Learning Decision Trees

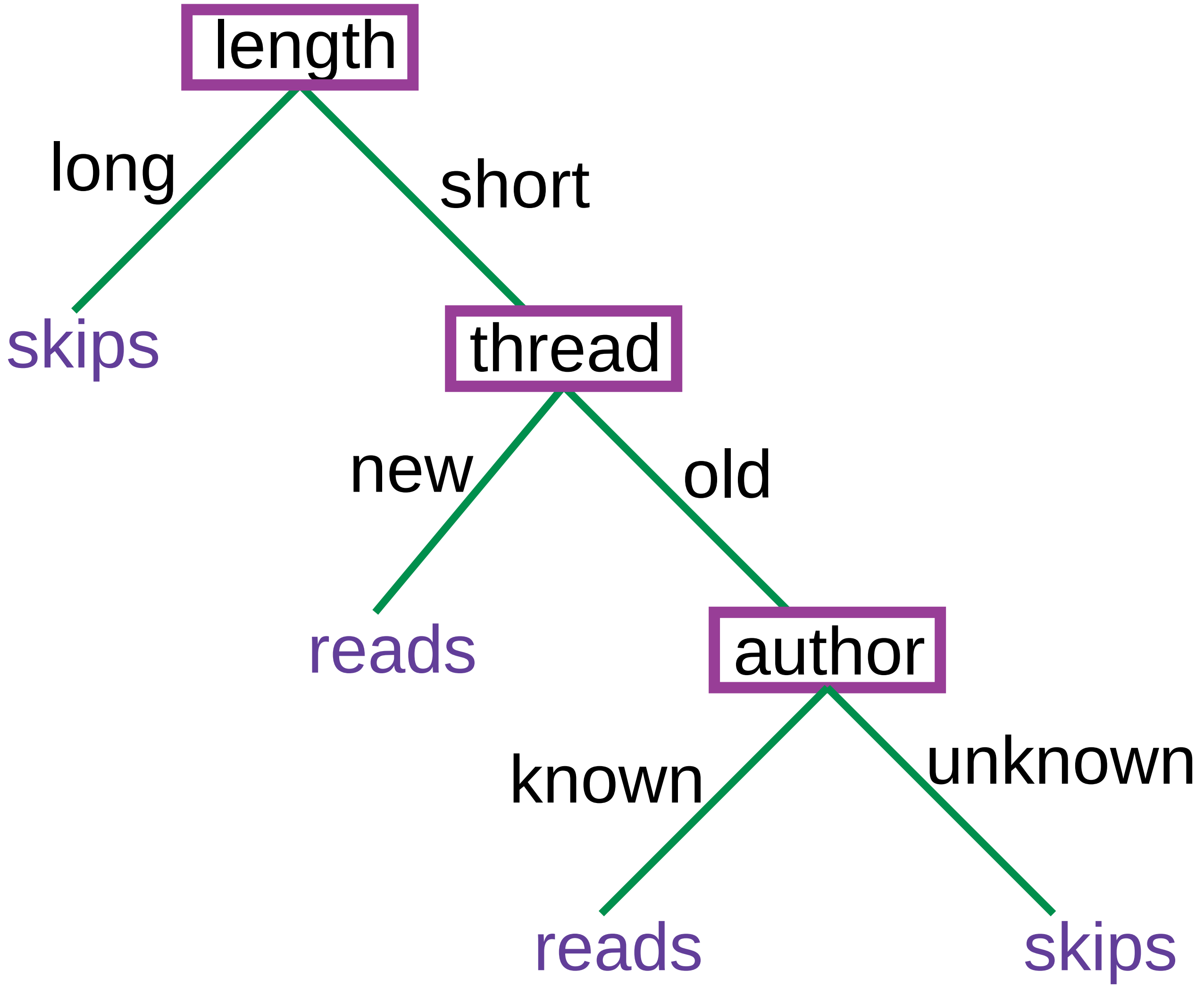
- Representation is a decision tree.
- Bias is towards simple decision trees.
- Search through the space of decision trees, from simple decision trees to more complex ones.

Decision trees

A decision tree is a tree where:

- The nonleaf nodes are labeled with attributes.
- The arcs out of a node labeled with attribute *A* are labeled with each of the possible values of the attribute *A*.
- The leaves of the tree are labeled with classifications.

Example Decision Tree



Equivalent Logic Program

$$prop(Obj, user\_action, skips) \leftarrow$$
$$prop(Obj, length, long).$$
$$prop(Obj, user\_action, reads) \leftarrow$$
$$prop(Obj, length, short) \wedge prop(Obj, thread, new).$$
$$prop(Obj, user\_action, reads) \leftarrow$$
$$prop(Obj, length, short) \wedge prop(Obj, thread, old) \wedge$$
$$prop(Obj, author, known).$$
$$prop(Obj, user\_action, skips) \leftarrow$$
$$prop(Obj, length, short) \wedge prop(Obj, thread, old) \wedge$$
$$prop(Obj, author, unknown).$$

Issues in decision-tree learning

- Given some data, which decision tree should be generated? A decision tree can represent any discrete function of the inputs.
- You need a bias. Example, prefer the smallest tree. Least depth? Fewest nodes? Which trees are the best predictors of unseen data?
- How should you go about building a decision tree? The space of decision trees is too big for systematic search for the smallest decision tree.

Searching for a Good Decision Tree

- The input is a target attribute (the *Goal*), a set of examples, and a set of attributes.
- Stop if all examples have the same classification.
- Otherwise, choose an attribute to split on,
  - for each value of this attribute, build a subtree for those examples with this attribute value.

Decision tree learning: Boolean attributes

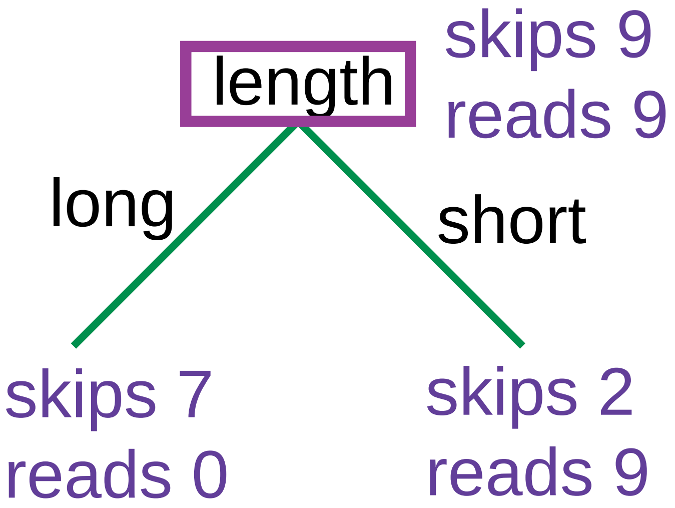
% $dtlearn(Goal, Examples, Attributes, DT)$ given $Examples$
% and $Attributes$ construct decision tree DT for $Goal$.

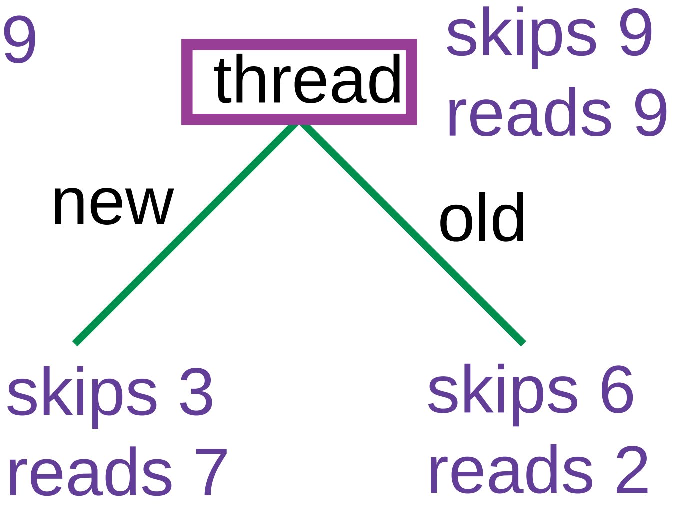
```
    $dtlearn(Goal, Exs, Atts, Val) \leftarrow$
        $all\_examples\_agree(Goal, Exs, Val).$
    $dtlearn(Goal, Exs, Atts, if(Cond, YT, NT)) \leftarrow$
        $examples\_disagree(Goal, Exs) \wedge$
        $select\_split(Goal, Exs, Atts, Cond, Rem\_Atts) \wedge$
        $split(Exs, Cond, Yes, No) \wedge$
        $dtlearn(Goal, Yes, Rem\_Atts, YT) \wedge$
        $dtlearn(Goal, No, Rem\_Atts, NT).$
```

Extended Example

Example	User Action	Author	Thread	Length	Where Read
e1	skips	known	new	long	home
e2	reads	unknown	new	short	work
e3	skips	unknown	old	long	work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work
e7	skips	unknown	old	short	work
e8	reads	unknown	new	short	work
e9	skips	known	old	long	home
e10	skips	known	new	long	work
e11	skips	unknown	old	short	home
e12	skips	known	new	long	work
e13	reads	known	old	short	home
e14	reads	known	new	short	work
e15	reads	known	new	short	home
e16	reads	known	old	short	work
e17	reads	known	new	short	home
e18	reads	unknown	new	short	work

Example: possible splits





Using this algorithm in practice

- Attributes can have more than two values. This complicates the trees.
- This assumes attributes are adequate to represent the concept. You can return probabilities at leaves.
- Which attribute to select to split on isn't defined. You want to choose the attribute that results in the smallest tree. Often we use information theory as an evaluation function in hill climbing.
- Overfitting is a problem.

Handling Overfitting

- This algorithm gets into trouble overfitting the data. This occurs with noise and correlations in the training set that are not reflected in the data as a whole.
- To handle overfitting:
  - You can restrict the splitting, so that you split only when the split is useful.
  - You can allow unrestricted splitting and prune the resulting tree where it makes unwarranted distinctions.

Maximizing Information Gain

Select attributes in the order of maximal information gain

$H(G)$	entropy of source regarding goal attribute G with distribution according to example set
$H(G\|A=a_i)$	entropy of same source based on subset of examples with $A = a_i$
q_i	fraction of example set with $A = a_i$
IG	information gain by asking for the attribute value of A

$$\boxed{IG = H(G) - \sum_i q_i H(G \mid A = a_i)}$$

Information theory overview

- A bit is a binary digit.
- 1 bit can distinguish 2 items
- k bits can distinguish 2^k items
- n items can be distinguished using $\log_2 n$ bits
- Can you do better?

Information and Probability

Let's design a code to distinguish elements of $\{a, b, c, d\}$ with

$$P(a) = \frac{1}{2}, P(b) = \frac{1}{4}, P(c) = \frac{1}{8}, P(d) = \frac{1}{8}$$

Consider the code:

a	0	b	10	c	110	d	111

This code sometimes uses 1 bit and sometimes uses 3 bits.

On average, it uses

$$\begin{aligned} & P(a) \times 1 + P(b) \times 2 + P(c) \times 3 + P(d) \times 3 \\ = \quad & \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{3}{8} = 1\frac{3}{4} \text{ bits.} \end{aligned}$$

The string *aacabbda* has code 00110010101110.

Information Content

- To identify x, you need $-\log_2 P(x)$ bits.
- If you have a distribution over a set and want to a identify a member, you need the expected number of bits:

  $$\sum_x -P(x) \times \log_2 P(x).$$

  This is the information content or entropy of the distribution.
- The expected number of bits it takes to describe a distribution given evidence e:

  $$I(e) = \sum_x -P(x|e) \times \log_2 P(x|e).$$

Information Gain

If you have a test that can distinguish the cases where α is true from the cases where α is false, the information gain from this test is:

$$I(true) - (P(\alpha) \times I(\alpha) + P(\neg\alpha) \times I(\neg\alpha)).$$

- $I(true)$ is the expected number of bits needed before the test
- $P(\alpha) \times I(\alpha) + P(\neg\alpha) \times I(\neg\alpha)$ is the expected number of bits after the test.