CS:4420 Artificial Intelligence

Spring 2019

Learning from Examples

Cesare Tinelli

The University of Iowa

[a] These notes were originally developed by Stuart Russell and are used with permission. They are copyrighted material and may not be used in other course settings outside of the University of Iowa in their current or modified form without the express written consent of the copyright holders.

Readings

- Chap. 18 of [Russell and Norvig, 3rd Edition]

Learning Agents

A distinct feature of intelligent agents in nature is their ability to learn from experience

Using his experience and its internal knowledge, a learning agent is able to produce new knowledge

That is, given its 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

With logical agents, learning can be formalized as follows.

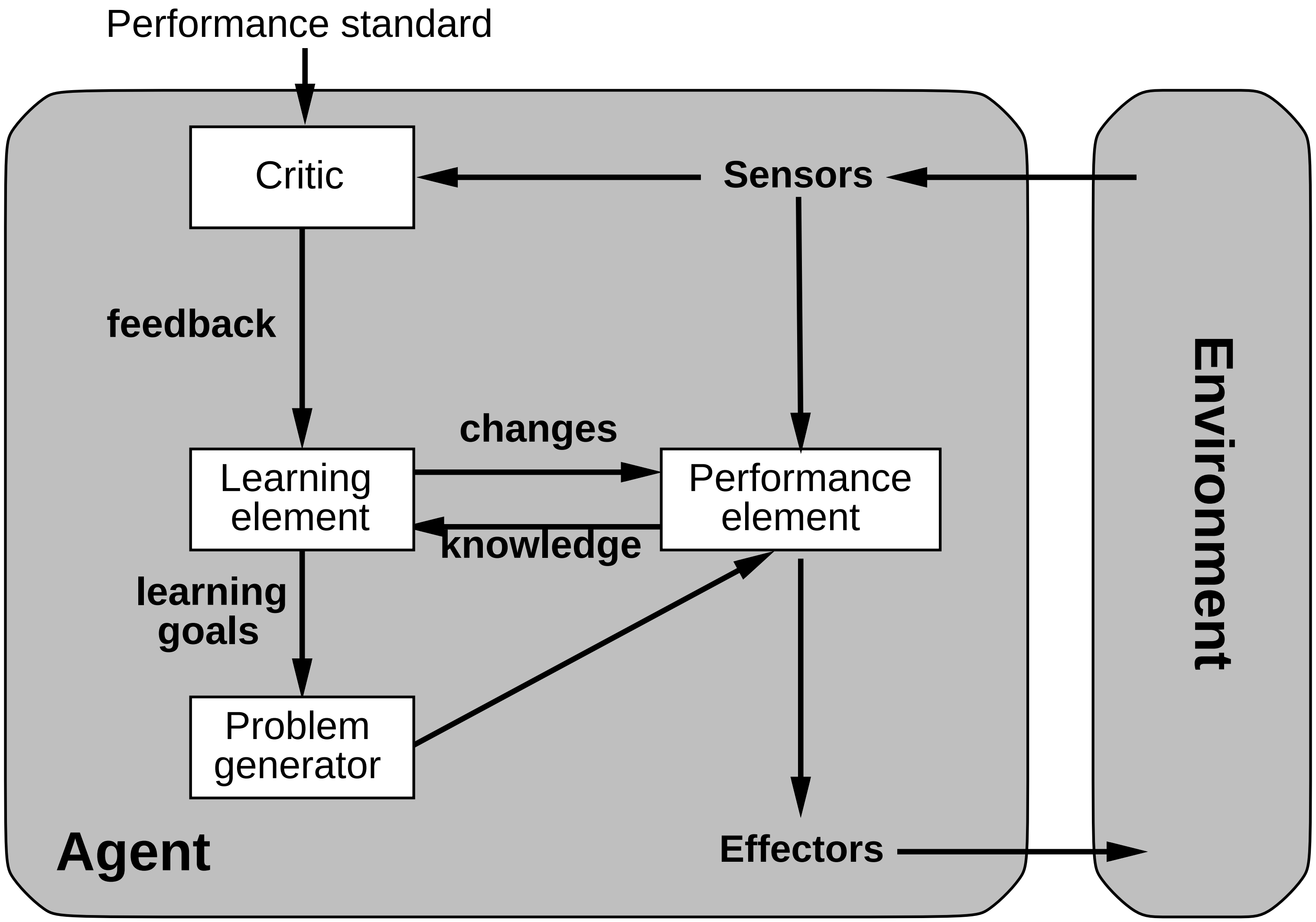
Let Γ, Δ be sets of sentences where

- Γ is the agent's knowledge base, its *current knowledge*
- Δ is a representation of a percept sequence, the *evidential data*

A learning agent is 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

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

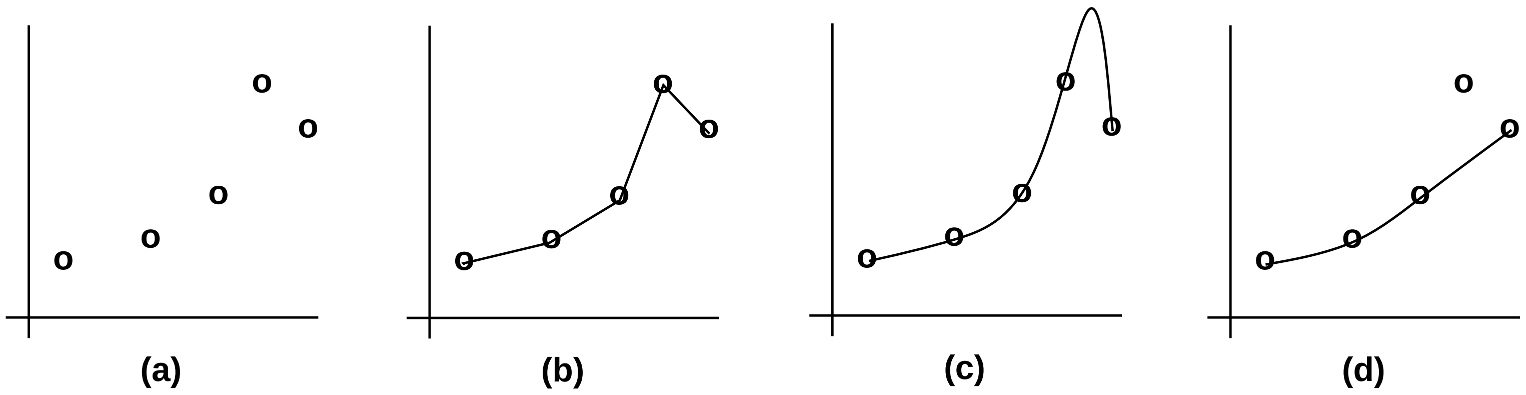
Examples:

1. 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
2. Your grandfather watches a soccer match for the first time and from the action and the commentators' words 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

A simple yet effective form of learning from examples

A *decision tree* is a function that

- maps objects with a certain set of discrete attributes to
- discrete values based on the values of those attributes

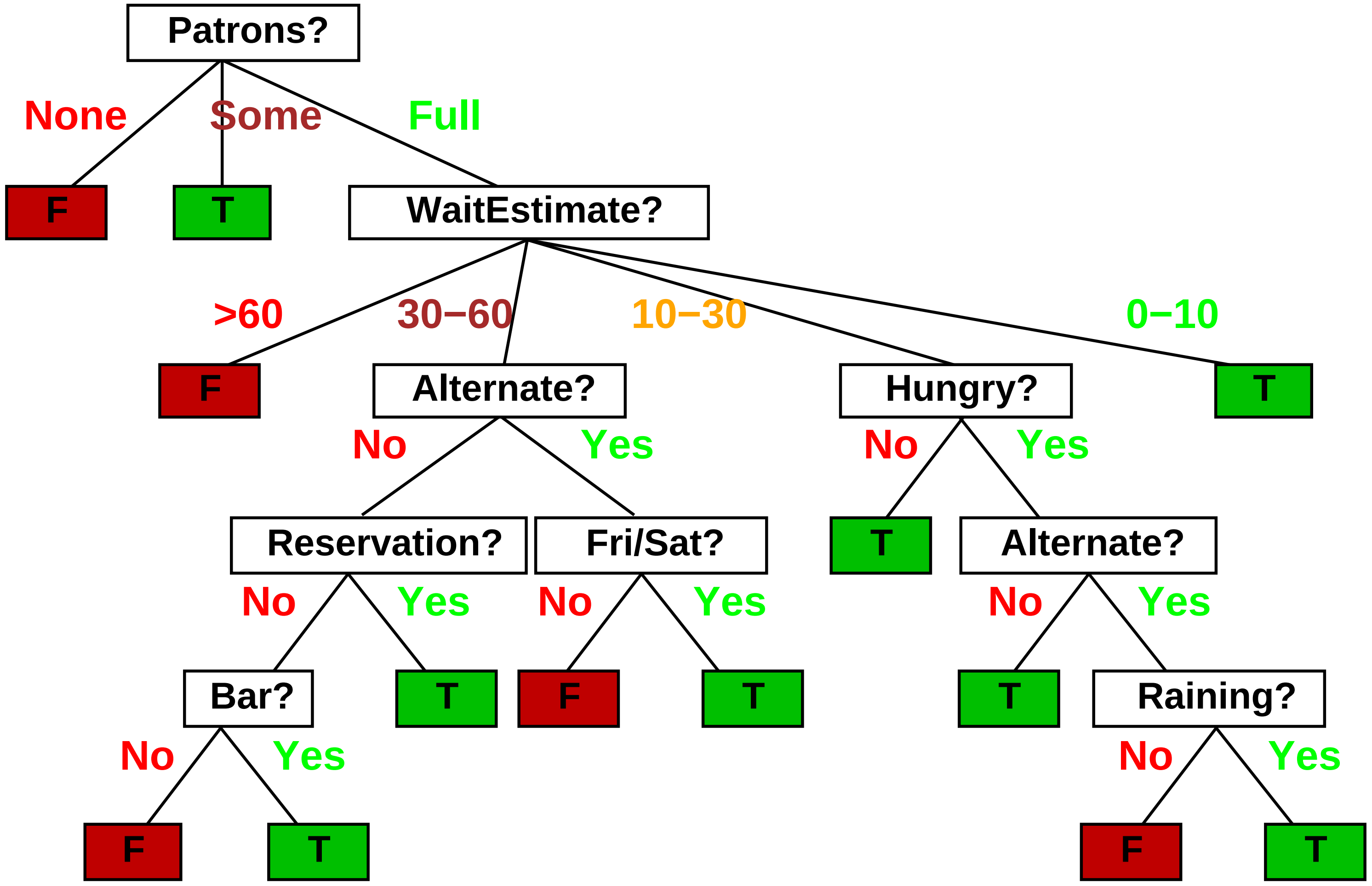
It is representable as a tree in which

- every non-leaf node corresponds to a test on the value of one of the attributes
- every leaf node specifies the value to be returned if that leaf is reached

A decision tree based on attributes $A_1, \ldots, A_n$ acts as *classifier* for objects that have those attributes

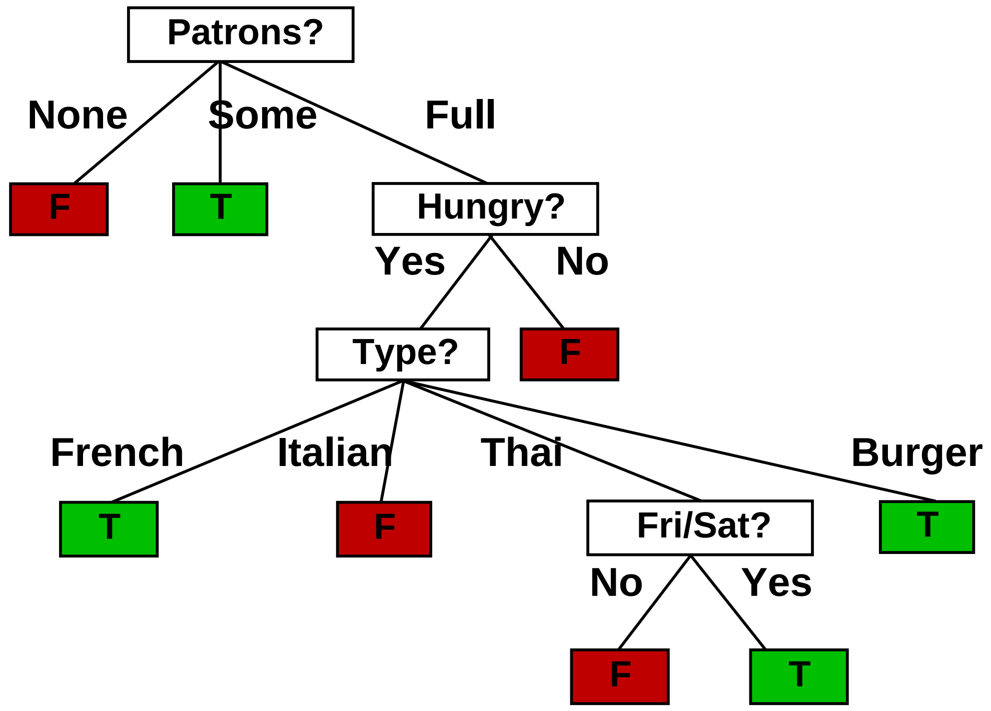
A Decision Tree

This tree can be used to decide whether to wait for a table at a restaurant



A Decision Tree as Predicates

A decision tree with Boolean output defines a logical predicate



$$\begin{aligned} WillWait &\Leftrightarrow && Patrons = Some \\ &\vee && Patrons = Full \wedge \neg Hungry \wedge Type = French \\ &\vee && Patrons = Full \wedge \neg Hungry \wedge Type = Burger \\ &\vee && Patrons = Full \wedge \neg Hungry \wedge Type = Thay \wedge isFriSat \end{aligned}$$

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

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 Boolean decision tree is mean to implement a *Boolean 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?)

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

1. extrapolates a common pattern from the examples
2. correctly classifies 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: prefer the simplest description, i.e., the smallest tree

Problem: searching through the space of possible trees and finding the smallest one takes exponential time

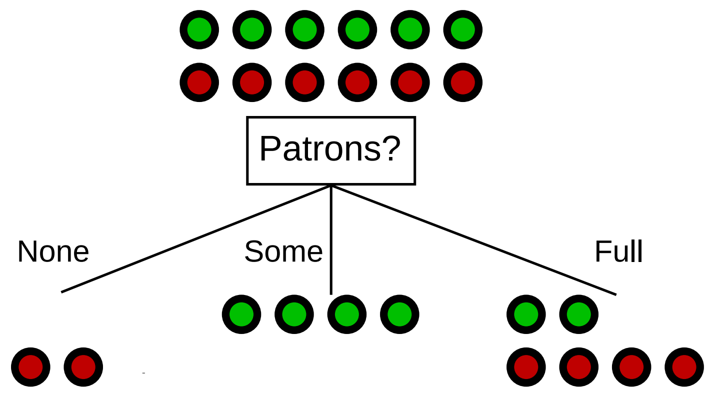
Solution: apply some simple heuristics that quickly lead to small (if not smallest) trees

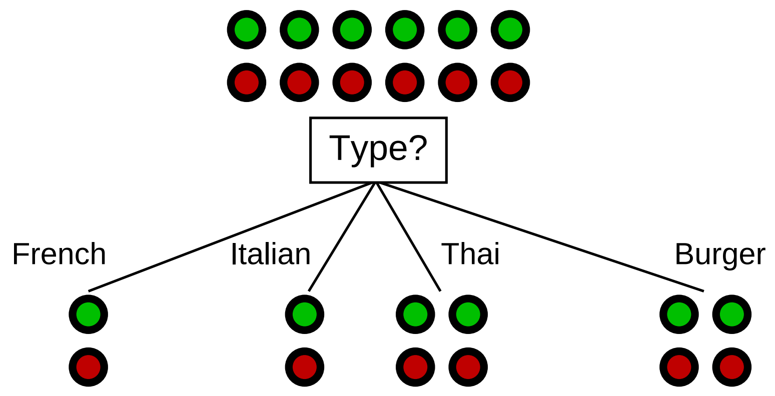
Tradeoff: learning speed and tree size vs. classification accuracy

Main Idea: start building the tree by testing at its root an attribute that better splits the training set into homogeneous classes

Choosing an attribute

A good attribute splits the examples into subsets that are ideally all positive or all negative

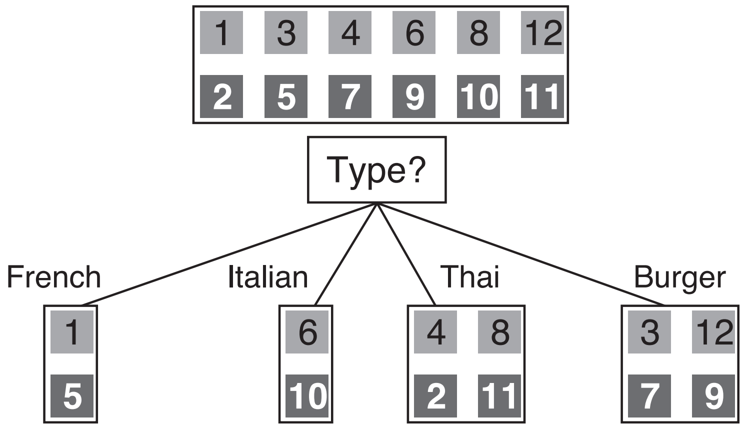




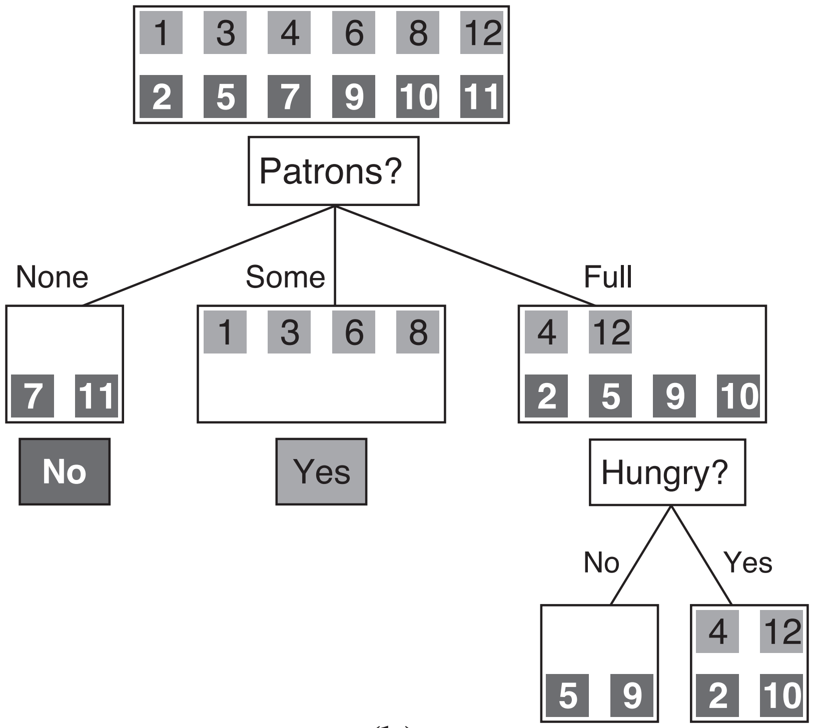
Patrons is a better choice: it gives more information about the classification

Choosing an attribute

Preferring more informative attributes leads to smaller trees



(a)



(b)

Building the Tree: General Procedure

1. Choose for the root node 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 $E_1, \ldots, E_n$. Add a branch i to the root node for each set E_i
3. For each branch i:
   (a) If E_i contains only positive examples, add a yes leaf to the branch
   (b) If E_i contains only negative examples, add a no leaf to the branch
   (c) If E_i is empty, chose the most common yes/no classification among E's examples and add a corresponding leaf to the branch
   (d) Otherwise, add a non-leaf node to the branch and apply the procedure recursively to that node with the remaining attributes and 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 every attribute splits the training set into non-homogeneous classes?

Which one is better?

Information Theory can help us chosing

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*, or *entropy*, of messages/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 prior probability $P(v_i)$, the *information content* or *entropy* H of the event's actual outcome is

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

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

Information Content/Entropy

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

Examples

1) Entropy of fair coin toss:

$$H(P(h), P(t)) = H(\tfrac{1}{2}, \tfrac{1}{2}) = -\tfrac{1}{2} \log_2 \tfrac{1}{2} - \tfrac{1}{2} \log_2 \tfrac{1}{2} = \tfrac{1}{2} + \tfrac{1}{2} = 1 \text{ bit}$$

2) Entropy of a loaded coin toss where $P(head) = 0.99$:

$$\begin{aligned} H(P(h), P(t)) &= H(\tfrac{99}{100}, \tfrac{1}{100}) = -0.99 \log_2 0.99 - 0.01 \log_2 0.01 \\ &\approx 0.08 \text{ bits} \end{aligned}$$

3) Entropy of coin toss for a coin with heads on both sides:

$$H(P(h), P(t)) = H(1, 0) = -1 \log_2 1 - 0 \log_2 0 = 0 - 0 = 0 \text{ bits}$$

Entropy of a Decision Tree

For decision trees, the event is question is whether the tree will return “yes” or “no” for a given input example e

Assume the training set E is a *representative sample* of the domain

Then, the relative frequency of positive examples in E closely approximates the prior probability of a positive example

Entropy of a Decision Tree

For decision trees, the event is question is whether the tree will return "yes" or "no" for a given input example e

Assume the training set E is a *representative sample* of the domain

Then, the relative frequency of positive examples in E closely approximates the prior probability of a positive example

If E contains p positive examples and n negative examples, the probability distribution of answers by a correct decision tree is:

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

Entropy of a Decision Tree

For decision trees, the event is question is whether the tree will return "yes" or "no" for a given input example e

Assume the training set E is a *representative sample* of the domain

Then, the relative frequency of positive examples in E closely approximates the prior probability of a positive example

If E contains p positive examples and n negative examples, the probability distribution of answers by a correct decision tree is:

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

Entropy of correct decision tree:

$$H\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information Content of an Attribute

Checking the value of a single attribute A in the tree provides only some of the information provided by the whole tree

But we can measure how much information is still needed after A has been checked

Information Content of an Attribute

Let $E_1, \ldots, E_m$ be the sets into which A partitions the current training set E

For $i = 1, \ldots, m$, 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, over all branches i of node A we will need

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

extra bits of information to classify the input example after we have checked 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

Problem: From 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

First identify the attribute with the lowest information remainder by using the whole table as the training set

(the positive examples are those with high output)

Since for each attribute A

$$Remainder(A)$$

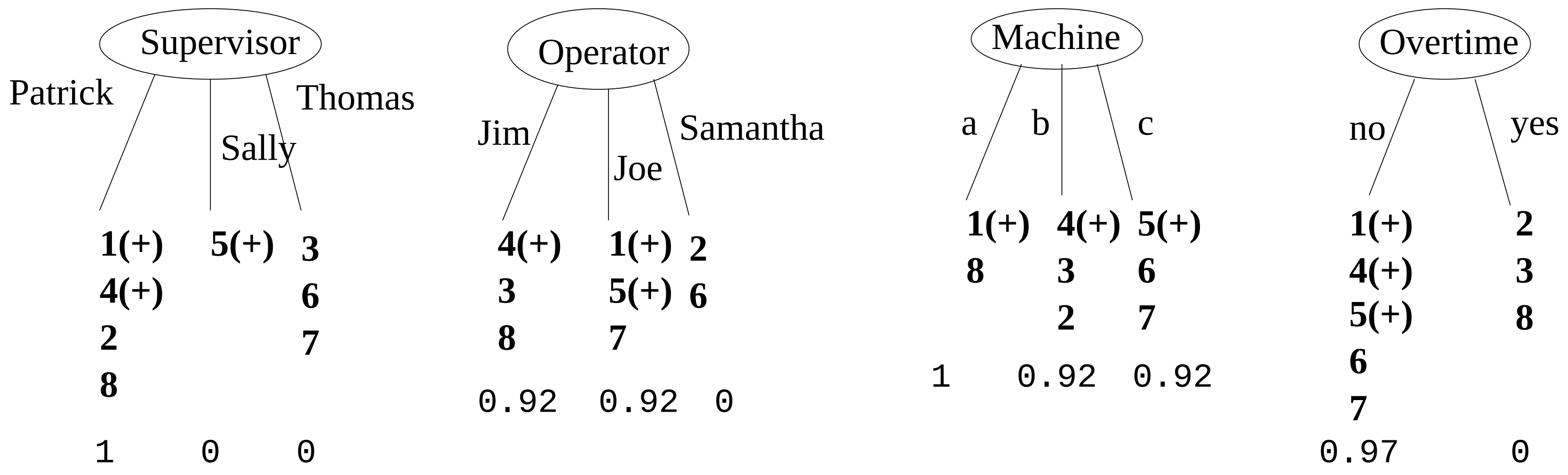
$$= \sum_{i=1}^{m} \frac{p_i+n_i}{p+n} H(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$$

$$= \sum_{i=1}^{n} \frac{p_i+n_i}{p+n} (-\frac{p_i}{p_i+n_i} \log_2 \frac{p_i}{p_i+n_i} - \frac{n_i}{p_i+n_i} \log_2 \frac{n_i}{p_i+n_i})$$

we need to compute all the relative frequencies involved

Example (1)

Here is how each attribute splits the training set, together with the entropy each branch

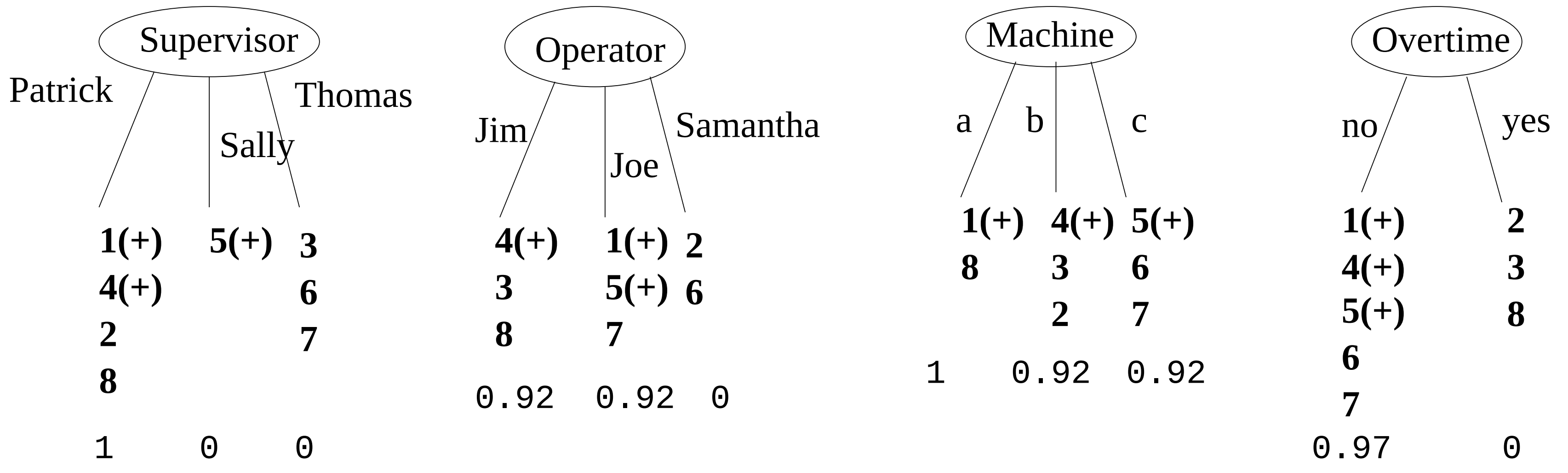


$$
\begin{aligned}
\mathit{Remainder}(\mathit{Supervisor}) &= \tfrac{4}{8} \times 1 + \tfrac{1}{8} \times 0 + \tfrac{3}{8} \times 0 &= 0.50\\
\mathit{Remainder}(\mathit{Operator}) &= \tfrac{3}{8} \times 0.92 + \tfrac{3}{8} \times 0.92 + \tfrac{2}{8} \times 0 &= 0.69\\
\mathit{Remainder}(\mathit{Machine})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

Choose Supervisor since it has the smallest remainder

Example (2)

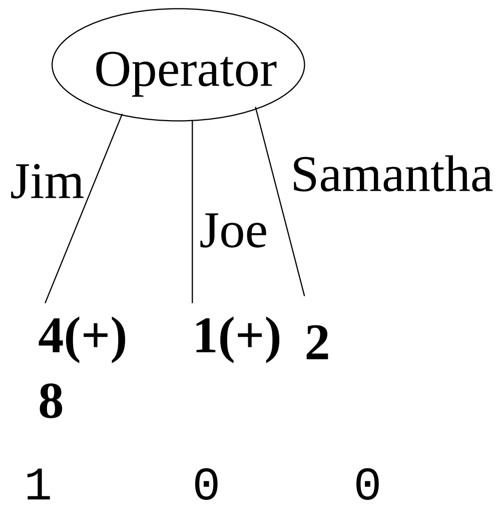
Thomas' runs are all negative and Sally's are all positive

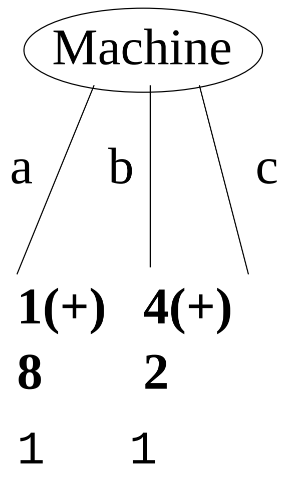


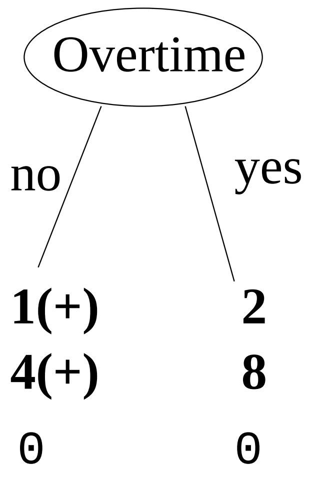
We need to further classify just Patrick's runs

Example (2)

Recompute the remainders of the remaining attributes, but this time based solely on Patrick's runs





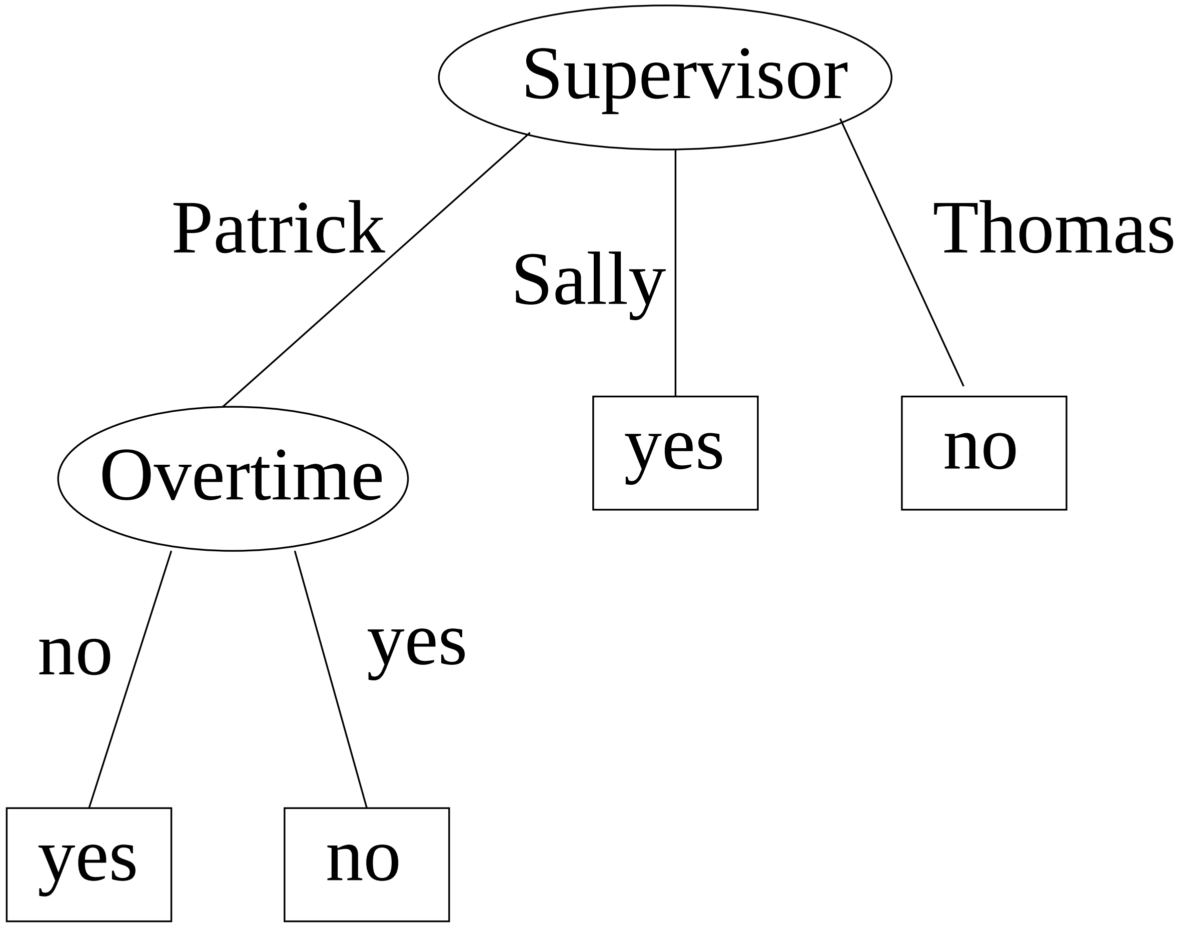


$$
\begin{aligned}
Remainder(Operator) &= \tfrac{2}{4} \times 1 + \tfrac{1}{4} \times 0 + \tfrac{1}{4} \times 0 = 0.5 \\
Remainder(Machine) &= \tfrac{2}{4} \times 1 + \tfrac{2}{4} \times 1 = 1 \\
Remainder(Overtime) &= \tfrac{2}{4} \times 0 + \tfrac{2}{4} \times 0 = 0
\end{aligned}
$$

Choose Overtime to further classify Patrick's runs

Example (3)

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

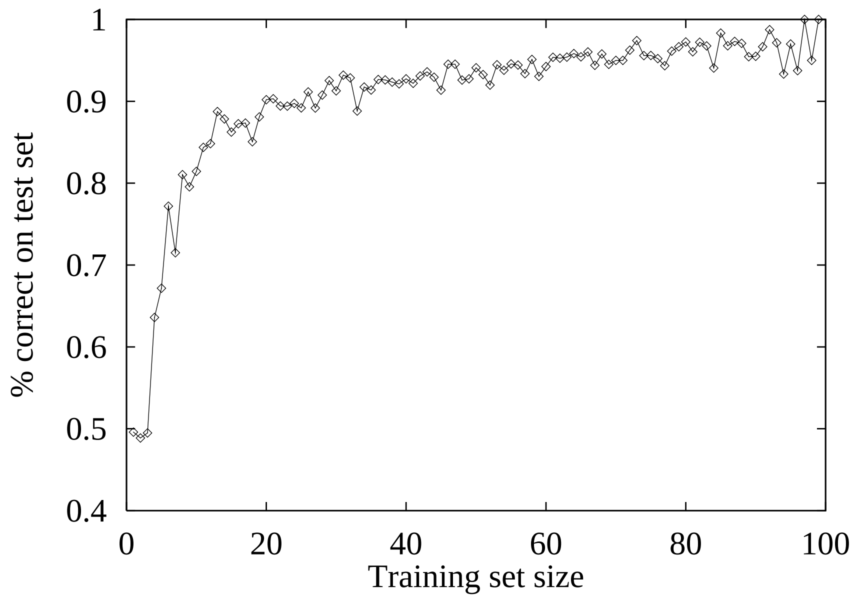
Continuous-valued attributes. They must be discretized to be used. Of all the possible discretizations, some are better than others for classification purposes.

Performance measurement

How do we know that the learned hypothesis h approximates the intended function f?

- Use theorems of computational/statistical learning theory
- Try h on a new *test set* of examples, using same distribution over example space as *training set*

Learning curve = % correct on test set as a function of training set size



- 100 randomly-generated restaurant examples
- graph averaged over 20 trials
- for $i = 1, \ldots, 99$, each trial selects i examples randomly

Choosing the best hypothesis

Consider a set $S = \{(x, y) \mid y = f(x)\}$ of N input/output examples for a target function f

Stationarity assumption: All examples $E \in S$ have the same prior probability distribution $\mathbf{P}(E)$ and each of them is independent from the previously observed ones

Error rate of an hypothesis h: $\frac{|\{(x,y) \mid (x,y) \in S,\ h(x) \neq y\}|}{N}$

Holdout cross-validation: Partions S randomly into a *training* set and a *test* set.

k-fold cross-validation: Partions S into k subsets $S_1, \ldots, S_n$ of the same size. For each $i = 1, \ldots, k$, use S_i as the test set and $S \setminus Si$ as the training set. Use the average error rate