Inductive Learning

Readings: Chapter 18 of Russell & Norvig.

Learning Agents

- A distinct feature of intelligent agents in nature is their ability to **learn** from experience.
- Using his experience *and* his internal knowledge, a learning agent is able to produce **new knowledge**.
- That is, given his internal knowledge and a percept sequence, the agent is able to learn facts that
  - are *consistent* with both the percepts and the previous knowledge,
  - *do not just follow* from the percepts and the previous knowledge.

Example: Learning for Logical Agents

Learning in logical agents can be formalized as follows.

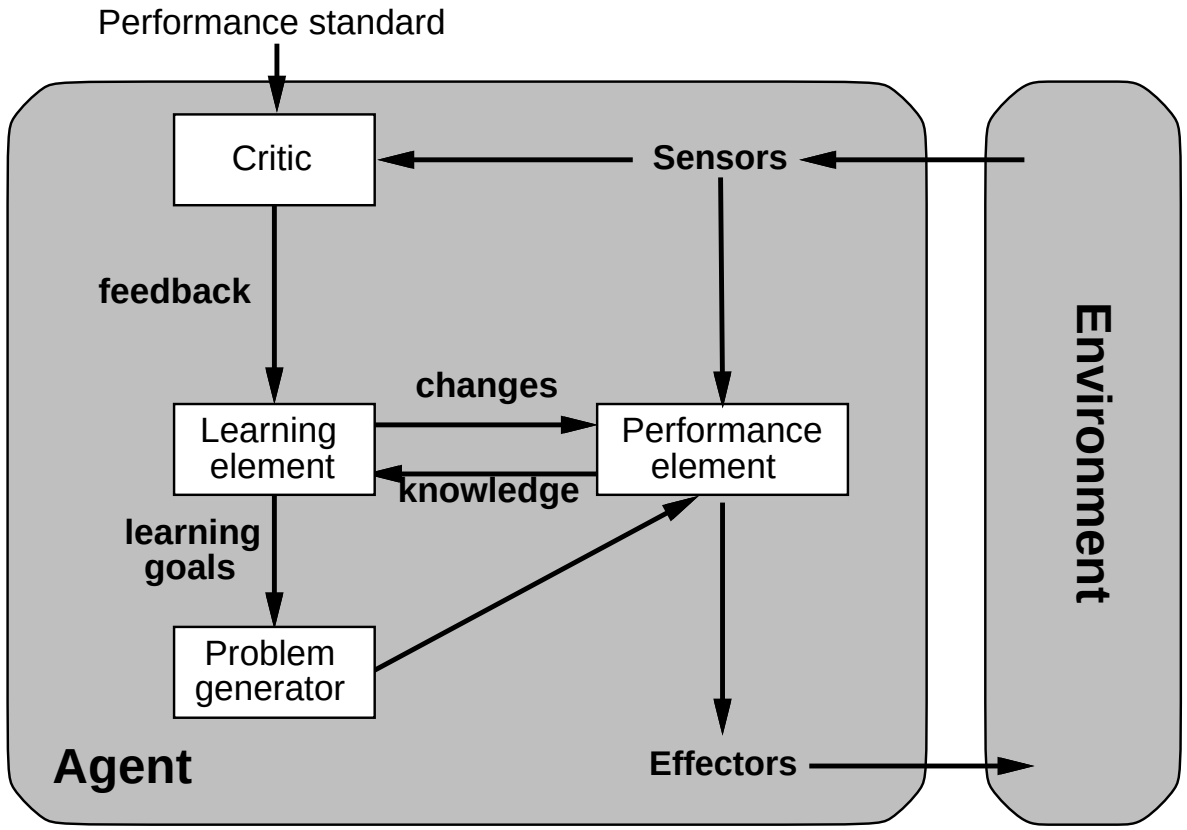
Let Γ, Δ be set of sentences where

- Γ is the agent's knowledge base (the agent's current knowledge),
- Δ is a representation of a percept sequence (the evidential data).

A learning agent is an agent able to generate facts φ from Γ and Δ such that

- $\Gamma \cup \Delta \cup \{\varphi\}$ is satisfiable (consistency of φ),
- usually, $\Gamma \cup \Delta \not\models \varphi$ (novelty of φ).

Learning Agent: Conceptual Components



Learning Elements

Machine learning research has produced a large variety of learning elements.

Major issues in the design of learning elements:

- Which **components** of the performance element are to be improved.
- What **representation** is used for those components.
- What kind of **feedback** is available.
  - **supervised** learning
  - **reinforcement** learning
  - **unsupervised** learning
- What **prior knowledge** is available.

Learning as Learning of Functions

Any component of a performance element can be described mathematically as a function:

- condition-action rules
- predicates in the knowledge base
- next-state operators
- goal-state recognizers
- search heuristic functions
- belief networks
- utility functions
- ...

All learning can be seen as learning the representation of a function.

Inductive Learning

A lot of learning is of an *inductive* nature.

Given some experimental data, the agent learns the general principles governing those data and is able to make correct predictions on future data, based on these general principles.

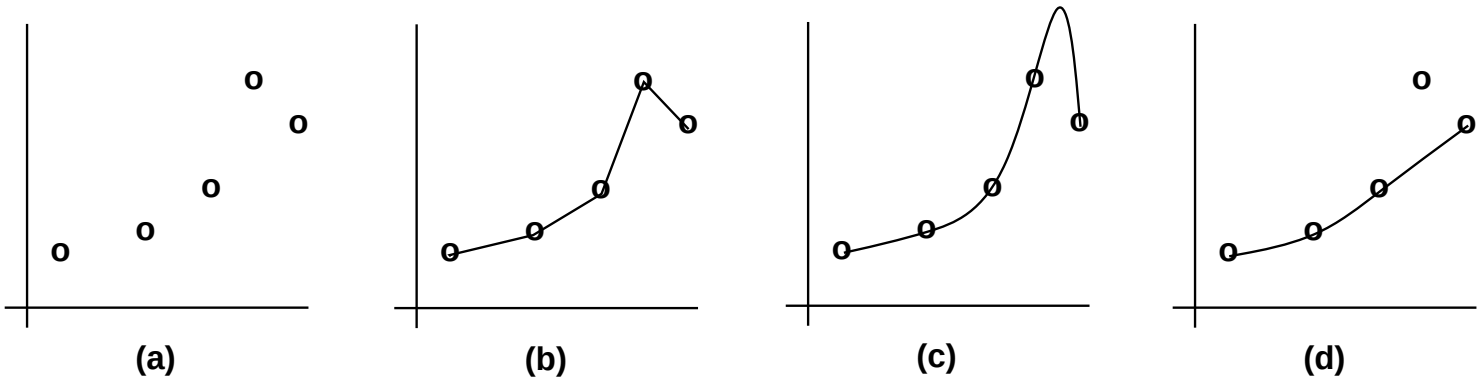
Examples:

- After a baby is told that certain objects in the house are chairs, the baby is able to learn the concept of "chair" and then recognize *previously unseen* chairs as such.
- Your grandfather watches a soccer match for the first time and from the action and the commentators' report is able to figure out the rules of the game.

Purely Inductive Learning

Given a collection $\{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ of input/output pairs, or **examples**, for a function f,

produce a **hypothesis**, (a compact representation of) a function h that approximates f.



In general, there are quite a lot of different hypotheses consistent with the examples.

Bias in Learning

Any kind of preference for a hypothesis h over another is called a **bias**.

Bias is inescapable

Just the choice of formalism to describe h already introduces a bias.

Bias is necessary

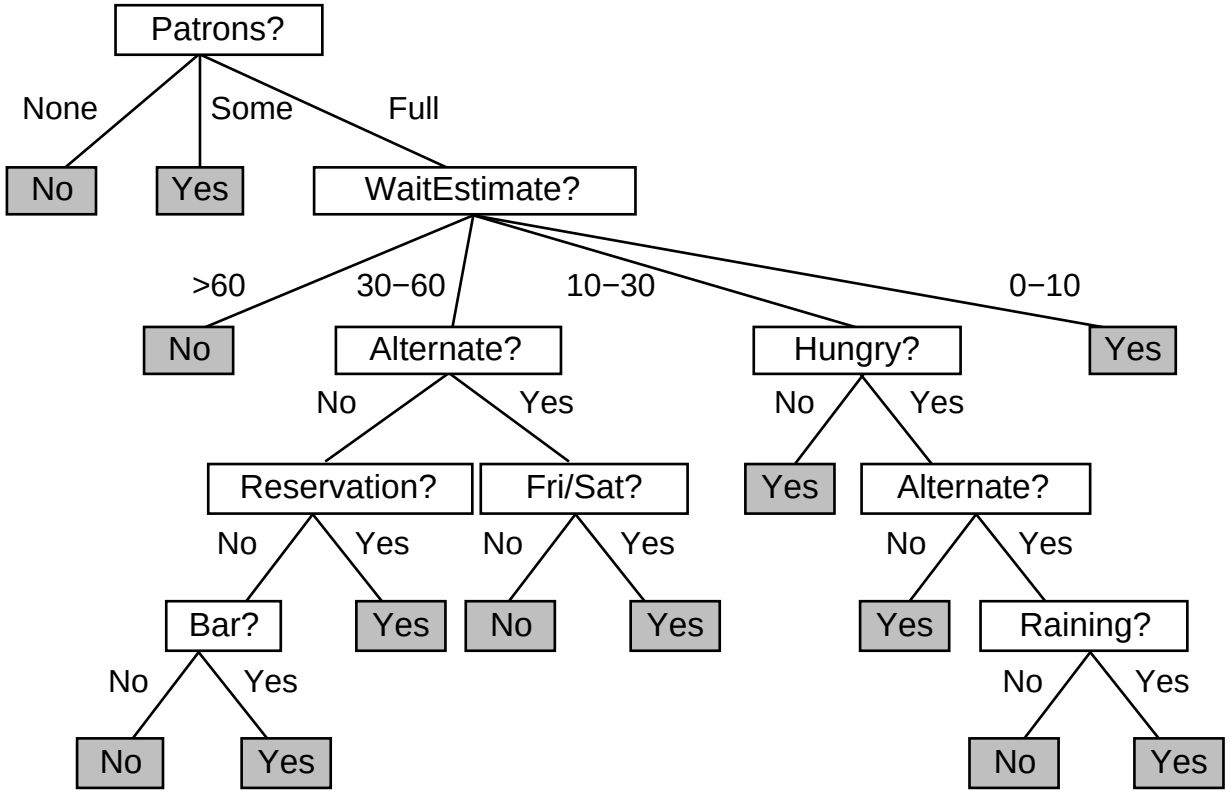
Learning is nearly impossible without bias.
(Which of the many hypotheses do you choose?)

Learning Decision Trees

- The simplest form of inductive learning occurs in learning decision trees.
- A **decision tree** is a Boolean operator that takes as input a set of properties describing an object or a situation, and outputs a yes/no decision.
- It is represented by a tree in which
  - every non-leaf node corresponds to a test on the value of one of the properties.
  - every leaf node specifies the value to be returned (yes or no) if that leaf is reached.

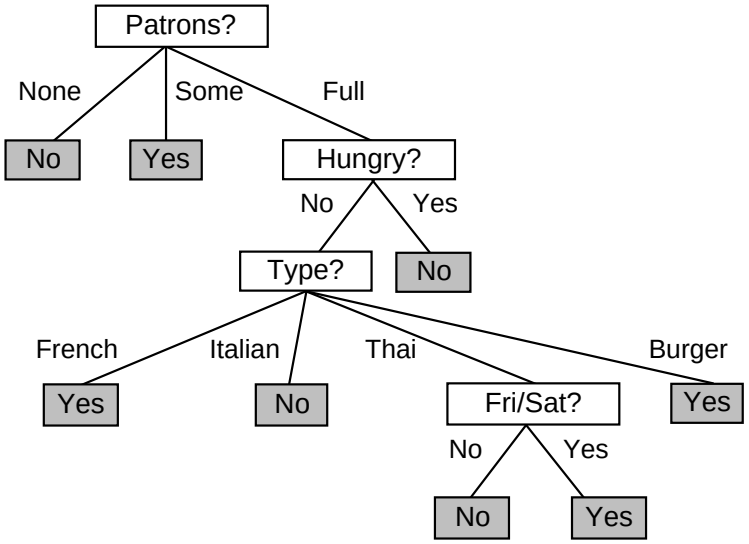
A Decision Tree

The following tree can be used to decide whether to wait for a table at a given restaurant.



A Decision Tree as Predicates

A decision tree is nothing but the definition of a logical predicate.



$$\begin{aligned}
\forall r \;\; WaitAt(r) \;\Leftrightarrow\; & Patrons(r) = Some \;\vee \\
& Patrons(r) = Full \wedge \neg Hungry \wedge Type(r) = French \;\vee \\
& Patrons(r) = Full \wedge \neg Hungry \wedge Type(r) = Burger \;\vee \\
& Patrons(r) = Full \wedge \neg Hungry \wedge Type(r) = Thay \wedge FriSat
\end{aligned}$$

Building Decision Trees

How can we build a decision tree for a specific predicate?

We can look at a number of examples that satisfy, or do not satisfy, the predicate and try to *extrapolate* the tree from them.

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Some Terminology

- The **goal predicate** is the predicate to be implemented by a decision tree.
- The **training set** is the set of examples used to build the tree.
- A member of the training set is a **positive example** if it is satisfies the goal predicate. It is a **negative example** if it does not.
- A decision tree can also be seen as a **classifier**:

  given a potential instance of a goal predicate, it is able to say, by looking at some attributes of the instance, whether the instance is a positive example of the predicate or not.

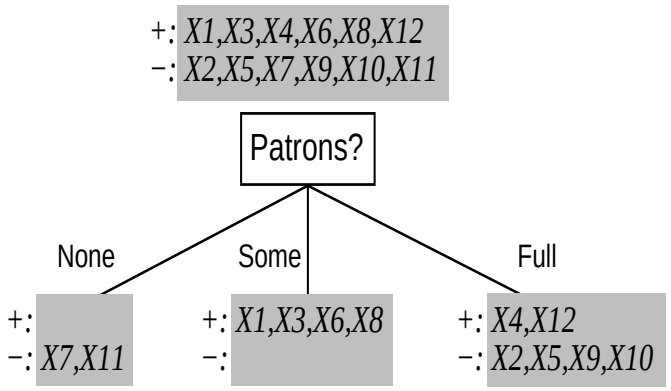
Good Decision Trees

- It is trivial to construct a decision tree that agrees with a given training set. (How?)
- However, the trivial tree will simply memorize the given examples.
- We want a tree that extrapolates a *common pattern* from the examples.
- We want the tree to correctly classify *all* possible examples, not just those in the training set.

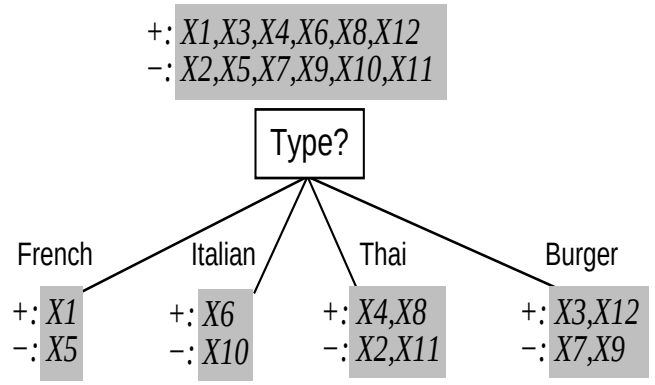
Looking for Decision Trees

- In general, there are several decision trees that describe the *same* goal predicate. Which one should we prefer?
- **Ockham's razor:** always prefer the simplest description, that is, the smallest tree.
- **Problem:** searching through the space of possible trees and finding the smallest one is possible but takes exponential time.
- **Solution:** apply some simple heuristics that lead to small (if not smallest) trees.
- **Main Idea:** start building the tree by testing at its root an attribute that better splits the training set into *homogeneous* classes.

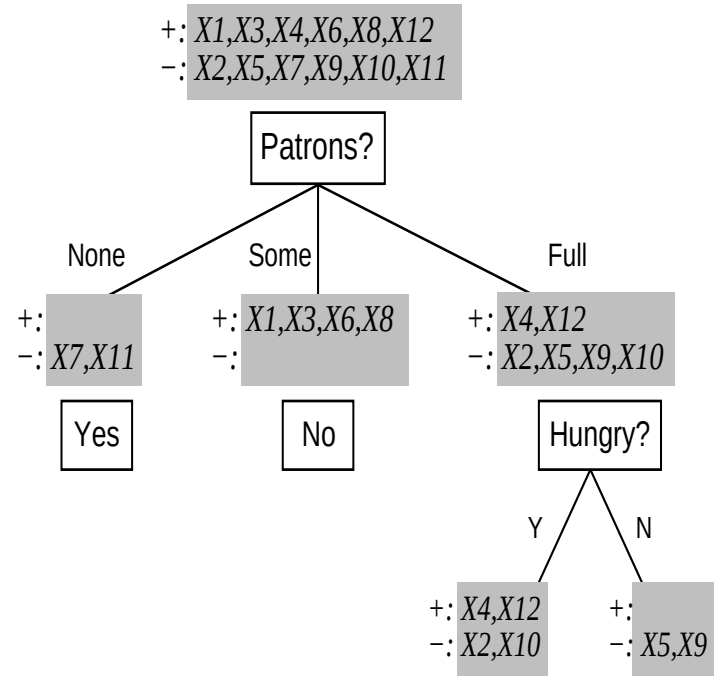
(a)


+: X1,X3,X4,X6,X8,X12
−: X2,X5,X7,X9,X10,X11
Patrons?
None
Some
Full
+:
−: X7,X11
+: X1,X3,X6,X8
−:
+: X4,X12
−: X2,X5,X9,X10

(b)


+: X1,X3,X4,X6,X8,X12
−: X2,X5,X7,X9,X10,X11
Type?
French
Italian
Thai
Burger
+: X1
−: X5
+: X6
−: X10
+: X4,X8
−: X2,X11
+: X3,X12
−: X7,X9

(c)


+: X1,X3,X4,X6,X8,X12
−: X2,X5,X7,X9,X10,X11
Patrons?
None
Some
Full
+:
−: X7,X11
+: X1,X3,X6,X8
−:
+: X4,X12
−: X2,X5,X9,X10
Yes
No
Hungry?
Y
N
+: X4,X12
−: X2,X10
+:
−: X5,X9

Building the Tree: General Procedure

1. Choose for the root node test the attribute that *best* partitions the given training set E into homogeneous sets.
2. If the chosen attribute has n possible values, it will partition E into n sets, $E_1, \ldots, E_n$ say.

   Add a branch i to the root node for each set E_i.
3. For each branch i:

   3.1. If E_i contains only positive examples, add a yes leaf to the branch.

   3.2. If E_i contains only negative examples, add a no leaf to the branch.

   3.3. Otherwise, add a non-leaf node to the branch and apply the procedure recursively to that node with E_i as the training set.

Choosing the Best Attribute

- What do we exactly mean by "best partitions the training set into homogeneous classes?"
- What if each attribute splits the training set into non-homogeneous classes?
- Which one is better?
- **Information Theory** can be used to devise a measure of goodness for attributes.

Information Theory

- Information Theory studies the mathematical laws governing systems designed to communicate or manipulate information.
- It defines quantitative measures of information and the capacity of various systems to transmit, store, and process information.
- In particular, it measures the **information content** of *messages (or, events)*.
- Information is measured in **bits**.
- One bit represents the information we need to answer a yes/no question when we have no idea about the answer.

Information Content

If an event has n possible outcomes v_i, each with probability $P(v_i)$, the information content I of the actual outcome is

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i)$$

that is, the average information content of each outcome $(-\log_2 P(v_i))$ weighted by the outcome's probability.

Example

The information content of tossing a fair coin is

$$I(P(head), P(tail)) = I(\frac{1}{2}, \frac{1}{2}) = -\frac{1}{2}\log_2 \frac{1}{2} - \frac{1}{2}\log_2 \frac{1}{2} = \frac{1}{2} + \frac{1}{2} = 1\ bit$$

The information content of tossing a coin with heads on both sides is

$$I(P(head), P(tail)) = I(1, 0) = -1 \log_2 1 - 0 \log_2 0 = 0 - 0 = 0\ bit$$

Information Content of a Decision Tree

- For decision trees, the event is question is whether the tree will answer "yes" or "no" to a given input example e.
- Assume the training set E is a *representative sample* of the domain.
- The probability of a yes (no) answer before a *correct* decision tree is applied to e can be estimated by the relative frequency of positive (negative) examples in E:

$$P(yes) = \frac{p}{p+n} \qquad P(no) = \frac{n}{p+n}$$

  where p is the number of positive examples and n the number of negative examples in E.
- Then, the information content of the whole tree is estimated by

$$I(\frac{p}{p+n}, \frac{n}{p+n}) \;=\; -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information Content of an Attribute

- Testing a single attribute A in the tree will provide only *some* of the information provided by the whole tree.

- But we can measure how much information is still needed after A has been tested.

Information Content of an Attribute

- Let $E_1, \ldots, E_m$ be the sets into which A divides the *current* training set E.
- Where $i = 1, \ldots, m$, let

$$\begin{aligned} p &= \#\text{ of positive examples in } E \\ n &= \#\text{ of negative examples in } E \\ p_i &= \#\text{ of positive examples in } E_i \\ n_i &= \#\text{ of negative examples in } E_i \end{aligned}$$

- Then, along branch i of node A we will need

$$Remainder(A) = \sum_{i=1}^{m} \frac{p_i + n_i}{p + n} I(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i})$$

extra bits of information to classify the input example *after* we have checked attribute A.

Choosing an Attribute

Conclusion: The smaller the value of $Remainder(A)$, the higher the information content of attribute A for the purpose of classifying the input example.

Heuristic: When building a non-leaf node of a decision tree, choose the attribute with the *smallest* remainder.

Building Decision Trees: An Example

Using the information below about several production runs in a given factory, construct a decision tree to determine the factors that influence production output.

Run	Supervisor	Operator	Machine	Overtime	Output
1	Patrick	Joe	a	no	high
2	Patrick	Samantha	b	yes	low
3	Thomas	Jim	b	yes	low
4	Patrick	Jim	b	no	high
5	Sally	Joe	c	no	high
6	Thomas	Samantha	c	no	low
7	Thomas	Joe	c	no	low
8	Patrick	Jim	a	yes	low

Building Decision Trees: An Example

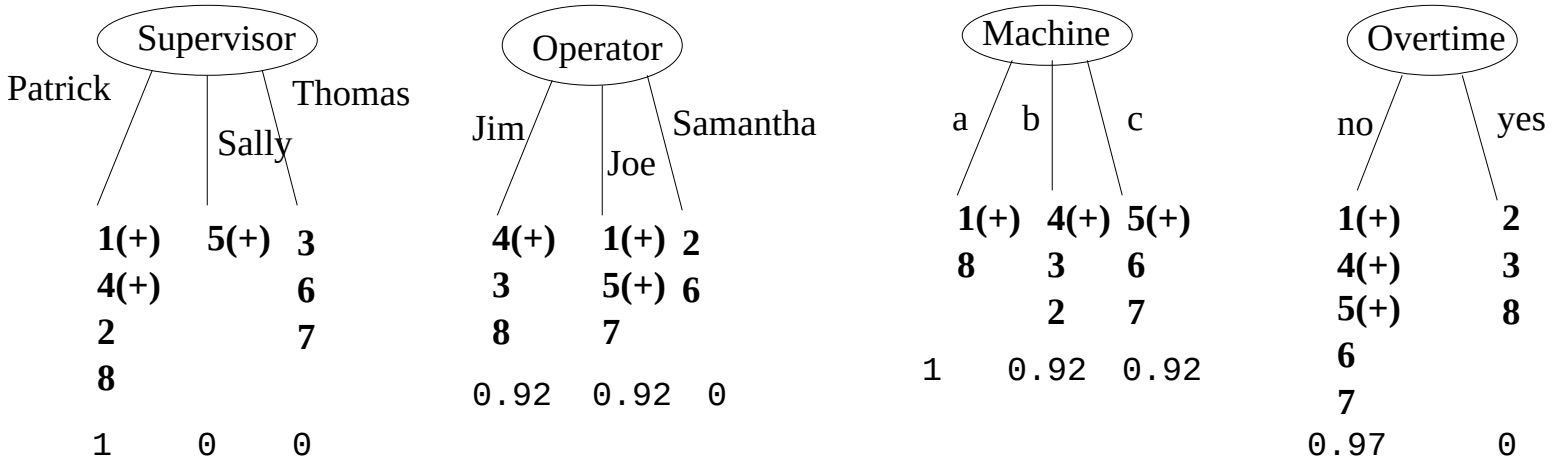
We first identify the attribute with the lowest information remainder by using the whole table as our training set (the positive examples are those with high output).

Since for each attribute A

$$\begin{aligned} Remainder(A) &= \sum_{i=1}^{m} \frac{p_i + n_i}{p + n} \, I(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}) \\ &= \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} \, (-\frac{p_i}{p_i + n_i} \log_2 \frac{p_i}{p_i + n_i} - \frac{n_i}{p_i + n_i} \log_2 \frac{n_i}{p_i + n_i}) \end{aligned}$$

we need to compute all the relative frequencies involved.

Here is how each attribute splits the training set, together with the information content for each branch.

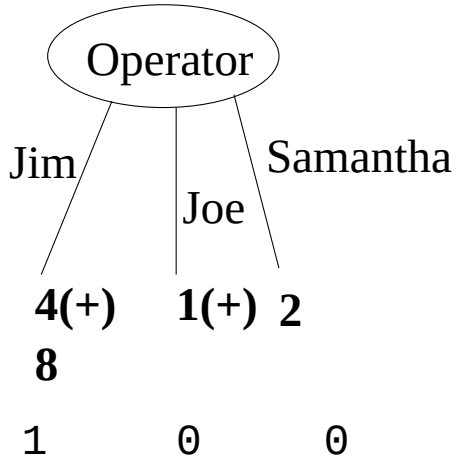


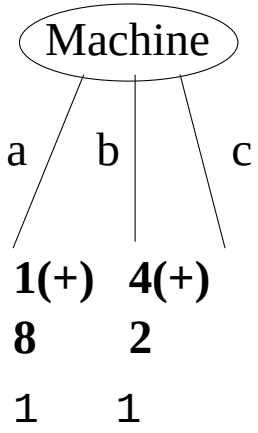
Then,

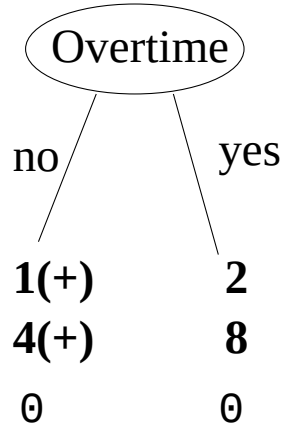
$$
\begin{aligned}
Remainder(Supervisor) &= \tfrac{4}{8} \times 1 + \tfrac{1}{8} \times 0 + \tfrac{3}{8} \times 0 &&= 0.50 \\
Remainder(Operator) &= \tfrac{3}{8} \times 0.92 + \tfrac{3}{8} \times 0.92 + \tfrac{2}{8} \times 0 &&= 0.69 \\
Remainder(Machine) &= \tfrac{2}{8} \times 1 + \tfrac{3}{8} \times 0.92 + \tfrac{3}{8} \times 0.92 &&= 0.94 \\
Remainder(Overtime) &= \tfrac{5}{8} \times 0.97 + \tfrac{3}{8} \times 0 &&= 0.61
\end{aligned}
$$

Since *Supervisor* has the lowest remainder, we choose it as the root of the tree.

Thomas' runs are all negative and Sally's are all positive. We need to further classify just Patrick's runs. So, we recompute the remainders of the remaining attributes, but this time based solely on Patrick's runs.





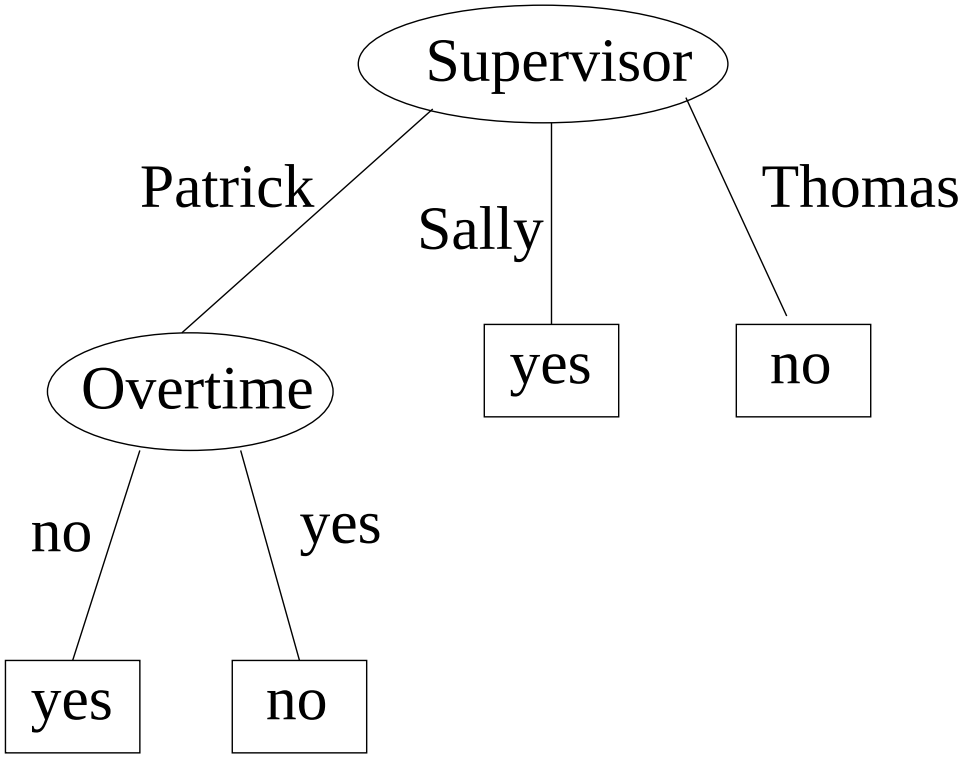


Then,

$$
\begin{aligned}
Remainder(Operator) &= \tfrac{2}{4} \times 1 + \tfrac{1}{4} \times 0 + \tfrac{1}{4} \times 0 = 0.5 \\
Remainder(Machine) &= \tfrac{2}{4} \times 1 + \tfrac{2}{4} \times 1 = 1 \\
Remainder(Overtime) &= \tfrac{2}{4} \times 0 + \tfrac{2}{4} \times 0 = 0
\end{aligned}
$$

Since *Overtime* has lowest the remainder, we choose it to further classify Patrick's runs.

The Final Decision Tree



Problems in Building Decision Trees

- **Noise.** Two training examples may have identical values for all the attributes but be classified differently.
- **Overfitting.** Irrelevant attributes may make spurious distinctions among training examples.
- **Missing data.** The value of some attributes of some training examples may be missing.
- **Multi-valued attributes.** The information gain of an attribute with many different values tends to be non-zero even when the attribute is irrelevant.
- **Continuous-valued attributes.** They must be discretized to be used. Of all the possible discretizations, some are better than others for classification purposes.