Outline

- Learning agents
- Inductive learning
- Decision tree learning
- Measuring learning performance
Learning

- Learning modifies the agent’s decision mechanisms to improve performance
- Learning is essential for unknown or uncertain environments,
  - i.e., when designer lacks omniscience
- Learning is useful as a *system-construction* method
  - i.e., expose the agent to reality rather than trying to write it down
  - E.g. neural nets, where learning is essential
Learning agents: General architecture

- Agent
  - Environment
  - Sensors
  - Effectors
  - Performance element
- Changes
- Knowledge learning
- Goals
- Problem generator
- Feedback
- Critic
- Performance standard
- Experiments
- Performance element is what we have called the “agent”
- Critic/LearningElement/ProblemGenerator do the “improving”
- Performance standard is fixed
- Learning may require experimentation - actions an agent might not normally consider.
Learning element

- Design of learning element is dictated by
  - what type of performance element is used
  - which functional component is to be learned
  - how that functional component is represented
  - what kind of feedback is available

<table>
<thead>
<tr>
<th>Perf. element</th>
<th>Component</th>
<th>Representation</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>α/β</td>
<td>search</td>
<td>Eval. function</td>
<td>Weighted linear Win/Loss function</td>
</tr>
<tr>
<td>Logical agent</td>
<td>Transition</td>
<td>Actions</td>
<td>Outcome</td>
</tr>
<tr>
<td>Reflex agent</td>
<td>Percept-action</td>
<td>Neural net</td>
<td>Correct action</td>
</tr>
</tbody>
</table>
Learning element

- Design of learning element is dictated by
  - what type of performance element is used
  - which functional component is to be learned
  - how that functional component is represented
  - what kind of feedback is available

- Example scenarios:

<table>
<thead>
<tr>
<th>Perf. element</th>
<th>Component</th>
<th>Representation</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>α/β search</td>
<td>Eval. function</td>
<td>Weighted linear function</td>
<td>Win/Loss</td>
</tr>
</tbody>
</table>
Learning element

- Design of learning element is dictated by
  - what type of performance element is used
  - which functional component is to be learned
  - how that functional component is represented
  - what kind of feedback is available

- Example scenarios:

<table>
<thead>
<tr>
<th>Perf. element</th>
<th>Component</th>
<th>Representation</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha/\beta$ search</td>
<td>Eval. function</td>
<td>Weighted linear function</td>
<td>Win/Loss</td>
</tr>
<tr>
<td>Logical agent</td>
<td>Transition model</td>
<td>Actions</td>
<td>Outcome</td>
</tr>
</tbody>
</table>
Learning element

- Design of learning element is dictated by
  - what type of performance element is used
  - which functional component is to be learned
  - how that functional component is represented
  - what kind of feedback is available

- Example scenarios:

<table>
<thead>
<tr>
<th>Perf. element</th>
<th>Component</th>
<th>Representation</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha/\beta$ search</td>
<td>Eval. function</td>
<td>Weighted linear function</td>
<td>Win/Loss</td>
</tr>
<tr>
<td>Logical agent</td>
<td>Transition model</td>
<td>Actions</td>
<td>Outcome</td>
</tr>
<tr>
<td>Reflex agent</td>
<td>Percept-action function</td>
<td>Neural net</td>
<td>Correct action</td>
</tr>
</tbody>
</table>
Learning from Observations: Types of Feedback

- **Unsupervised learning**:  
  - Agent learns patterns in the input; no explicit feedback  
  - Example: clustering techniques.
Learning from Observations: Types of Feedback

- **Unsupervised learning**:
  - Agent learns patterns in the input; no explicit feedback
  - Example: clustering techniques.

- **Reinforcement learning**:
  - Agent given a series of “rewards” and “punishments”
  - Problem: Given a sequence of actions leading to a desired outcome (e.g. winning a game, getting a tip), which actions were responsible?
Learning from Observations: Types of Feedback

- **Unsupervised learning:**
  - Agent learns patterns in the input; no explicit feedback
  - Example: clustering techniques.

- **Reinforcement learning:**
  - Agent given a series of “rewards” and “punishments”
  - Problem: Given a sequence of actions leading to a desired outcome (e.g. winning a game, getting a tip), which actions were responsible?

- **Supervised learning:**
  - Agent is given examples and their classification
  - Requires a “teacher”
  - Early example: Winston, with examples and “near misses”.
Inductive Learning

- Induction is basically what science does.
Inductive Learning

- Induction is basically what science does.
- Simplest form: learn a function from examples
  - \( f \) is the target function
  - An example is a pair \((x, f(x))\)
    - E.g.: (customer info, good/bad credit risk?)
Inductive Learning

- Induction is basically what science does.
- Simplest form: learn a \textit{function} from \textit{examples}
  - $f$ is the \textit{target function}
  - An \textit{example} is a pair $(x, f(x))$
    - E.g.: (customer info, good/bad credit risk?)
- Problem formulation:
  - given a \textit{training set} of examples
  - find a \textit{hypothesis} $h$ such that $h \approx f$
    - $h$ is the “best guess” of $f$
Inductive Learning

- Induction is basically what science does.
- Simplest form: learn a function from examples
  - $f$ is the target function
  - An example is a pair $(x, f(x))$
    - E.g.: (customer info, good/bad credit risk?)
- Problem formulation:
  - given a training set of examples
  - find a hypothesis $h$ such that $h \approx f$
    \[ h \text{ is the “best guess” of } f \]
- This is a highly simplified model of real learning:
  - ignores prior knowledge
  - assumes a deterministic, observable “environment”
  - assumes examples are given
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- E.g., curve fitting:

$$f(x)$$

- Q: What’s an appropriate function to fit to the known points?
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- E.g., curve fitting:

\[ f(x) \]

- A straight line is the simplest curve.
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- E.g., curve fitting:

  ![Graph showing curve fitting]

  - A nonlinear (e.g. polynomial) function will give a better fit.
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- E.g., curve fitting:

  A nonlinear (e.g. polynomial) function will give a better fit.
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- E.g., curve fitting:

\[
\begin{align*}
f(x)
\end{align*}
\]

- In fact, any number of functions will fit the data.
- Q: How do we choose between different hypotheses that are consistent with the data?
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- E.g., curve fitting:

\[ f(x) \]

- **Ockham’s razor**: Choose the *simplest* hypothesis
  - But defining simplicity isn’t easy.
Attribute-Based Learning

- We’ll look at decision tree learning
  - One of the simplest but most successful forms of learning.
  - Idea: Learn $f(\langle attributes\rangle) \rightarrow result$. 

Given:
- A set of attributes (e.g. salary, debt, etc.) and an attribute for classification (e.g. credit risk is high, medium, low).
- A set of training instances: attributes + outcome.

Construct a tree where:
- The root is the attribute that discriminates best among the outcomes.
  - I.e. the root is the best predictor of the outcome.
- Descendants are the successive attributes that best discriminate among the remaining instances.

A decision tree is a tree where a node is labelled by an attribute and the edges from the node are labelled with the attribute’s values.
Attribute-Based Learning

• We’ll look at decision tree learning
  • One of the simplest but most successful forms of learning.
  • Idea: Learn $f(\langle\text{attributes}\rangle) \rightarrow \text{result}$.

• Given:
  • A set of attributes (e.g. salary, debt, etc.) and an attribute for classification (e.g. credit risk is high, medium, low).
  • A set of training instances: attributes + outcome.
Attribute-Based Learning

• We’ll look at *decision tree learning*
  - One of the simplest but most successful forms of learning.
  - Idea: Learn $f(\langle \text{attributes} \rangle) \rightarrow \text{result}$.

• Given:
  - A set of attributes (e.g. salary, debt, etc.) and an attribute for classification (e.g. credit risk is high, medium, low).
  - A set of training instances: attributes + outcome.

• Construct a tree where:
  - The root is the attribute that discriminates best among the outcomes.
    - I.e. the root is the best *predictor* of the outcome.
  - Descendants are the successive attributes that best discriminate among the remaining instances.
Attribute-Based Learning

• We’ll look at *decision tree learning*
  • One of the simplest but most successful forms of learning.
  • Idea: Learn \( f(\langle attributes\rangle) \rightarrow result \).

• Given:
  • A set of attributes (e.g. salary, debt, etc.) and an attribute for classification (e.g. credit risk is high, medium, low).
  • A set of training instances: attributes + outcome.

• Construct a tree where:
  • The root is the attribute that discriminates best among the outcomes.
    - I.e. the root is the best *predictor* of the outcome.
  • Descendants are the successive attributes that best discriminate among the remaining instances.

**A decision tree** is a tree where a node is labelled by an attribute and the edges from the node are labelled with the attribute’s values.
Decision tree learning

- Examples given by *attribute values* (Boolean, discrete, continuous, etc.)
  - E.g., situations where I will/won’t wait for a restaurant table:

<table>
<thead>
<tr>
<th>Ex.</th>
<th>Alt</th>
<th>Bar</th>
<th>Fri</th>
<th>Hun</th>
<th>Pat</th>
<th>Price</th>
<th>Rain</th>
<th>Res</th>
<th>Type</th>
<th>Est</th>
<th>Target: Wait?</th>
</tr>
</thead>
<tbody>
<tr>
<td>X₁</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₂</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
</tr>
<tr>
<td>X₃</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₄</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
</tr>
<tr>
<td>X₅</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>French</td>
<td>&gt; 60</td>
<td>F</td>
</tr>
<tr>
<td>X₆</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Italian</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₇</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₈</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Some</td>
<td>$</td>
<td>T</td>
<td>T</td>
<td>Thai</td>
<td>0–10</td>
<td>T</td>
</tr>
<tr>
<td>X₉</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt; 60</td>
<td>F</td>
</tr>
<tr>
<td>X₁₀</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$$$</td>
<td>F</td>
<td>T</td>
<td>Italian</td>
<td>10–30</td>
<td>F</td>
</tr>
<tr>
<td>X₁₁</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
</tr>
<tr>
<td>X₁₂</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
</tr>
</tbody>
</table>

- *Classification* of examples is positive (T) or negative (F)
Decision trees

- One possible representation for hypotheses
- E.g., the author’s “personal” tree for deciding whether to wait:

```
Patrons?

None  | Some  | Full
F     | T     |

WaitEstimate?

>60  | 30−60 | 10−30 | 0−10
F     | T     |

Alternate?

No   | Yes
F     | T

Hungry?

No   | Yes
F     | T

Reservation?  | Fri/Sat?

No   | Yes | No | Yes
F     | T   | T   |

Bar?

No | Yes
F | T

Alternate?

No | Yes
F | T

Raining?

No | Yes
F | T
```
Decision trees

- A decision tree reaches its decision by making a series of tests on an example, beginning at the root.
- Note that all attributes may not be used in the tree.
- An attribute may occur in $\geq 1$ place in the tree.
- Different attributes may appear at the same depth on the tree.
- Issue: There are lots of trees that will classify the same data. How to choose a “good” tree?
• Decision trees can express any function of the input attributes.
Expressiveness

- Decision trees can express any function of the input attributes.
  - E.g., for Boolean functions, truth table row $\rightarrow$ path to leaf:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A xor B</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

  ![Decision Tree Diagram]

  - Trivially, there is a consistent DT for any consistent training set, with one path to leaf for each example, but it probably won't generalize to new examples.
  - Prefer to find more compact decision trees.
  - Consider this as a search problem, searching the space of decision trees (next) ...
Expressiveness

- Decision trees can express any function of the input attributes.
  - E.g., for Boolean functions, truth table row → path to leaf:

```
A  B  A xor B
F  F  F
F  T  T
T  F  T
T  T  F
```

- Trivially, there is a consistent DT for any consistent training set, with one path to leaf for each example
  - but it probably won’t generalize to new examples
- Prefer to find more *compact* decision trees
- Consider this as a search problem, searching the space of decision trees (next) ...
Hypothesis spaces

• How many distinct decision trees with $n$ Boolean attributes?

• E.g., with 6 Boolean attributes, there are $18,446,744,073,709,551,616$ trees

• How many purely conjunctive hypotheses (e.g. $\text{Hungry} \land \neg \text{Rain}$)?

• Each attribute can be in, in, or out $3^n$ distinct conjunctive hypotheses

• A more expressive hypothesis space increases chance that target function can be expressed (good!)

• Increases number of hypotheses consistent with training set $\Rightarrow$ makes searching the space harder (bad!)

• $\Rightarrow$ may get worse predictions (bad!)
Hypothesis spaces

- How many distinct decision trees with $n$ Boolean attributes?  
  $= \text{number of Boolean functions}$

- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

- How many purely conjunctive hypotheses (e.g. Hungry $\land \neg$ Rain)?
  - Each attribute can be in (positive), in (negative), or out
  $= 3^n$ distinct conjunctive hypotheses

- A more expressive hypothesis space
  - increases chance that target function can be expressed (good!)
  - increases number of hypotheses consistent with training set

  $\Rightarrow$ makes searching the space harder (bad!)

  $\Rightarrow$ may get worse predictions (bad!)
Hypothesis spaces

- How many distinct decision trees with $n$ Boolean attributes?
  - $=$ number of Boolean functions
  - $=$ number of distinct truth tables with $2^n$ rows

- E.g., with 6 Boolean attributes, there are $18,446,744,073,709,551,616$ trees

- How many purely conjunctive hypotheses (e.g. Hungry $\land \neg$ Rain)?
  - Each attribute can be in (positive), in (negative), or out
  - $3^n$ distinct conjunctive hypotheses

- A more expressive hypothesis space
  - increases chance that target function can be expressed (good!)
  - increases number of hypotheses consistent with training set
    $\Rightarrow$ makes searching the space harder (bad!)
  - $\Rightarrow$ may get worse predictions (bad!)
Hypothesis spaces

- How many distinct decision trees with $n$ Boolean attributes?
  - $= \text{number of Boolean functions}$
  - $= \text{number of distinct truth tables with } 2^n \text{ rows } = 2^{2^n}$

- How many purely conjunctive hypotheses (e.g., Hungry $\land \neg$ Rain)?
  - Each attribute can be in (positive), in (negative), or out
  - $3^n$ distinct conjunctive hypotheses

- A more expressive hypothesis space
  - increases chance that target function can be expressed (good!)
  - increases number of hypotheses consistent with training set
    $\Rightarrow$ makes searching the space harder (bad!)
    $\Rightarrow$ may get worse predictions (bad!)
Hypothesis spaces

• How many distinct decision trees with $n$ Boolean attributes?
  $=$ number of Boolean functions
  $=$ number of distinct truth tables with $2^n$ rows $= 2^{2^n}$

• E.g., with 6 Boolean attributes, there are
  $18,446,744,073,709,551,616$ trees
Hypothesis spaces

• How many distinct decision trees with \( n \) Boolean attributes?
  \( = \) number of Boolean functions
  \( = \) number of distinct truth tables with \( 2^n \) rows \( = 2^{2n} \)

• E.g., with 6 Boolean attributes, there are
  \( 18,446,744,073,709,551,616 \) trees

• How many purely conjunctive hypotheses
  (e.g. \( Hungry \land \neg Rain \))?
Hypothesis spaces

• How many distinct decision trees with \(n\) Boolean attributes?
  \(=\) number of Boolean functions
  \(=\) number of distinct truth tables with \(2^n\) rows \(= 2^{2^n}\)

• E.g., with 6 Boolean attributes, there are
  18,446,744,073,709,551,616 trees

• How many purely conjunctive hypotheses
  (e.g. \(Hungry \land \neg Rain\))?

• Each attribute can be in (positive), in (negative), or out
  \(3^n\) distinct conjunctive hypotheses
Hypothesis spaces

• How many distinct decision trees with \( n \) Boolean attributes?
  \[ \text{number of Boolean functions} = \text{number of distinct truth tables with } 2^n \text{ rows} = 2^{2^n} \]

• E.g., with 6 Boolean attributes, there are
  \[ 18,446,744,073,709,551,616 \] trees

• How many purely conjunctive hypotheses (e.g. \( \text{Hungry} \land \neg \text{Rain} \))?
  \[ 3^n \] distinct conjunctive hypotheses

• A more expressive hypothesis space
  • increases chance that target function can be expressed (good!)
  • increases number of hypotheses consistent with training set
    \[ \Rightarrow \] makes searching the space harder (bad!)
  \[ \Rightarrow \] may get worse predictions (bad!)
Decision tree learning

- **Aim**: Find a *small* tree consistent with the training examples
Decision tree learning

- **Aim**: Find a *small* tree consistent with the training examples
- **Idea**: Incrementally build a decision tree top down by repeatedly splitting the training data.
Decision tree learning

- **Aim**: Find a *small* tree consistent with the training examples
- **Idea**: Incrementally build a decision tree top down by repeatedly splitting the training data.
- **Input** is a target attribute, a set of attributes, and a set of examples. (Also optionally a default value.)
Decision tree learning

- **Aim**: Find a *small* tree consistent with the training examples
- **Idea**: Incrementally build a decision tree top down by repeatedly splitting the training data.
- **Input** is a target attribute, a set of attributes, and a set of examples. (Also optionally a default value.)
- **Stop** if all examples have the same value (or if you run out of examples).
Decision tree learning

- **Aim**: Find a *small* tree consistent with the training examples
- **Idea**: Incrementally build a decision tree top down by repeatedly splitting the training data.
- **Input** is a target attribute, a set of attributes, and a set of examples. (Also optionally a default value.)
- **Stop** if all examples have the same value (or if you run out of examples).
- **Otherwise,**
  - choose an attribute to split on, and
  - for each of that attribute’s values build a subtree for those examples with that attribute value.
Decision tree learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose “best” attribute as root of (sub)tree

Function $\text{DTL}(\text{examples, attributes, default})$ returns a decision tree

1. if examples is empty then return default
2. if all examples have the same classification then return the classification
3. if attributes is empty then return $\text{Mode}(\text{examples})$
4. $\text{best} \leftarrow \text{Choose-Attribute}(\text{attributes, examples})$
5. $\text{tree} \leftarrow$ a new decision tree with root test $\text{test } \text{best}$
6. for each value $v_i$ of $\text{best}$ do
   a. $\text{examples}_i \leftarrow \{\text{elements of examples with } \text{best} = v_i\}$
   b. $\text{subtree} \leftarrow \text{DTL}(\text{examples}_i, \text{attributes } \text{–} \text{best, Mode( examples))}$
   c. add a branch to $\text{tree}$ with label $v_i$ and subtree $\text{subtree}$
7. return $\text{tree}$
Choosing an attribute

- **Idea**: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”
- Below, *Patrons* is a better choice since it gives *information* about the classification.
Choosing an attribute

- **Idea**: A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”
- Below, *Patrons* is a better choice since it gives *information* about the classification

```
<table>
<thead>
<tr>
<th>Type?</th>
<th>French</th>
<th>Italian</th>
<th>Thai</th>
<th>Burger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patrons?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>None</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Some</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Full</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- **Problem**: How to formalise this?
Information

- **Information** answers questions
- I.e. the more clueless I am about an answer initially, the more information is contained in the answer
Information

- *Information* answers questions
- I.e. the more clueless I am about an answer initially, the more information is contained in the answer
- *Information theory* measures information content in *bits*.
- Scale:
  - 1 bit = answer to Boolean question with prior \(\langle 0.5, 0.5 \rangle\)
- I.e. 1 bit of information is enough to answer a yes/no question about which one has no idea.
• *Information* answers questions
• I.e. the more clueless I am about an answer initially, the more information is contained in the answer
• *Information theory* measures information content in *bits*.
• Scale:
  
  - 1 bit = answer to Boolean question with prior $\langle 0.5, 0.5 \rangle$
• I.e. 1 bit of information is enough to answer a yes/no question about which one has no idea.
• Or (the same thing):
  
  - 1 bit lets you distinguish 2 items.
  - 2 bits lets you distinguish 4 items;
  - $n$ bits lets you distinguish $2^n$ items.
The information in an answer when the prior is $\langle P_1, \ldots, P_n \rangle$ is

$$I(\langle P_1, \ldots, P_n \rangle) = - \sum_{i=1}^{n} P_i \log_2 P_i$$

(also called entropy of the prior)

Entropy is a measure of the uncertainty of a random variable.

A RV with prior $\langle 0.5, 0.5 \rangle$ has lower entropy than one with $\langle 0.25, 0.25, 0.25, 0.25 \rangle$.

A RV with prior $\langle 1.0, 0.0 \rangle$ has no uncertainty

• Hence it has entropy of 0.
Information

- The *information* in an answer when the prior is \( \langle P_1, \ldots, P_n \rangle \) is

\[
I(\langle P_1, \ldots, P_n \rangle) = - \sum_{i=1}^{n} P_i \log_2 P_i
\]

(also called *entropy* of the prior)

- *Entropy* is a measure of the uncertainty of a random variable.

- A RV with prior \( \langle 0.5, 0.5 \rangle \) has lower entropy than one with \( \langle 0.25, 0.25, 0.25, 0.25 \rangle \).

- A RV with prior \( \langle 1.0, 0.0 \rangle \) has no uncertainty
  - Hence it has entropy of 0.

- RV with prior \( \langle 0.5, 0.5 \rangle \) has entropy 1.0.
Information

- The *information* in an answer when the prior is $\langle P_1, \ldots, P_n \rangle$ is
  \[
  I(\langle P_1, \ldots, P_n \rangle) = - \sum_{i=1}^{n} P_i \log_2 P_i
  \]
  (also called *entropy* of the prior)
- *Entropy* is a measure of the uncertainty of a random variable.
- A RV with prior $\langle 0.5, 0.5 \rangle$ has lower entropy than one with $\langle 0.25, 0.25, 0.25, 0.25 \rangle$.
- A RV with prior $\langle 1.0, 0.0 \rangle$ has no uncertainty
  - Hence it has entropy of 0.
- RV with prior $\langle 0.5, 0.5 \rangle$ has entropy 1.0.
  RV with prior $\langle 0.25, 0.25, 0.25, 0.25 \rangle$ has entropy 2.0.
Information

- The *information* in an answer when the prior is $\langle P_1, \ldots, P_n \rangle$ is
  \[ I(\langle P_1, \ldots, P_n \rangle) = -\sum_{i=1}^{n} P_i \log_2 P_i \]
  (also called *entropy* of the prior)

- *Entropy* is a measure of the uncertainty of a random variable.

- A RV with prior $\langle 0.5, 0.5 \rangle$ has lower entropy than one with $\langle 0.25, 0.25, 0.25, 0.25 \rangle$.

- A RV with prior $\langle 1.0, 0.0 \rangle$ has no uncertainty
  - Hence it has entropy of 0.

- RV with prior $\langle 0.5, 0.5 \rangle$ has entropy 1.0.
  RV with prior $\langle 0.25, 0.25, 0.25, 0.25 \rangle$ has entropy 2.0.
  RV with prior $\langle 0.95, 0.05 \rangle$ has entropy about .08.
Using Information Theory to Choose an Attribute

• Suppose we have $p$ +ve and $n$ −ve examples at the root
Using Information Theory to Choose an Attribute

- Suppose we have $p$ +ve and $n$ −ve examples at the root
- This is an estimate of the probabilities of the possible answers.
Using Information Theory to Choose an Attribute

• Suppose we have $p$ +ve and $n$ –ve examples at the root
• This is an estimate of the probabilities of the possible answers.
• $I(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle)$ bits needed to classify a new example
Using Information Theory to Choose an Attribute

- Suppose we have \( p \) +ve and \( n \) −ve examples at the root
- This is an estimate of the probabilities of the possible answers.
- \( I(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle) \) bits needed to classify a new example
- I.e.:
  \[
  I(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle) = -\frac{p}{p+n} \log_2 \left( \frac{p}{p+n} \right) - \frac{n}{p+n} \log_2 \left( \frac{n}{p+n} \right)
  \]
- E.g., 12 restaurant examples: \( p = n = 6 \) so we need 1 bit
Information Gain

- An attribute $A$ with $n$ values splits the training set $E$ into disjoint subsets $E_1, \ldots, E_n$, according to their values for $A$. 

- Information Gain (IG) or reduction in entropy from the attribute test:

\[
IG(A) = I\left(\frac{p_i(p_i + n_i), n_i(p_i + n_i)}\right) - \text{Remainder}(A)
\]

- Choose the attribute with the largest information gain.
Information Gain

- An attribute $A$ with $n$ values splits the training set $E$ into disjoint subsets $E_1, \ldots, E_n$, according to their values for $A$.
- Let $E_i$ have $p_i$ positive and $n_i$ negative examples.

Information Gain (IG) or reduction in entropy from the attribute test:

$$\text{IG}(A) = I(\langle p_i(p_i + n_i) \rangle, n_i(p_i + n_i)) - \text{Remainder}(A)$$

Choose the attribute with the largest information gain.
Information Gain

- An attribute $A$ with $n$ values splits the training set $E$ into disjoint subsets $E_1, \ldots, E_n$, according to their values for $A$.
- Let $E_i$ have $p_i$ positive and $n_i$ negative examples
  - $I(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle)$ bits needed to classify a new example
Information Gain

- An attribute $A$ with $n$ values splits the training set $E$ into disjoint subsets $E_1, \ldots, E_n$, according to their values for $A$.
- Let $E_i$ have $p_i$ positive and $n_i$ negative examples
  - $I(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle)$ bits needed to classify a new example
- The expected number of bits per example over all branches is
  \[
  \text{Remainder}(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} \cdot I\left(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle\right)
  \]
Information Gain

- An attribute $A$ with $n$ values splits the training set $E$ into disjoint subsets $E_1, \ldots, E_n$, according to their values for $A$.
- Let $E_i$ have $p_i$ positive and $n_i$ negative examples
  - $I(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle)$ bits needed to classify a new example
- The *expected* number of bits per example over all branches is

$$\text{Remainder}(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} I\left(\left\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \right\rangle\right)$$

- **Information Gain** (IG) or *reduction in entropy* from the attribute test:

$$\text{IG}(A) = I\left(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle\right) - \text{Remainder}(A)$$
Information Gain

- An attribute $A$ with $n$ values splits the training set $E$ into disjoint subsets $E_1, \ldots, E_n$, according to their values for $A$.
- Let $E_i$ have $p_i$ positive and $n_i$ negative examples
  - $I(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle)$ bits needed to classify a new example
- The expected number of bits per example over all branches is
  \[
  \text{Remainder}(A) = \sum_{i=1}^{n} \frac{p_i + n_i}{p + n} I\left(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle\right)
  \]
- **Information Gain (IG)** or reduction in entropy from the attribute test:
  \[
  IG(A) = I(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle) - \text{Remainder}(A)
  \]
- Choose the attribute with the largest information gain.
Information contd.

- For the training set, \( p = n = 6 \) and \( I(6/12, 6/12) = 1 \) bit.
Information contd.

- For the training set, $p = n = 6$ and $I(6/12, 6/12) = 1$ bit.
- $IG(Patrons) = 1 - 0.459 = .541$
For the training set, $p = n = 6$ and $I(6/12, 6/12) = 1$ bit.

$IG(Patrons) = 1 - 0.459 = .541$
$IG(Type) = 1 - 1 = 0$
• For the training set, \( p = n = 6 \) and \( I(6/12, 6/12) = 1 \) bit.

• \( IG(Patrons) = 1 - 0.459 = 0.541 \)
  \( IG(Type) = 1 - 1 = 0 \)

• \( Patrons \) has the highest IG of all attributes and so is chosen by the DTL algorithm as the root.
• Decision tree learned from the 12 examples:

- Patrons?
  - None: F
  - Some: T
  - Full
    - Hungry?
      - Yes: T
      - No: F
- Type?
  - French: T
  - Italian: F
  - Thai
    - Fri/Sat?
      - No: F
      - Yes: T
  - Burger
Compare:

```
Example contd.

None  Some  Full  >60  30−60  10−30  0−10  No  Yes  No  Yes  No  Yes  No  Yes  No  Yes  No  Yes  No  Yes

vs:

Patrons?  Type?  Fri/Sat?  WaitEstimate?
None  Some  Full  French  Italian  Thai  Burger  No  Yes  No  Yes  No  Yes

F  T  F  T  F  T  F  T
```
Performance Measurement

• How do we know if the result of learning is “reasonable”?

Answer: Test it on other examples

Methodology for assessing performance:
1. Collect a large set of examples.
2. Divide into two sets: a training set and a test set.
3. Apply the learning algorithm to the training set. E.g. generate a decision tree
4. Measure the proportion of examples in the test set that are correctly classified.
5. Repeat the above steps for different sizes of training sets and different training sets for each size.

Intuition: Small training sets will tend to be inaccurate; large training sets will be more work. Try to find a good balance.
Performance Measurement

- How do we know if the result of learning is “reasonable”?  
  - Answer: Test it on other examples
Performance Measurement

• How do we know if the result of learning is “reasonable”?  
  • Answer: Test it on other examples

• Methodology for assessing performance:

  1. Collect a large set of examples.
  2. Divide into two sets: a training set and a test set.
  3. Apply the learning algorithm to the training set.
     • E.g. generate a decision tree
  4. Measure the proportion of examples in the test set that are correctly classified.
  5. Repeat the above steps for different sizes of training sets and different training sets for each size.

• Intuition: Small training sets will tend to be inaccurate; large training sets will be more work.
  • Try to find a good balance.
Performance Measurement

• Plotting the results gives a *learning curve*:

![Graph showing a learning curve](image)

• So big gains when a small training set is increased in size, after which it tapers off.
Performance measurement contd.

- Learning curve depends on
  - *realizable* vs. *non-realizable* learning problem.
    - realizable: can express target function
    - non-realizability can be due to missing attributes or restricted hypothesis class (e.g., thresholded linear function)
  - *redundant* expressiveness (e.g., loads of irrelevant attributes)
Summary

• Learning needed for unknown environments, “lazy” designers
• Learning agent = performance element + learning element
• Learning method depends on type of performance element, available feedback, type of component to be improved, and its representation
• For supervised learning, the aim is to find a simple hypothesis that is approximately consistent with training examples
• Decision tree learning uses information gain
• Learning performance = prediction accuracy on test sets