Learning

Remember “Deduction vs. Induction”

Induction: Derive a general rule (axiom) from background knowledge and observations.

| Socrates is a human. (background knowledge) |
| Socrates is mortal. (observation/example) |
| Therefore, I hypothesize that all humans are mortal. (generalization) |

Most important approach to learning: Inductive Learning

others: rote learning, adaptation/re-structuring, speed-up learning

Deduction is knowledge extraction, induction is knowledge acquisition!
Intelligent Systems must Learn

A flexible and adaptive organism cannot rely on a fixed set of behavior rules but must learn (over its complete life-span)!

If an expert system – brilliantly designed, engineered and implemented – cannot learn not to repeat its mistakes, it is not as intelligent as a worm or a sea anemone or a kitten.

(O. Selfridge)
Learning vs. Knowledge Engineering

- Technological advantage: learning avoids the “knowledge engineering bottleneck”!

- Good analysis of data/observations can often be superior to explicit programming.

- E.g., expert system for medical diagnosis:
  - Describe each patient by his/her clinical data,
  - let physicians decide whether some disease is implied or not,
  - *learn the classification rule* (relevant featureurs and their dependencies)

- Epistemological problem: induced information can never be “sure” knowledge, it has hypothetical status and is open to revision!
Classification/Concept Learning

- Classification/concept learning is the best researched area in machine learning.

- Representation of a concept:
  - **Extensional**: (infinite) set of all entities belonging to a class/concept.
  - **Intensional**: finite characterization
    \[
    T = \{ x \mid \text{has-3/4-Legs}(x), \text{has-Top}(x) \} \]

- Learning: Construction of a finite characterization from a subset of entities (training examples).

- Typically, **supervised** learning: pre-classified positive and negative examples for the concept/class.

- In the following: **decision tree learning** as a symbolic approach to inductive learning of concepts. (Quinlan, ID3, C4.5)
Fictional Example: Learn in which regions of the world, the Tse-Tse fly is living.

Training Examples: can presented sequentially (incremental learning) or all at once (batch learning)

<table>
<thead>
<tr>
<th>Nr</th>
<th>Vegetation</th>
<th>Longitude</th>
<th>Humidity</th>
<th>Altitude</th>
<th>Tse-Tse fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>swamp</td>
<td>near eq.</td>
<td>high</td>
<td>high</td>
<td>occurs</td>
</tr>
<tr>
<td>2</td>
<td>grassland</td>
<td>near eq.</td>
<td>high</td>
<td>low</td>
<td>occurs not</td>
</tr>
<tr>
<td>3</td>
<td>forrest</td>
<td>far f. eq.</td>
<td>low</td>
<td>high</td>
<td>occurs not</td>
</tr>
<tr>
<td>4</td>
<td>grassland</td>
<td>far f. eq.</td>
<td>high</td>
<td>high</td>
<td>occurs not</td>
</tr>
<tr>
<td>5</td>
<td>forrest</td>
<td>near eq.</td>
<td>high</td>
<td>low</td>
<td>occurs</td>
</tr>
<tr>
<td>6</td>
<td>grassland</td>
<td>near eq.</td>
<td>low</td>
<td>low</td>
<td>occurs not</td>
</tr>
<tr>
<td>7</td>
<td>swamp</td>
<td>near eq.</td>
<td>low</td>
<td>low</td>
<td>occurs</td>
</tr>
<tr>
<td>8</td>
<td>swamp</td>
<td>far f. eq.</td>
<td>low</td>
<td>low</td>
<td>occurs not</td>
</tr>
</tbody>
</table>
Feature Vectors

Examples are represented as feature vectors $X_i$

Attributes and their value ranges:

- **Vegetation**: $x_1 \in \{0, 1, 2\}$
- **Longitude**: $x_2 \in \{0, 1\}$
- **Humidity**: $x_3 \in \{0, 1\}$
- **Altitude**: $x_4 \in \{0, 1\}$

Class $y \in \{0, 1\}$
Hypotheses: IF-THEN Classification Rules
IF: Disjunction of conjunctions of constraints over attribute values THEN: Decision for a class

Each path is a rule
- IF grassland THEN no Tse-Tse fly
- IF (swamp AND near equator) THEN Tse-Tse fly
- ...

Generalization: only relevant attributes, rules can be applied to classify unseen examples
ID3(examples, attributes, target-class):

1. Create a root node
2. IF all examples are positive, return the root with label “target-class”+
3. IF all examples are negative, return the root with label “target-class”−
4. IF attributes is empty, return the root and use the most common value for the target class in the examples as label
5. Otherwise
   (a) Select an attribute $A$ from attributes
   (b) Assign $A$ to the root
   (c) For each possible value $v_i$ of $A$
      i. Add a new branch below the root, corresponding to the test $A = v_i$
      ii. Let $examples_{v_i}$ be the subset of examples that have value $v_i$ for $A$
      iii. IF $examples_{v_i}$ is empty
           THEN add a leaf node and use the most common value for the target class in the examples as label
           ELSE ID3($examples_{v_i}$, attributes \ $A$, target-class)
6. Return the tree
Example

x1 (vegetation)

swamp 0

{1, 7, 8}

next attribute?

1

grassland

{2, 4, 6}

NO Tse–Tse Fly

2

forest

{3, 5}

next attribute?

x2 (longitude)

swamp 0

{1, 7, 8}

Tse–Tse Fly

near eq.

{1, 7}

NO Tse–Tse Fly

far from eq.

{8}
Selecting an Attribute

- Naive: From the list of attributes in some arbitrary order, take always the next one
  Problem: DT might be more complex then necessary; contain branches which are irrelevant

- Selection by information gain $\rightarrow$ DT with minimal depth
  What is the contribution of an attribute to decide whether an entity belongs to the class or not?

- Entropy of a collection $S$

\[
H(S) = \sum_{i=1}^{n} -p_i \cdot \log_2 p_i
\]

with $p_i$ as proportion (relative frequency) of objects belonging to category $i$.

- Logarithm to base 2: information theory, expected encoding length measured in bits.
Illustration of entropy

S: flipping a fair coin
\[ p_{\text{head}} = p_{\text{tail}} = 0.5 \]

\[ H(\text{Coin}) = \left[ -0.5 \cdot \log_2 0.5 \right] + \left[ -0.5 \cdot \log_2 0.5 \right] = 1 \text{ bit} \]

- Flipping a coin with 99% heads: \( H = 0.08 \) bits
- Flipping a coin with 100% head: \( H = 0 \) bits

Entropy of a set of training examples: proportion of examples belonging to the class vs. not belonging to the class

\[ H(S') = -\frac{5}{8} \cdot \log_2 \frac{5}{8} - \frac{3}{8} \cdot \log_2 \frac{3}{8} \approx 0.95 \]

Without looking at any attribute, we have a slight advantage if guessing the more frequent class “no Tse-Tse fly”
Information Gain

What does the knowledge of the value of an attribute $A$ contribute to predict the class?

$$Gain(S, A) = H(S) - \sum_{i=1}^{v} \frac{|S_{v}|}{|S|} \cdot H(S_{v})$$

“relative” Entropy of $A$

Example: Attribute $x_2$ “Longitude”

$H(x_2 = 0) = -\frac{3}{5} \cdot \log_2 \frac{3}{5} - \frac{2}{5} \cdot \log_2 \frac{2}{5} \approx -0.6 \cdot -0.74 - 0.4 \cdot -1.32 \approx 0.972$

$H(x_2 = 1) = -\frac{0}{3} \cdot \log_2 \frac{0}{3} - \frac{3}{3} \cdot \log_2 \frac{3}{3} = 0 \cdot 1 = 0$

(remark $\log 0$ is undefined, we set $0 \cdot \log 0$ as $0$)

weight with relation of examples with $x_2 = 0$: $\frac{5}{8}$ and $x_2 = 1$: $\frac{3}{8}$

$$Gain(S, x_2) = 0.95 - [(0.625 \cdot 0.972) + (0.375 \cdot 0)] \approx 0.34$$

Remark: In different subtrees, different attributes might give the best information gain.
Performance Evaluation

A learning algorithm is good, if it produces hypotheses which work good when classifying unseen examples.

Validation: use a training set and a test set
- Construct a hypothesis using the training set
- Estimate the generalization error using the test set

For decision trees, working on linearly separable data, the training error is always 0.

Danger: Overfitting of data
- If crucial information is missing (relevant features), the DT algorithm might nevertheless find a hypothesis consistent with the examples: use irrelevant attributes.
  - E.g., predict whether head or tail of a dice is top based on the day of the week.
How to Avoid Overfitting

- Pruning (Okam’s razor – prefer simple hypotheses): do not use attributes with information gain near zero
- Cross-validation: run the alg. $k$-times, each time leave out a different subset of $\frac{1}{k}$ examples
Characteristics DT Algorithms

- Hypothesis Language (DTs): restricted to separation planes which are parallel to the axis in feature space
- Inductive biases (no learning without bias):
  - Each learning algorithm has a **language bias**: the hypothesis language restricts what can be learned
  - Search bias: top-down search in the space of possible decision trees
- DT research was inspired by psychological work of Bruner, Goodnow, & Austin, 1956 on concept formation
- Advantage of symbolic approach: verbalizable rules which can be directly used in knowledge-based systems
DTs vs. Perceptrons: XOR

Perceptron: \[ o(\vec{x}) = \begin{cases} 
1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\
-1 & \text{otherwise}
\end{cases} \]

- Output: calculated as linear combination of the inputs
- Learning: incremental, change weights \( \vec{w} \) when classification error occurs
- Works for linearly separable classes
- e.g. not for XOR

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Other Learning Approaches

- Theoretical foundations:
  Algorithmic/Computational/Statistic Learning Theory
  What kind of hypotheses can be learned from which data with which effort?

- Three main approaches:
  1. Supervised Learning: learning from examples
     - Decision Tree Algorithms
     - Inductive Logic Programming (ILP)
     - Genetic Algorithms
     - Feedforward Nets (Backpropagation)
     - Support Vector Machines, Statistical approaches
Other Learning Approaches cont.

... (2) Unsupervised Learning: $k$-means clustering, Self-organized Maps
- Clustering of data in groups (e.g. customers with similar behavior)
- Application: Data Mining

(3) Reinforcement Learning: learning from problem solving experience, positive/negative feedback
Other Learning Approaches cont.

- Inductive Program Synthesis; learn recursive rules, typically, small sets of only positive examples
  - ILP
  - Genetic Programming
  - Functional Approaches (recurrence detection in terms)
  - Grammar Inference

- Explanation-base learning

- Learning from Analogical/Case-Based Reasoning (Schema acquisition, abstraction)

Textbook:
Recursive Rules with ILP

**Background Knowledge:**

parent(X,Y) ← mother(X,Y).
father(bob,sharon).
mother(sharon,victor).

**Training Examples:**

ancestor(bob,sharon). ancestor(victor,bob).
ancestor(bob,victor). ancestor(sharon,bob).
ancestor(sharon,victor). ...

Search a hypothesis $H$ (set of horn clauses) so that

$\forall x \in E^\oplus : \text{covers}(H \cup B, x)$ (completeness)

$\forall x \in E^\ominus : \text{covers}(H \cup B, x)$ (consistency)

ancestor(X,Y) ← parent(X,Y).
ancestor(X,Y) ← parent(X,Z), ancestor(Z,Y).
Recursive Functional Programs

Summer’s Approach

- Training Examples: First $k$ input/output pairs
- Hypothesis Language: Linear Recursive Program Scheme

\[
F(x) \leftarrow (p_1(x) \rightarrow f_1(x), \\
\ldots, \\
p_k(x) \rightarrow f_k(x), \\
T \rightarrow C(F(b(x)), x))
\]

- Learning in two steps
  - Rewrite I/O examples into finite term
  - Folding based on detection of recurrence relations
Unpack Example

I/O Pairs:

\[ \text{nil} \rightarrow \text{nil} \]

\[ (A) \rightarrow ((A)) \]

\[ (A \ B) \rightarrow ((A) \ (B)) \]

\[ (A \ B \ C) \rightarrow ((A) \ (B) \ (C)) \]

Rewritten to Term:

\[ F_L(x) \leftarrow (\text{atom}(x) \rightarrow \text{nil}, \right. \]

\[ \left. \text{atom}(\text{cdr}(x)) \rightarrow \text{cons}(x, \text{nil}), \right. \]

\[ \text{atom}(\text{cddr}(x)) \rightarrow \text{cons}(\text{cons}(\text{car}(x), \text{nil}), \text{cons}(\text{cdr}(x), \text{nil})), \]

\[ T \rightarrow \text{cons}(\text{cons}(\text{car}(x), \text{nil}), \text{cons}(\text{cons}(\text{cadr}(x), \text{nil}), \text{cons}(\text{cddr}(x), \text{nil})))) \]
Unpack Example cont.

Folded Program:

\[
\begin{align*}
\text{unpack}(x) & \leftarrow (\text{atom}(x) \rightarrow \text{nil}, \\
& \quad T \rightarrow \text{u}(x)) \\
\text{u}(x) & \leftarrow (\text{atom}(\text{cdr}(x)) \rightarrow \text{cons}(x, \text{nil}), \\
& \quad T \rightarrow \text{cons}(\text{cons}(\text{car}(x), \text{nil}), \text{u}(\text{cdr}(x))))
\end{align*}
\]
Why Program Synthesis

- Generating program code which meets a given specification is an intellectual ability (special case of problem solving)
  - identify principles how humans acquire such knowledge

- Much of programming is non-creative routine work
  - automatization

- Support endprogrammers which cannot write formal specifications or complex programs

- Recursive rules can be used to guide plan construction
  make AI planning for more complex domains feasible
The Running Gag of CogSysI

Question: How many AI people does it take to change a lightbulb? Answer: At least 67.

6th part of the solution: The Learning Group (4)

- One to collect twenty lightbulbs
- One to collect twenty "near misses"
- One to write a concept learning program that learns to identify lightbulbs
- One to show that the program found a local maximum in the space of lightbulb descriptions

("Artificial Intelligence", Rich & Knight)