Design of a learning algorithm is affected by
- Which components of the performance element are to be learned
- What feedback is available to learn these components
- What representation is used for the components

Type of feedback:
- **Supervised learning**: correct answers for each example
- **Unsupervised learning**: correct answers not given
- **Reinforcement learning**: occasional rewards
Supervised Learning
What is Supervised Learning?

- Learning from example data
- Labeled data with outcomes
- Data consists of attribute-value pairs
The data that the computer will learn from

Needs to be different from test set!

But be a nice representative sample
Inductive Learning
Supervised Learning
Inductive Learning (aka Science)

- $f$: target function that actually explains data
- $h$: the hypothesis given a training set of examples

- Simplifies real learning
  - Ignores prior knowledge
  - Assumes a fully observable environment
  - Assumes (good) examples are given
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- ($h$ is consistent if it agrees with $f$ on all examples)

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

($h$ is consistent if it agrees with $f$ on all examples)

E.g., curve fitting:
Inductive learning method

- Construct/adjust $h$ to agree with $f$ on training set
- ($h$ is consistent if it agrees with $f$ on all examples)

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

(h is consistent if it agrees with $f$ on all examples)

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

($h$ is consistent if it agrees with $f$ on all examples)

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

$h$ is consistent if it agrees with $f$ on all examples

E.g., curve fitting:}

Ockham’s razor: prefer the simplest hypothesis consistent with data
Decision Trees

Supervised Learning
Decision Trees

- Tree representation that can express any function of input attributes

<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>

![Tree Diagram]

- Representation of A xor B function using a decision tree.
Learning decision trees

Problem: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)
Attribute-based representations

- Examples described by **attribute values** (Boolean, discrete, continuous)

<table>
<thead>
<tr>
<th>Example</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</th>
<th>Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</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>
<td></td>
</tr>
<tr>
<td>$X_2$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>30–60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_3$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>Some</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>$X_4$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>10–30</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>$X_5$</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>
<td></td>
</tr>
<tr>
<td>$X_6$</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>
<td></td>
</tr>
<tr>
<td>$X_7$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>0–10</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_8$</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>
<td></td>
</tr>
<tr>
<td>$X_9$</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>Full</td>
<td>$</td>
<td>T</td>
<td>F</td>
<td>Burger</td>
<td>&gt;60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_{10}$</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>
<td></td>
</tr>
<tr>
<td>$X_{11}$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>None</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Thai</td>
<td>0–10</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>$X_{12}$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>Full</td>
<td>$</td>
<td>F</td>
<td>F</td>
<td>Burger</td>
<td>30–60</td>
<td>T</td>
<td></td>
</tr>
</tbody>
</table>

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

- One possible representation for hypotheses
Aim: find a small tree consistent with the training examples

Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```python
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value $v_i$ of best do
            examples$_i$ ← {elements of examples with best = $v_i$}
            subtree ← DTL(examples$_i$, attributes $-$ best, MODE(examples))
            add a branch to tree with label $v_i$ and subtree subtree
        return tree
```
Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"

- *Patrons?* is a better choice
To implement **Choose-Attribute** in the DTL algorithm

Information Content (Entropy):

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

For a training set containing \( p \) positive examples and \( n \) negative examples:

\[ I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n} \]
A chosen attribute $A$ divides the training set $E$ into subsets $E_1, \ldots, E_v$ according to their values for $A$, where $A$ has $v$ distinct values.

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

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

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - \text{remainder}(A)$$

Choose the attribute with the largest IG
For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

Consider the attributes *Patrons* and *Type* (and others too):

$$IG(Patrons) = 1 - \left[\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I(\frac{2}{6}, \frac{4}{6})\right] = .0541 \text{ bits}$$

$$IG(Type) = 1 - \left[\frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{2}{12} I(\frac{1}{2}, \frac{1}{2}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4}) + \frac{4}{12} I(\frac{2}{4}, \frac{2}{4})\right] = 0 \text{ bits}$$

*Patrons* has the highest IG of all attributes and so is chosen by the DTL algorithm as the root
How do we know that $h \approx f$?

1. Use theorems of computational/statistical learning theory
2. Try $h$ on a new test set of examples
   (use same distribution over example space as training set)

Learning curve = % correct on test set as a function of training set size
Overfitting: learning a tree that is too good on the example data and will not generalize to test data
Neural Networks
Supervised Learning
Neural Networks

A mathematical model to solve engineering problems
- Group of highly connected neurons to realize compositions of non-linear functions

Tasks
- Classification
- Discrimination
- Estimation

2 types of networks
- Feed forward Neural Networks
- Recurrent Neural Networks
Feed Forward Neural Networks

- The information is propagated from the inputs to the outputs
- Linear and nonlinear functions
- Time is not represented
Perceptrons

- Initial proposal of connectionist networks
- Rosenblatt, 50’s and 60’s
- Essentially a linear discriminant composed of nodes, weights

\[
\begin{align*}
O &= \begin{cases} 
1: \left( \sum w_i I_i \right) + \theta > 0 \\
0: & \text{otherwise}
\end{cases}
\end{align*}
\]
Perceptron Example

Randomly assign weights (between 0-1)

Present inputs from training data

Get output $O$, nudge weights to gives results toward our desired output $T$

Repeat; stop when no errors, or enough epochs completed

$2(0.5) + 1(0.3) + -1 = 0.3$, $O=1$
Perception Training

\[ w_i(t + 1) = w_i(t) + \Delta w_i(t) \]
\[ \Delta w_i(t) = \eta(T - O)I_i \]

Learning rate

Weights include Threshold. \( T \) = target output, \( O \) = actual output.

Example: \( T=0, O=1, W1=0.5, W2=0.3, I1=2, I2=1, \text{Theta}=-1 \)

\[ w_1(t + 1) = 0.5 + (0 - 1)(2) = -1.5 \]
\[ w_2(t + 1) = 0.3 + (0 - 1)(1) = -0.7 \]
\[ w_\theta(t + 1) = -1 + (0 - 1)(1) = -2 \]

If we present this input again, we’d output 0 instead.
How might you use a perceptron network?

- This (and other networks) are generally used to learn how to make classifications.

- Say you have collected some data regarding the diagnosis of patients with heart disease:
  - Age, Sex, Chest Pain Type, Resting BPS, Cholesterol, ..., Diagnosis (<50% diameter narrowing, >50% diameter narrowing)

- 67, 1, 4, 120, 229, ..., 1
- 37, 1, 3, 130, 250, ..., 0
- 41, 0, 2, 130, 204, ..., 0

- Train network to predict heart disease of new patient
Perceptron

Linear separation

\[ y = \text{sign}(v) \]

\[ v = c_0 + c_1 x_1 + c_2 x_2 \]

\[ c_0 + c_1 x_1 + c_2 x_2 = 0 \]
XOR Problem: Not Linearly Separable!

We could however construct multiple layers of perceptrons to get around this problem.
Typically we have *fully connected, feedforward* networks.
Learning Procedure:

Randomly assign weights (between 0-1)

Present inputs from training data, propagate to outputs

Compute outputs \( O \), adjust weights according to the delta rule, backpropagating the errors. The weights will be nudged closer so that the network learns to give the desired output.

Repeat; stop when no errors, or enough epochs completed
Backprop - Modifying Weights

We had computed:

$$\Delta w_k = c I_k (T_j - O_j) f'(ActivationFunction), \quad f = \left( \frac{1}{1 + e^{-\text{sum}}} \right)$$

$$\Delta w_k = c I_k (T_j - O_j) f(\text{sum})(1 - f(\text{sum}))$$

For the Output unit $k$, $f(\text{sum})=O(k)$. For the output units, this is:

$$\Delta w_{j,k} = c H_j (T_k - O_k) O_k (1 - O_k)$$

For the Hidden units (skipping some math), this is:

$$\Delta w_{i,j} = c H_j (1 - H_j) I_i \sum_k (T_k - O_k) O_k (1 - O_k) w_{j,k}$$
Backprop

Fig. 7.21. All paths up to input site i
Backprop

- Very powerful - can learn any function, given enough hidden units! With enough hidden units, we can generate any function.

- Have the same problems of Generalization vs. Memorization. With too many units, we will tend to memorize the input and not generalize well. Some schemes exist to “prune” the neural network.

- Networks require extensive training, many parameters to fiddle with. Can be extremely slow to train. May also fall into local minima.

- Inherently parallel algorithm, ideal for multiprocessor hardware.

- Despite the cons, a very powerful algorithm that has seen widespread successful deployment.
Unsupervised Learning

clustering
Unsupervised Learning

- Unlabeled data

- Find patterns or anomalies
K-means Clustering

- How many clusters do you want? $K$
- Pick a random $K$ points in space to be the center of your clusters
- Until cluster centers do not change
  - Assign every data point to closest cluster center
  - Update cluster center to be centroid of newly formed cluster
K-means clustering

The Good

- Simple
- Fast
- Does a reasonable job for simple clusters

The Bad and/or Ugly

- What is k?
- Non-overlapping clusters
- Sensitive to outliers