As a result, it cannot determine how many alternative decision trees are consistent with the training data.
  - No backtracking: Once ID3 selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice. This puts ID3 under the risk of converging to a local minima (A solution that is not globally optimum).
  - Statistically-based search choices: Robust to noisy data.

3.6 INDUCTIVE BIAS IN DECISION TREE LEARNING

- Inductive Bias is the set of assumptions that, together with the training data, deductively justify the classifications assigned by the learning algorithm to future instances. In other words, it’s the policy by which a learning algorithm generalizes from observed training examples to classify unseen instances.
- Approximate Inductive Bias of ID3:
  - Shorter trees are preferred over longer trees.
  - Trees that places high information gain attributes close to the root are preferred over those that do not.
- Types of Inductive Bias:
  - Preference Bias: Searching a complete hypothesis space in an incomplete way till finding **a hypothesis consistent** with the data.
  - Restriction Bias: Searching an incomplete hypothesis space (One that express only a subset of the potentially teachable concepts) in a complete way, finding **every hypothesis consistent** with the training data.

  Preference Bias is more desirable because it allows the learner to work in a complete hypothesis space that is assured to contain the target function.
- ID3’s exhibits a Preference Bias.

- Why Prefer Short Hypothesis?
  - Occam's Razor: Prefer the simplest hypothesis that fits the data.
  - Fewer short hypothesis than long ones:

    Argument in favor:

    → A short hypothesis that fits the data is less likely to be a statistical coincidence.

    Argument opposed:

    → There're many ways to define small sets of hypotheses.

    → The size of a hypothesis is determined by the particular representation used internally by the learner.

3.7 <u>ISSUES IN DECISSION TREE LEARNING</u>

- Practical Issues in Learning Decision Trees Include:
  - Determining how deeply to grow the decision tree.
  - Handling continuous attributes.
  - Choosing an appropriate attribute selection measure.
  - Handing data with missing attribute values.
  - Handling attributes with differing costs.
  - Improving computational efficiency.
- Overfitting the Data:
  - A hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution of instances, including instances beyond the training set.
  - Formal definition: 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 that *h'* over the training examples, but *h'* has smaller error that *h* over the entire distribution of instances.
  - Why overfitting happens:

    → When the training examples contain random errors or noise.

    → When small number of examples are associated with leaf nodes, it's 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.
  - Approaches to avoid overfitting:

    → Stop growing the tree earlier before it overfits.

    → Allow the tree to overfit, then post-prune it.
  - Approaches to determine the correct final tree size:

    → Use a separate set of examples to evaluate the utility of post-pruning (A validation set).

    → Use all the available data for training, then apply a statistical test (e.g. chi-square) to estimate whether expanding or pruning the tree is likely to produce an improvement beyond the training set.

    → Use an explicit measure of complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized.

- Reduced error pruning:
    - → Reduced error pruning is used to lower the possibility of overfitting.
    - → Steps:
        1. Split data into training and validation set.
        2. For each possible node, remove the subtree rooted at that node, and assign to the resulting leaf the most common classification of the training examples affiliated with that node.
        3. Evaluate the performance of the resulting tree using the validation set.
        4. Greedily remove the nodes that most improve the validation set accuracy.
        5. Pruning of nodes continues until further pruning decreases the validation set accuracy.
- Rule post-pruning:
    - → Steps:
        1. Infer the decision tree from the training set and grow the tree allowing for overfitting.
        2. Convert the learned tree to an equivalent set of rules (One rule for each path from root to leaf).
        3. Prune each rule (Reduce its preconditions) independently from others.
        4. Sort the pruned rules by their estimated accuracy.
    - → Advantages:
        1. Converting to rules allows distinguishing among the different contexts in which a decision node is used.
        2. Converting to rules removes the distinction between attributes tests that occur near the root and those that occur near the leaves.
- Incorporating Continuous-Values Attributes:
    - → ID3 restrictions:
        1. The target attribute whose value is predicted by the learned tree must be discrete valued.
        2. The attributes tested in the decision nodes must also be discrete valued.
    - → One approach to solve this issue is to dynamically define new discrete values attributes that partition the continuous attribute value into a discrete set of intervals.

• Alternative Measure for Selecting Attributes:
- There's a bias in the Information Gain measure that favors attributes with many values over those with few values. An attribute with so many possible values is bound to separate the training examples into very small subsets.
- One solution to this problem is to select attributes based on some measure other than the Information Gain.
- Gain Ratio: This measure penalizes attributes with many values by incorporating a term called "split information" that is sensitive to how broadly and uniformly the attribute splits the data:

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

    - → *S_1* through *S_c* are the *c* subsets of examples resulting form partitioning *S* by the *c*-valued attribute *A*.
    - → *SplitInformation* is actually the entropy of *S* with respect to the values of attribute *A*.

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

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

- One issue arises in using Gain Ratio in place of Gain is that the denominator can be zero or very small when $|S_i|$ ~= $|S|$ from one S_i.
  → To avoid this issue, we can first calculate the Gain for each attribute, and then applying the Gain Ratio test only considering only attributes with above average Gain.

• Handling Training Examples with Missing Attribute Values:
  - What if some examples missing values of *A*?
    1. If node *n* tests *A*, assign most common value of *A* among other examples sorted to node *n*.
    2. Assign most common values of *A* among other examples with the same target value.
    3. Assign probability p_i to each possible value v_i of *A*, then assign fractional p_i of example to each descendent in the tree to calculate the Information Gain.