Supervised Learning

Neural Nets

- regression: given inputs and outputs, find good weights of input nodes
- neural networks: given inputs and outputs, introduce hidden layers and find good weights
- start with random weights and adjust based on perceived error

Activation function: the output of each node is based on the weighted sum of inputs, run through a nonlinear function (often sigmoid)

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Nonlinear activation function means nonlinear system behavior

Supervised Learning

Neural Nets

Feedforward

BackProp

Success

Decision Trees

Neural Nets

- regression: given inputs and outputs, find good weights of input nodes
- neural networks: given inputs and outputs, introduce hidden layers and find good weights
- start with random weights and adjust based on perceived error

Activation function: the output of each node is based on the weighted sum of inputs, run through a nonlinear function (often sigmoid)

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Nonlinear activation function means nonlinear system behavior

Feedforward

- With good weights, a neural net has a good approximation function: inputs and outputs match the training data
- Feed the input, calculate the values of each node, check the outputs
- Initial weights were random, though!
- We need to change weights for every connection to decrease the output error

Back-propagation

- After feeding input values forward, we have a prediction for the output
- Based on this prediction and the actual labeled output, we trace error backwards through system

For the last set of weights:

$$W_{i,j} = W_{i,j} - \alpha\left(\frac{\delta Error}{\delta W_{i,j}}\right)$$
$$= W_{i,j} - \alpha\Delta i$$

To update hidden layer weights, we use weighted sum of changes

$$W_{i,j} = W_{i,j} + \alpha \sum_j W_{i,j}\Delta_j$$

Successful Neural Nets

- $+$: Neural nets perform better with multiple hidden layers
- $+$: Excellent results for many training sets
- $-$: Training is computationally intensive
- $-$: Weights and hidden nodes are not intuitive

Decision Trees

Example

Example	Input Attributes										Output
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Restaurant domain

Learning a Decision Tree

Goals:

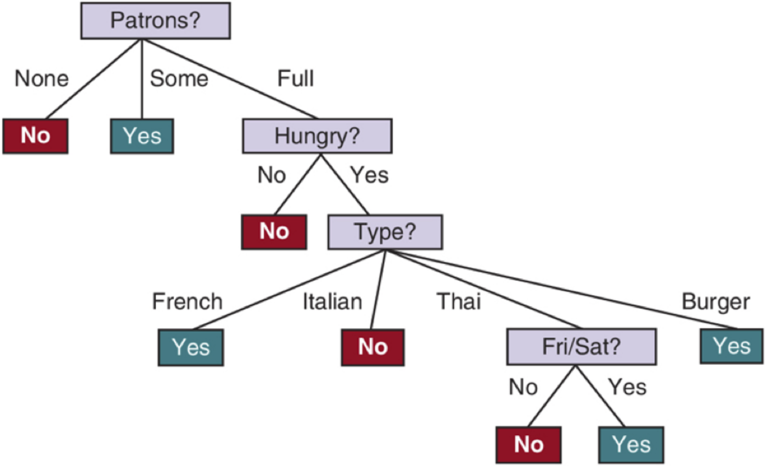
- Decision tree matches data
- Decision tree is small (fewer nodes, smaller height)

Approach:

- Analyze attributes for best simplification
- Split examples based on that attribute
- For all values of attribute, find remaining best attribute…
- Resulting tree is a small tree that makes good predictions

Resulting tree may not match original (unobservable) tree

Example



Restaurant Decision Tree