Unit 2

Learning

Chapter 18, 20

Learning: A simple example with Bayesian Networks

- Previously, we gave CPTs to describe probability distribution at each node
- Can we learn these from data?

Alarm example

JohnCalls	MaryCalls	Alarm	Burglary	Earthquake	Count
N	N	N	N	N	85
N	Y	N	N	N	5
Y	N	N	N	N	1
Y	Y	Y	Y	N	1
Y	Y	Y	N	Y	3
Y	N	Y	N	N	1
Y	N	Y	N	Y	2
N	N	N	Y	N	1
N	N	N	N	Y	1

What is learning?

- Improving performance based on experience:
  - Discovering new relationships between input and output
    - Which input data is important and which can be disregarded
  - Discovering properties of the environment
    - Learning the configuration of a maze
- Describing experience in a concise way
  - Which features are best mapping from input to output
  - Better than lookup table with previous experiences

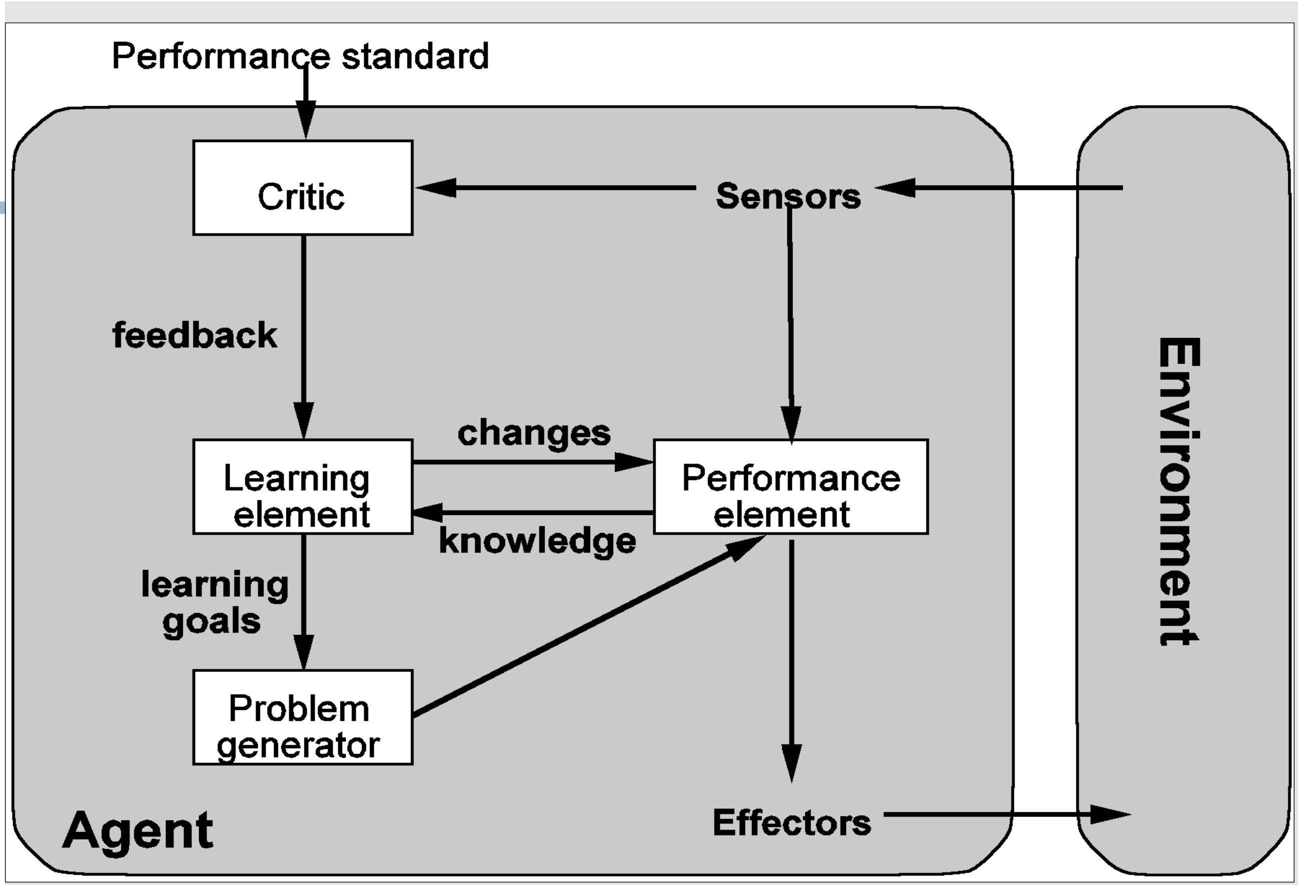
Learning Agents

- Agents have two major components:
  - Performance element:
    - Selecting action to take at each timestep
  - Learning element:
    - Improving the selection process by comparing outcome to optimal outcome
- Performance element must be written so that it can be modified by learning
  - Logical statements or very modular code
  - Parameters, parameters, parameters!

What can be learned?

- Classifications:
  - Identify hand-written digits
  - Filter mail into spam/not spam
  - Find the face in a photo

- Actions:
  - Robot balances upright on two legs
  - Autopilot flies level
  - Keep vehicle in lane



Problem generator suggests exploratory actions rather than just letting the performance element select the action that it has already learned is best

Feedback

- Indicates the results of agent's output
- Can be in terms of correct answer provided by a friendly teacher
  - *Supervised* learning
  - Agent compares its answer to answer key
  - Good for precise answers like classifications (recognizing handwritten digits)
- Can be a system of rewards and punishments
  - *reinforcement*
  - Better for gradient actions (balancing on two legs)

Prior Knowledge

- Learning can begin with the first experience or can benefit from built-in bias
  - Canned chess moves
  - Ratio of letter occurrences in English

Hypotheses (reminder)

- A hypothesis h is an approximation of the true function f that you are trying to learn
- Pure inductive inference
  - Given {(x,f(x))}, return h(x) which approximates f(x)
- The space of all hypothesis functions is **H**
  - This is chosen by the person designing the learning algorithm

Hypothesis spaces

- A hypothesis is **consistent** if it can explain all of the data in the training set
- Ockham's razor: prefer the simplest hypothesis consistent with the data
  - Prefer 4th degree over 12th degree poly.
- Learning problem is **realizable** if $f \in H_L$
  - Above function is unrealizable in H_1, H_2
  - Above function is realizable in H_k s.t. $k \geq 4$

Forming hypotheses

- Imagine you had the following data:

PassExam	PassHomework	Pass5522
Y	Y	Y
Y	N	N
Y	Y	Y
N	Y	N
N	N	N
N	Y	N

- What's the decision rule for Pass5522?

Decision Trees

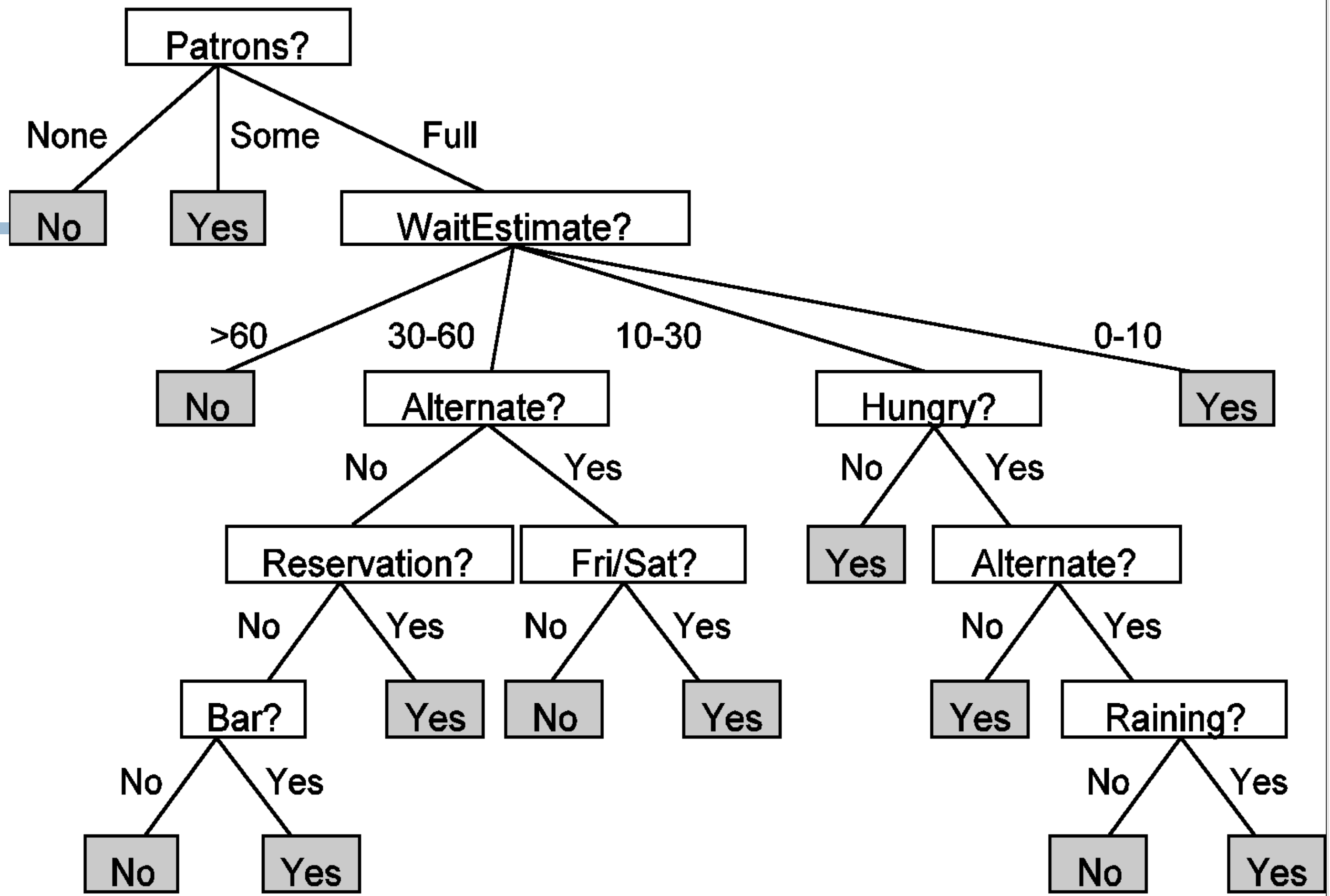
- Input examples: feature vectors
- Output:
  - Interior nodes: yes/no decision
  - Leaf nodes: Action or classification
- Learns by progressively subdividing data into clusters with homogeneous properties
- Good at determining which features are good discriminators

Attribute-based representations

Examples described by attribute values (Boolean, discrete, continuous, etc.)
E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Classification of examples is positive (T) or negative (F)


Patrons?
None
Some
Full
No
Yes
WaitEstimate?
>60
30-60
10-30
0-10
No
Alternate?
Hungry?
Yes
No
Yes
No
Yes
Reservation?
Fri/Sat?
Yes
Alternate?
No
Yes
No
Yes
No
Yes
Bar?
Yes
No
Yes
Yes
Raining?
No
Yes
No
Yes
No
Yes
No
Yes

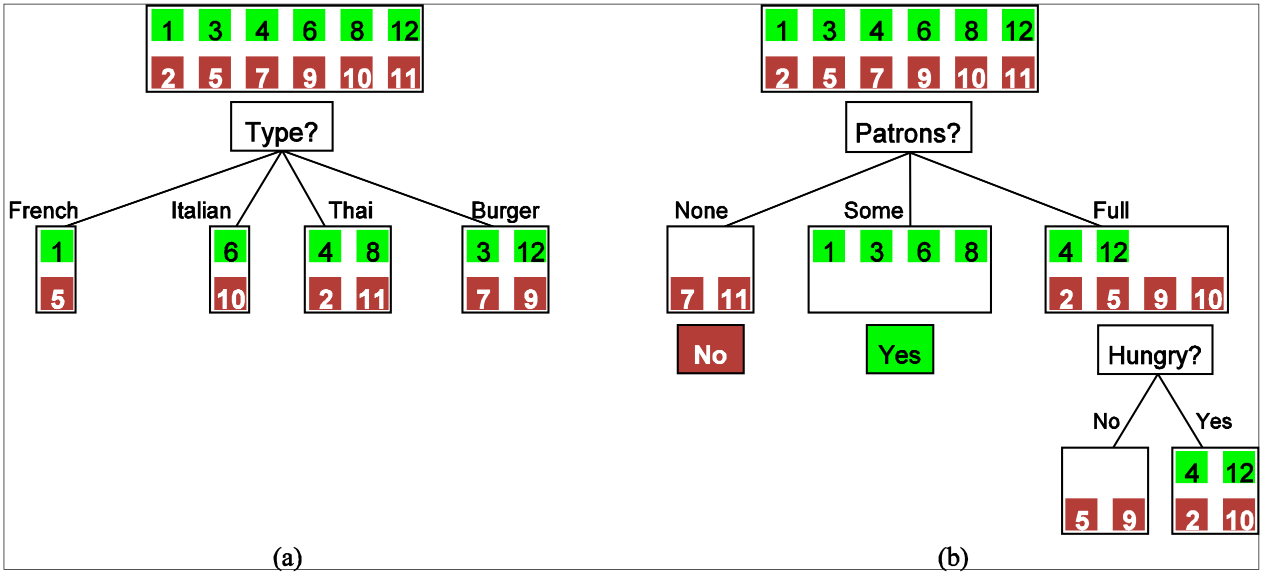
Building a decision tree

1) Assign training examples to node
2) Determine which property creates best partition for data at the node using information gain
3) Partition the data and store in child nodes
4) Stop when:
    1) Child node contains data of one classification
       (If multiple classifications but all attributes same, the majority classification will be used)
    2) Improvement from further splits below some threshold
    3) No more than X singleton nodes

Choosing the best Partition

- Greedy algorithm chooses the split that optimizes information gain at each node
  1. Generate all possible splits
  2. For each split, calculate information gain
  3. Choose split with highest gain

Two potential top-level splits



Green: Wait=Y
Brown: Wait=N

Information Gain

- Information: how much information does an actual observation provide compared to the prior bias

The entropy (uncertainty) of a node prior to a split:

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

If variables are binary (2 values), then 0 ≤ I(P(t),P(f)) ≤1

P(1/2, 1/2) =1, P(0,1)=0

Information Gain

- Information: how much information does an actual observation provide compared to the prior bias

The entropy (uncertainty) of a node prior to a split:

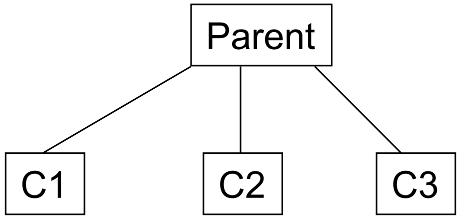
$$I(P(v_1),\ldots,P(v_n)) = \sum_{i=1}^{n} -P(v_i)\log_2 P(v_i)$$

After a split on some question Q, we will now have k nodes $q_1 \ldots q_k$, each with a probability distribution P_j.

$$Remainder(Q) = \sum_{j=1}^{k} P(Q = q_j) I(P_j(v_1),\ldots,P_j(v_n))$$

$$Gain(Q) = I(P(v_1),\ldots,P(v_n)) - Remainder(Q)$$

Example of gain calculation



- Gain (for a boolean classification var):

$$C_i = count(data_{child\_i})$$

$$C_i^+ = count(trues_{child\_i})$$

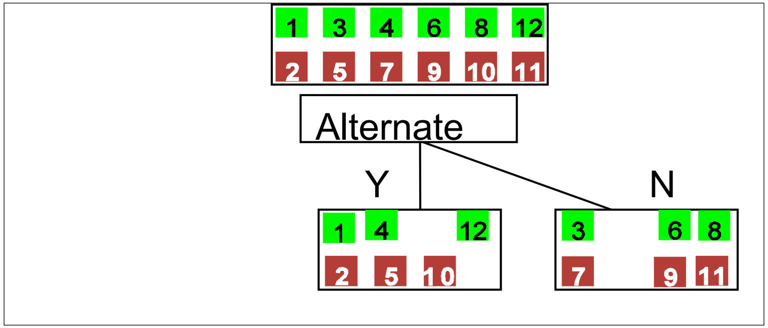
$$C_i^- = count(falses_{child\_i})$$

$$P = count(data_{parent})$$

$$Gain = I_{parent} - \left[\frac{C_1}{P} * I\left(\frac{C_1^+}{C_1}, \frac{C_1^-}{C_1} \right) + \frac{C_2}{P} * I\left(\frac{C_2^+}{C_2}, \frac{C_2^-}{C_2} \right) + \frac{C_3}{P} * I\left(\frac{C_3^+}{C_3}, \frac{C_3^-}{C_3} \right) \right]$$

Testing top-level splits: Alternative

•6 records where Alt=Y and 6 where Alt=N

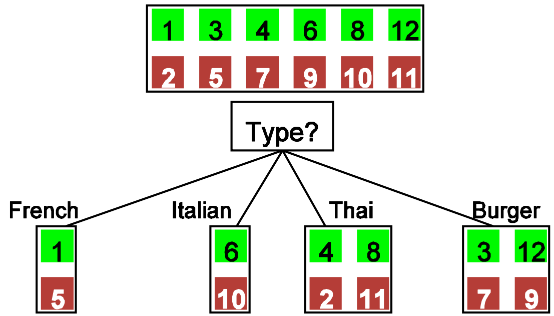


Gain = 1 – [6/12*I(3/6,3/6) + 6/12*I(3/6,3/6)]
= 1 – [.5*1 + .5*1] = 0

k = 6, 3 have WAIT=t, 3 have WAIT=f

$$I\,(3/6,3/6) = -3/6\log_2 3/6 - 3/6\log_2 3/6$$

Testing top-level splits: Type



$$\text{Gain} = 1 - [2/12*I(1/2,1/2) + 2/12*I(1/2,1/2) + 4/12*I(2/4,2/4) + 4/12*I(2/4,2/4)]$$

$$= 1 - [1/6 + 1/6 + 2/6 + 2/6] = 0$$

k = 4, 2 have WAIT=t, 2 have WAIT=f

$$I\,(2/4,2/4) = -2/4\log_2 2/4 - 2/4\log_2 2/4$$

Testing top-level splits: Patrons

Gain = 1 –

[2/12*I(0/2,2/2)+4/12*I(4/4,0/4)+6/12*I(2/6,4/6)]

= 1 – [2/12*0 + 4/12*0 + 6/12*.918

= 1 - .459 = .541

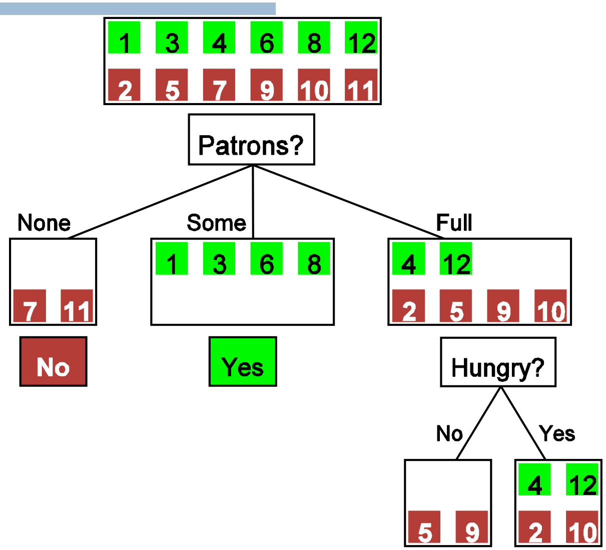
(so this is a better split than ‘alt’)

Split on Hungry within patrons=full:

Gain = .918 – [2/6* I(0/2,2/2)

+4/6*I(2/4,2/4)]

= .918 – [0 + 4/6*1] = .251



What do we do with the decision tree?

- Present it with new data (test data)
- See how well it classifies
- Report classification success on test data
  - This gives an idea of how well it would work on data it has not seen before
- Deploy it in the field

Another Example: Spam detection

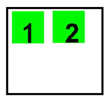
- Incoming email classified as spam/not spam
- Generate a list of possible questions about the data:
  - Is the sender from a known list of anonymous sites?
  - Does the subject line contain non-words?
  - Does the body contain jpegs?
  - Does the body contain spam words?
  - Does my email address appear in the recipients list?
  - Is the body in html format?
  - …

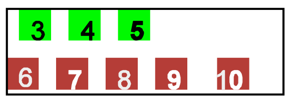
Spam Example

	Sndr	Recip	Viagra	Mortgage	Jpg	Non-words	html	Class
1	Y	N	N	Y	N	Y	Y	S
2	Y	N	N	N	N	N	Y	S
3	N	Y	N	N	Y	Y	N	S
4	N	Y	N	N	N	N	N	S
5	N	N	Y	N	N	N	N	S
6	Y	N	N	N	N	N	N	N
7	Y	Y	N	N	N	N	N	N
8	Y	Y	N	N	N	Y	N	N
9	N	N	N	Y	N	N	N	N
10	Y	N	N	N	N	N	N	N

Split on HTML

- I of top node is 1 (perfectly balanced)
- HTML has 2 values: Y and N

- Y node: 

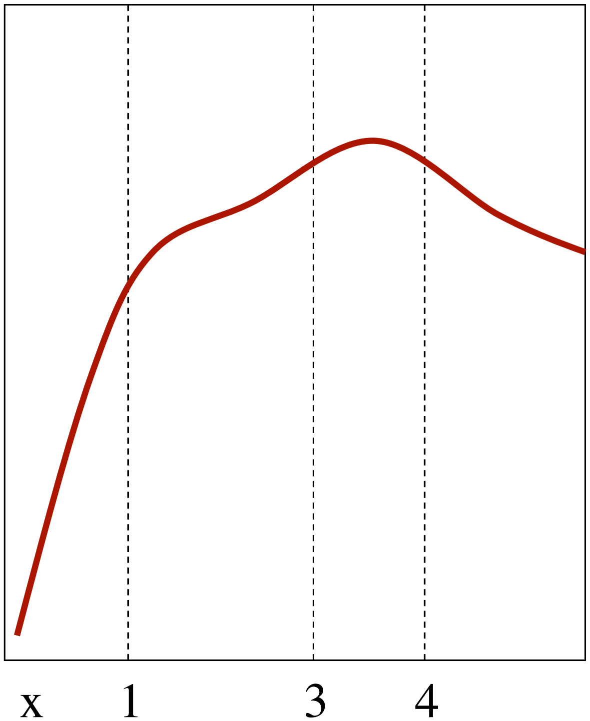
- N node: 

Gain = 1 – [2/10*I(2/2,0/0) + 8/10*I(3/8,5/8)]
= 1 – [0 + 8/10*(-.53 - -(.42))] = 1 - .76 = .24

Continuous D-Tree attributes

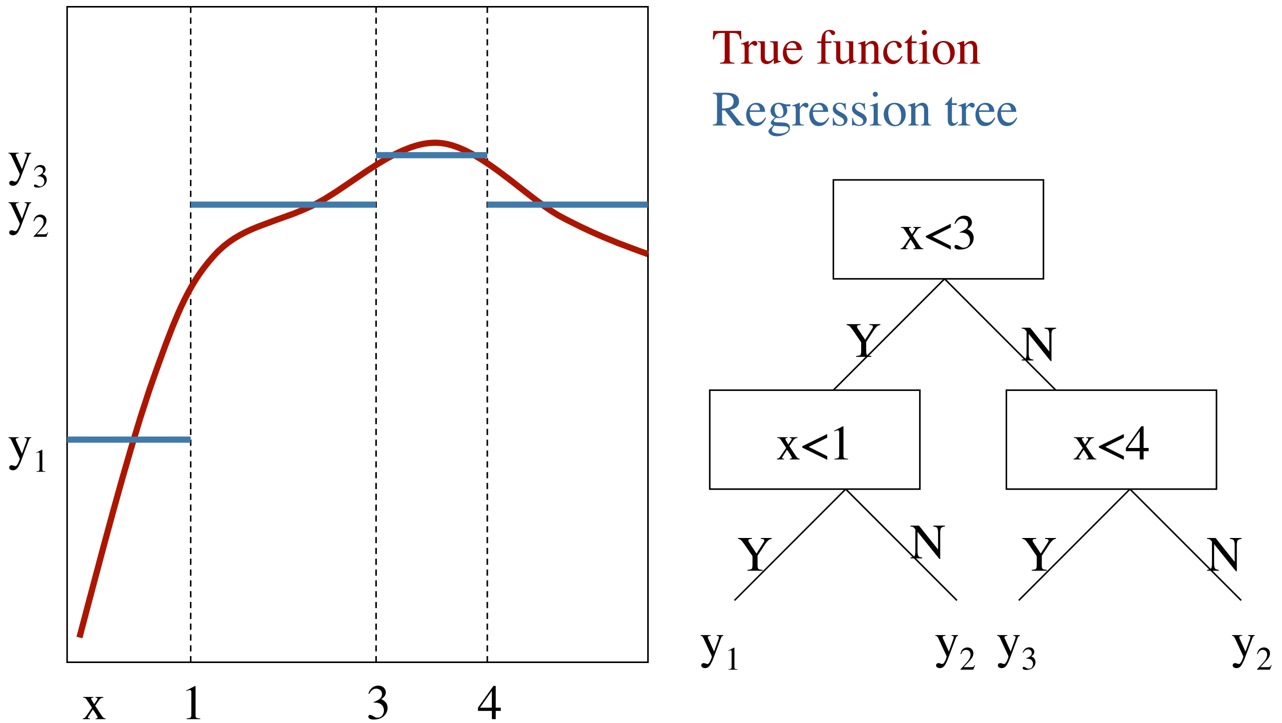
- So far, we have only talked about
  - Predicting binary (discrete) variables
  - using binary (discrete) attributes
- Can have continuous variables, however:
  - Continuous attributes: questions are *split points*
    - Age < 18, Height > 60
    - Theoretically, infinitely many split points
    - In practice, # of split points determined by data
  - Predicting continuous variables: regression trees
    - Each leaf has a local function that predicts the value

Regression tree example

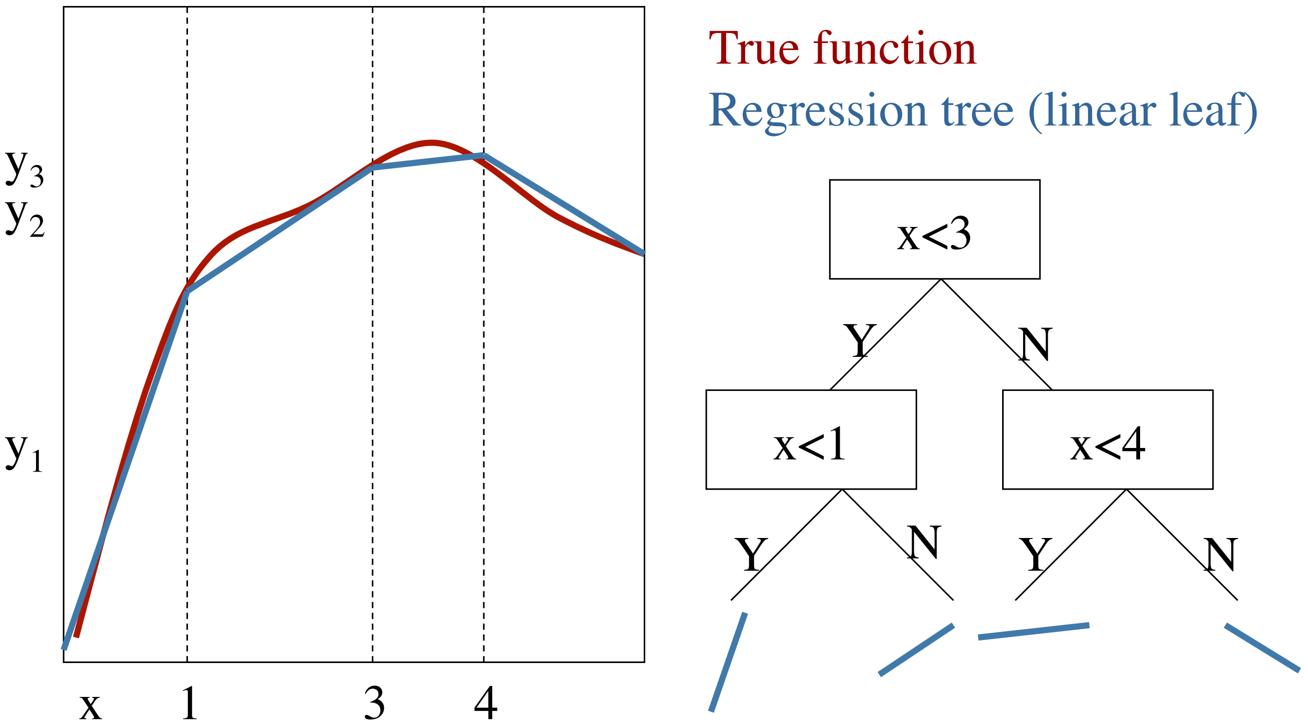


True function

Regression tree example



Regression tree example



Evaluating Learning

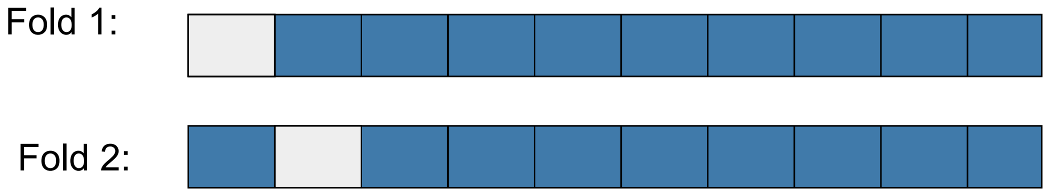
- Training data must be selected so as to reflect the global data pool
- Testing on unseen data is crucial to prevent *over-fitting* to the training data*:*
  - Unintended correlations between input and output
    - Eg. Photos with tanks were taken on a sunny day
  - Correlations specific to the set of training data
    - Eg. Language processing trained on Wall Street Journal articles may not extend to spoken dialog

Training vs. Test data

- Learning agents are presented a collection of *training examples*
- Modifications are made to the learning algorithm until it performs well on the training data
- Test data is *held out* from this process
- When performance on training data is acceptable, algorithm is run on test data
- Only performance on test data is reported

Test data methodologies

- Single pass: reserve x% of data for test
- Cross-validation:
  - Each *fold* reserves x% of data, trains on rest and tests on held out data



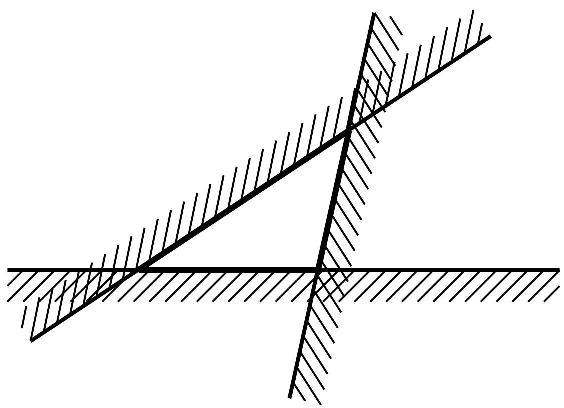
  - Performance is averaged across folds
  - Statistical tests tell how many test cases are needed for reliable conclusions

Integrating CV with Decision Trees

- As a tree gets deeper, hypothesis becomes more specific
  - If too specific, then hypothesis overfits training data
- For each CV fold, train full d-tree on training set
  - Then prune back tree to minimize error on CV set
- Find best depth d across all folds
  - Regrow tree using full data set, but stop at depth d.

Ensemble learning

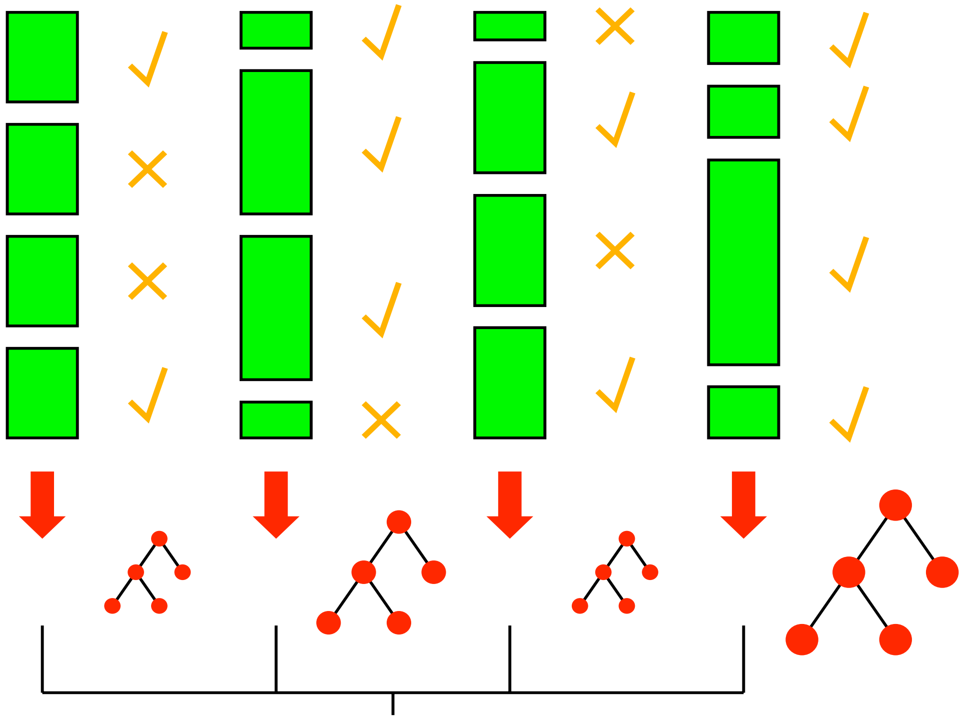
- Develop a suite of classifiers and combine their votes (pick the majority classification)
- Train a new classifier on the errors from another classifier



Boosting

- Training examples are assigned weight or importance
- The classifier’s opinion is weighted in proportion to the weight of the examples it learned
- Initially all examples weighted the same
- Weight of misclassified ones are boosted

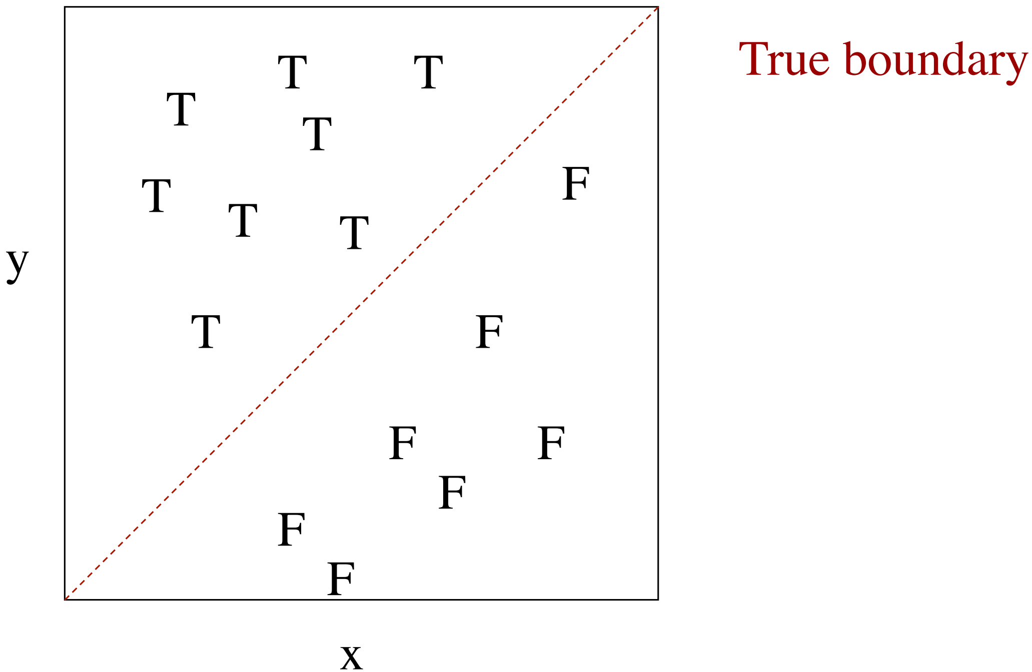
Boosting



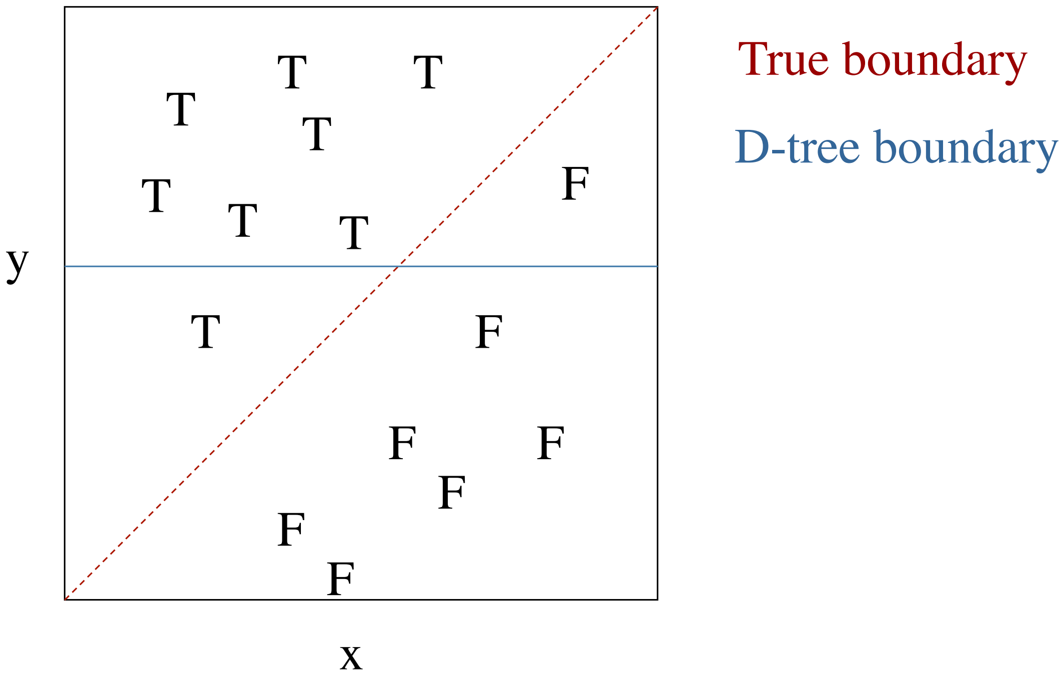
Hypothesis spaces (again)

- Consider the expression x < y
  - Three variables: X, Y, X<Y
    - First two are real, last is boolean
- Would a decision tree be able to learn this relation?

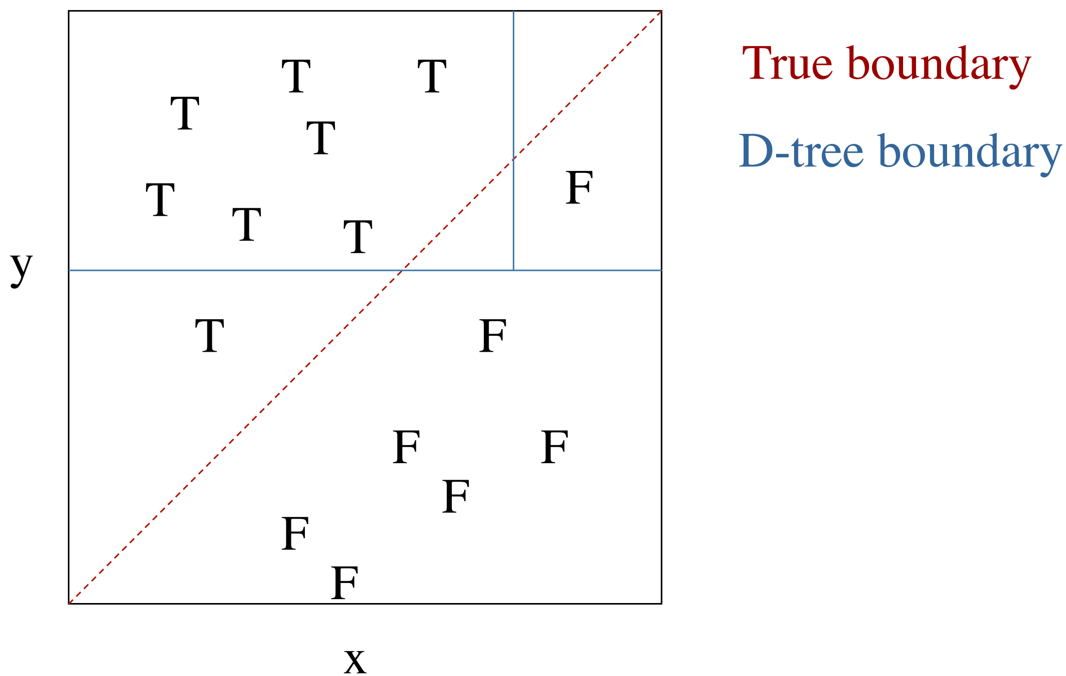
X < Y



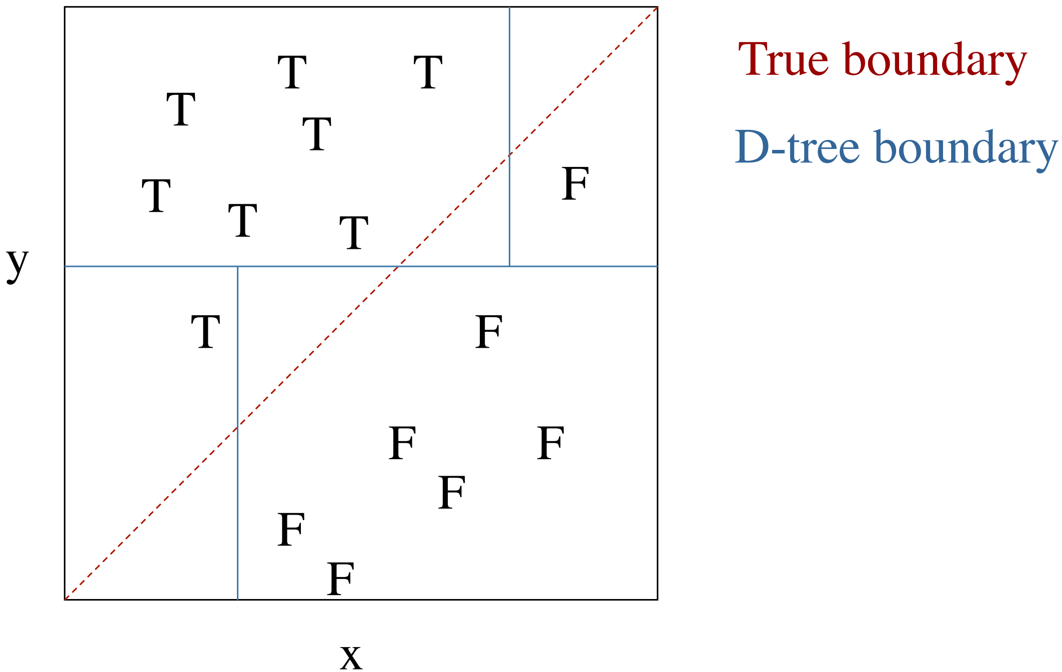
X < Y



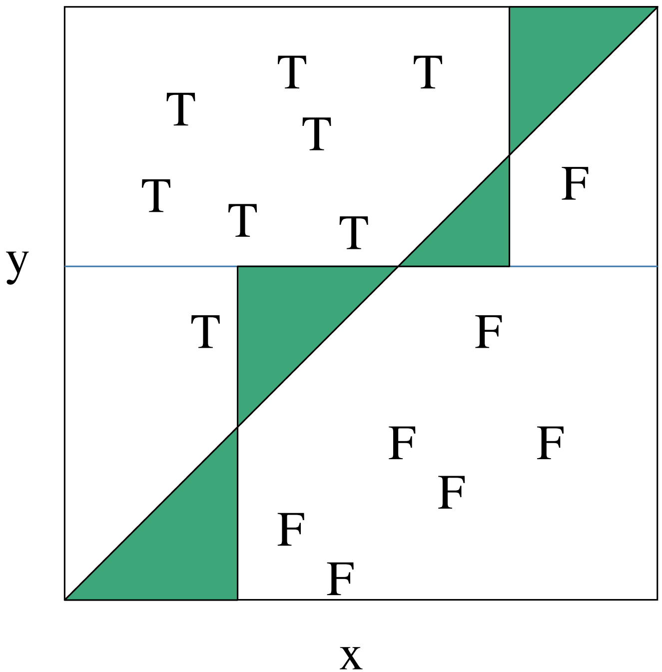
X < Y



X < Y



X < Y



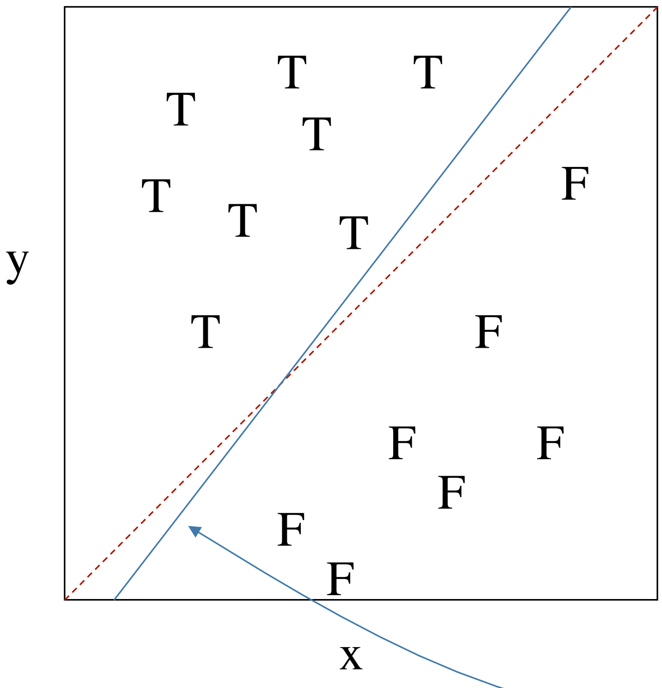
True boundary

D-tree boundary

Errors made by
d-tree

Only with infinite data can d-tree get true boundary

X < Y



True boundary

- If hypotheses are allowed to be general 2-space lines, can get better approximation
- $w_x x + w_y y - w_b < 0$

Neural networks

- A trainable, mathematical model for finding boundaries
- Inspired by biological neurons
  - Neurons collect input from receptors, other neurons
  - If enough stimulus collected, then neuron fires
- Inputs in artificial neurons are variables,outputs of other neurons
- Output is an “activation” determined by the weighted inputs

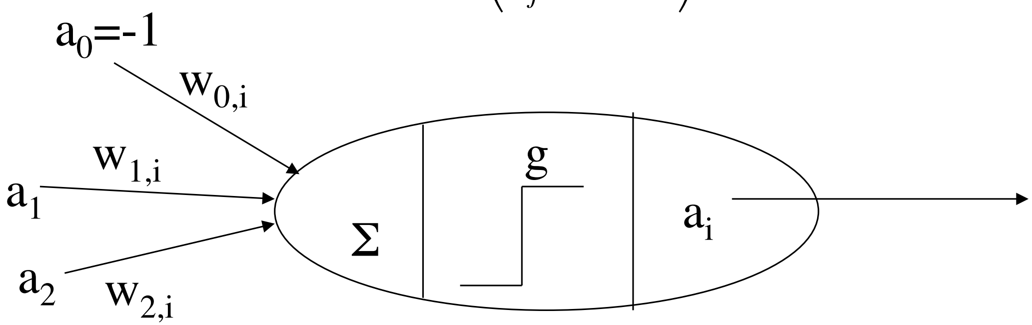
Neural Network Units

- The input to a neuron is given as

$$in_i = \sum_j w_{j,i} a_j$$

- The activation of the unit is given as

$$a_i = g(in_i) = g\left(\sum_j w_{j,i} a_j\right)$$



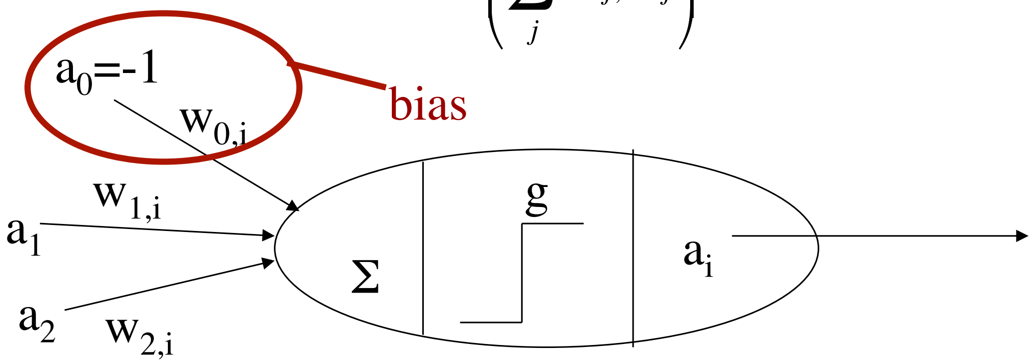
Neural Network Units

- The input to a neuron is given as

$$in_i = \sum_j w_{j,i} a_j$$

- The activation of the unit is given as

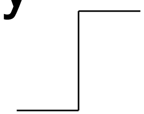
$$a_i = g(in_i) = g\left(\sum_j w_{j,i} a_j\right)$$



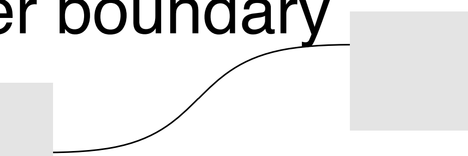
Activation function

- The g() function acts as a decision rule
- Thresholding: hard boundary

  g(x)=0 if x<0, 1 otherwise



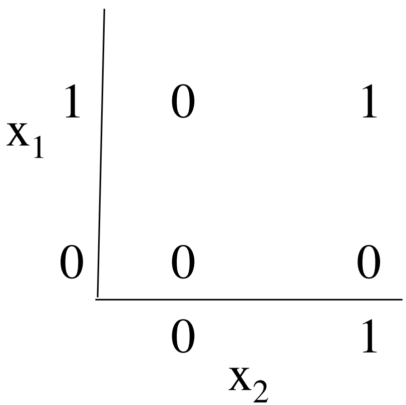
- Sigmoid function: softer boundary



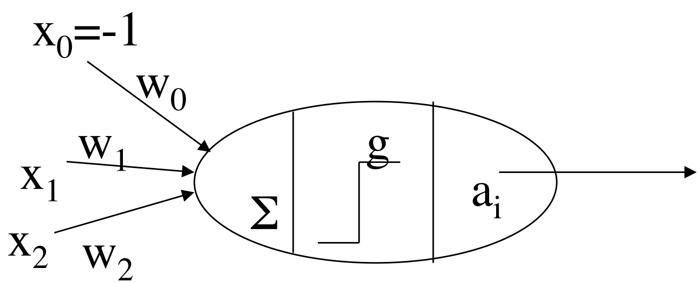
$$g(x) = \frac{1}{1+e^{-x}}$$

Single layer perceptron & logical functions

- An array of neurons is called a perceptron
- A single neuron can be used to represent the logical functions and, or, not



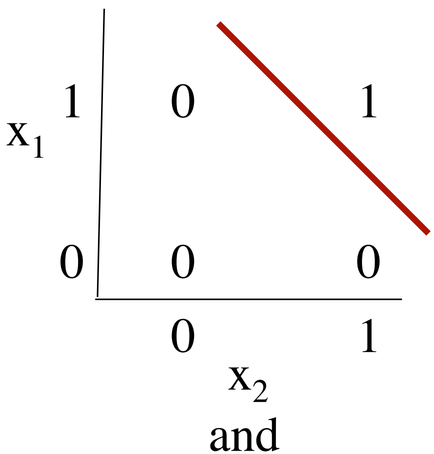
and

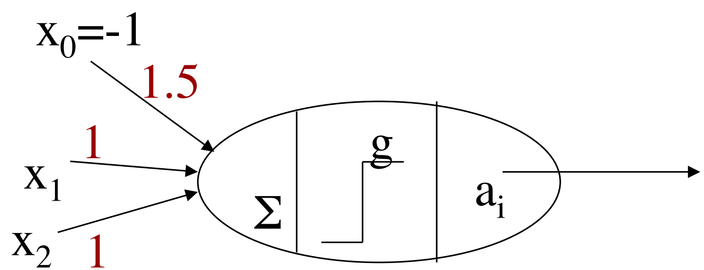


$$a_i = g\left(\sum_{j=0}^{2} w_j x_j\right) > 0$$

Single layer perceptron & logical functions

- Setting w_0 to 1.5 and w_1,w_2 to 1 gives "and" rule.
- Similar for or, not.





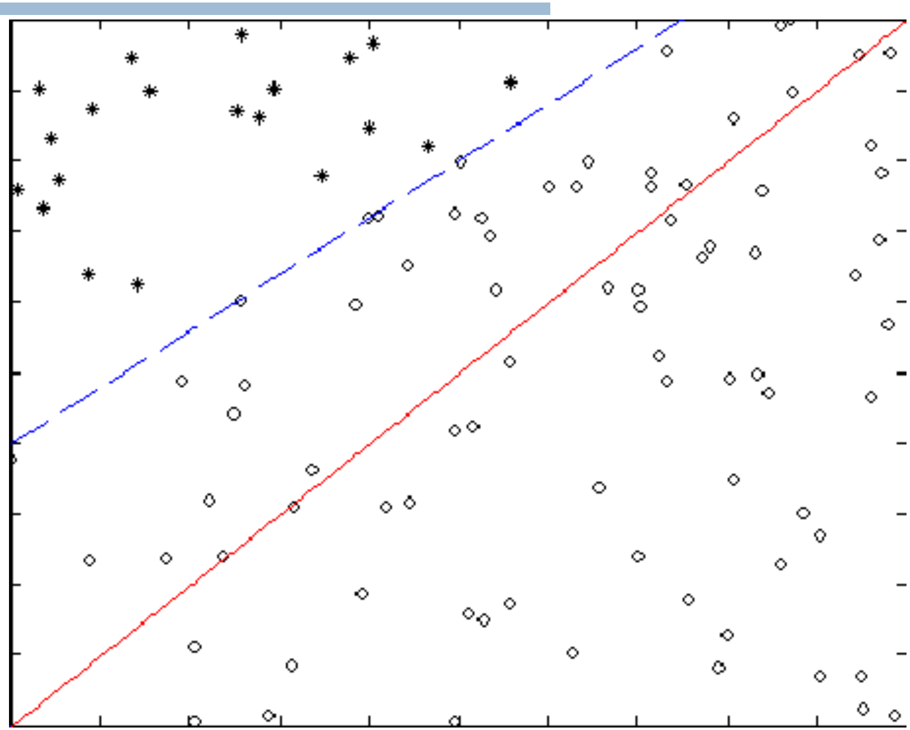
$$a_i = g\left(\sum_{j=0}^{2} w_j x_j\right) > 0$$

Perceptron learning rule

- Let x be a training example
- Let y_x be the desired output (0 or 1)
- calculate output $O_x=g(Wx)$
- calculate error $e_x=y_x-o_x$
- Update weight $W_j \leftarrow W_j + \alpha\, g'(Wx)\, e_x\, x_j$
  - Threshold units: $W_j \leftarrow W_j + \alpha\, e_x\, x_j$
- Continue until stopping criterion is reached

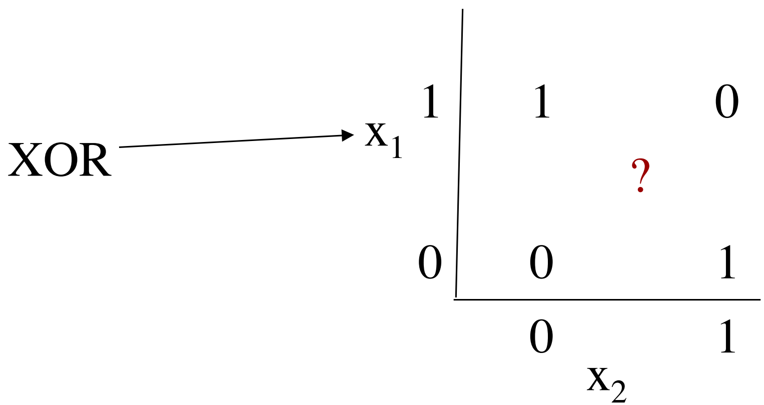
Example of PLR in action

- True boundary is blue dotted line
- Watch as red line moves – red points are ones with errors



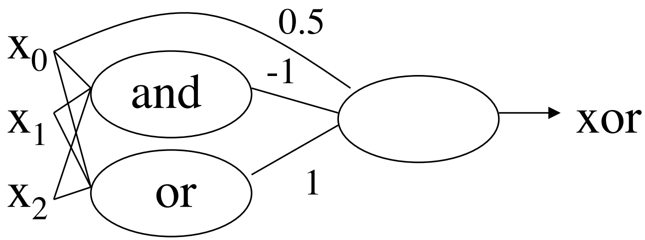
XOR problem

- Originally, a lot of excitement over neural networks
- Minsky and Pappert (1969) then showed that there were problems that you couldn't represent using single layer perceptrons



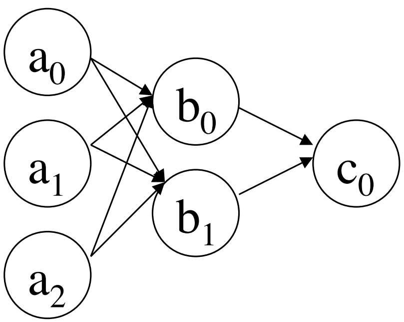
XOR problem

- Notice that you can express XOR as a combination of other functions: x_1 XOR $x_2 = (x_1 \vee x_2) \wedge \sim(x_1 \wedge x_2)$
- We can build an ensemble network: multi-layer perceptron



Multi-layer perceptron

- Key idea: output from first neurons becomes input for later neurons
- Middle node known as **hidden layer**

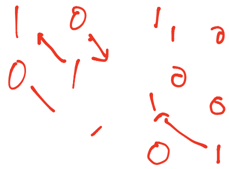


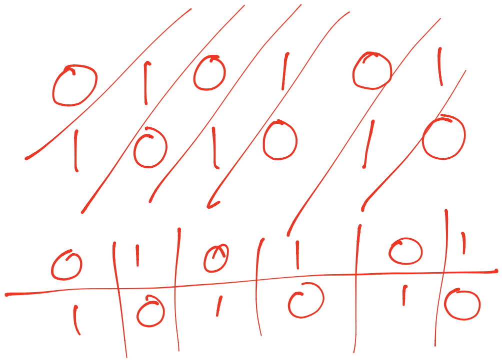
$$o = g\left(\sum_j w_j^{bc}\, g\left(\sum_k w_{kj}^{ab} a_k\right)\right)$$

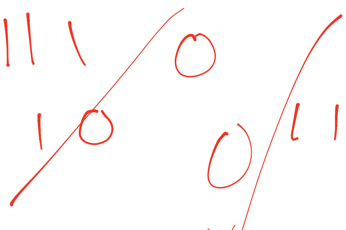
g(Wx)

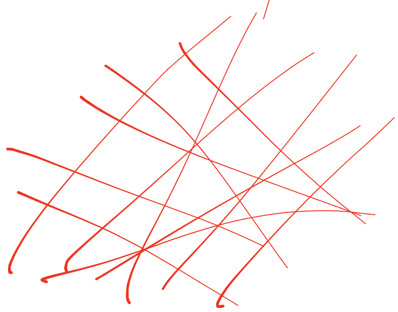
V · Wx

(VW)x









Training Multi-Layer Perceptrons

- Let's go back to the perceptron learning rule
- Define Error $E = \frac{1}{2}Err^2 = \frac{1}{2}(y_x - o_w(x))^2$
- Can re-write update rule as

$$W_j \leftarrow W_j + \alpha \frac{\partial E}{\partial W_j}$$

PLR as error deriviative

- After some math…

$$\frac{\partial E}{\partial W_j} = Err \times \left(-g' \left(\sum_i w_i x_i \right) \right) \times x_j$$

PLR as error deriviative

- After some math…

$$\frac{\partial E}{\partial W_j} = Err \times \left(-g'\left(\sum_i w_i x_i \right) \right) \times x_j$$

Which direction is error going?

How close to boundary are you?

What is the size of the input?

PLR as error deriviative

- But for MLPs, x_i is the output of previous layer…

$$\frac{\partial E}{\partial W_j} = Err \times \left(-g'\left(\sum_i w_i x_i \right) \right) \times x_j$$

Which direction is error going?

How close to boundary are you?

What is the size of the input?

Error backpropagation

- Now E defined as

$$E = \frac{1}{2}\left[y - g\left(\sum_j w_j^{bc}\, g\left(\sum_k w_{kj}^{ab} a_k \right) \right) \right]^2$$

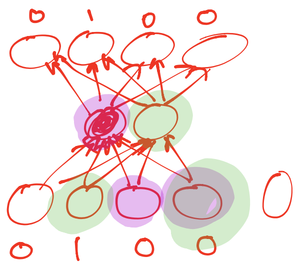
- Calculate $\partial E / \partial w_j^{bc}$ as before
- Use chain rule to calculate $\partial E / \partial w_{kj}^{ab}$

Intuition behind backprop

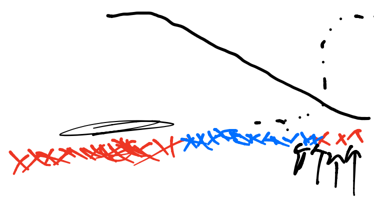
- We have some error and want to assign the blame to weights proportionally
- We can compute the error derivative for the last layer
  - Accumulate blame at each of the hidden nodes by summing over weights attached to that node
- Now distribute blame to previous layer

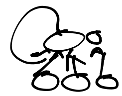
Backprop training

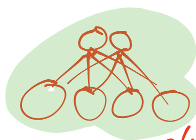
- Start with random weights
- Run the network forward
- Calculate the error
- Propagate the error backward
- Update the weights
- Repeat with next training example until you stop improving
- Cross validation data is very important here



Restricted Bolzmann Machine







$g(Wx)$

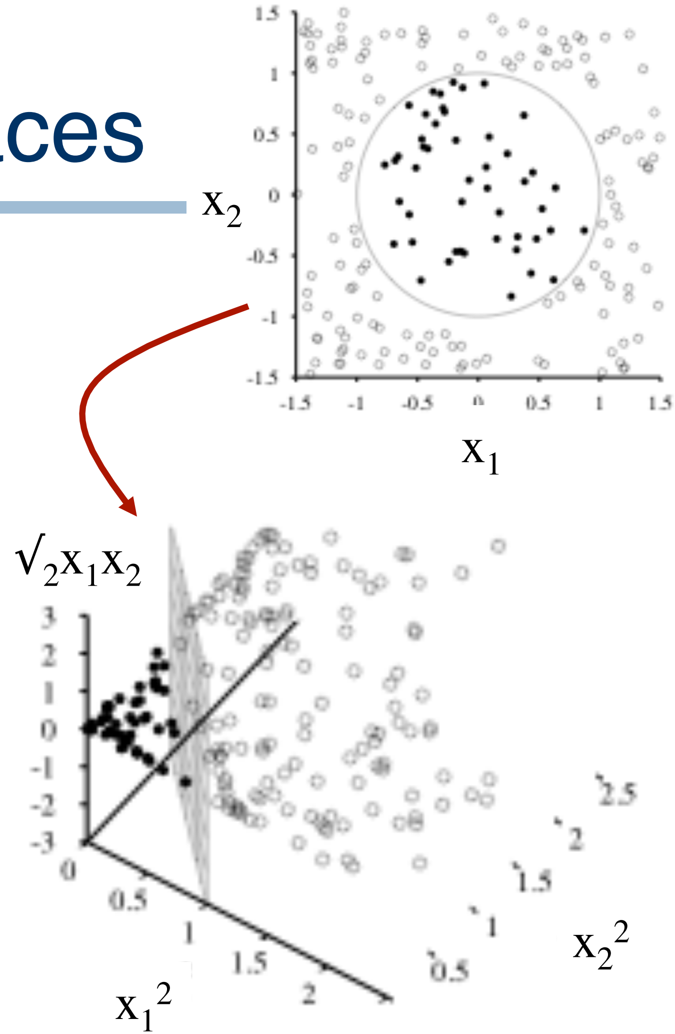
$g(W\,g(Wx)) \approx x$

Support Vector Machines

- Neural networks find AN answer that is suitable, but not necessarily optimal
- Support Vector Machines (SVMs) find the linear separator that is optimal
  - **Optimal**: one that maximizes the margin over all data points
  - **Margin**: the distance to the decision boundary
  - For misclassifications, the margin is negative

Non-linear spaces

- SVMs find linear separators
- For non-linear spaces, need to project to a higher dimensional space



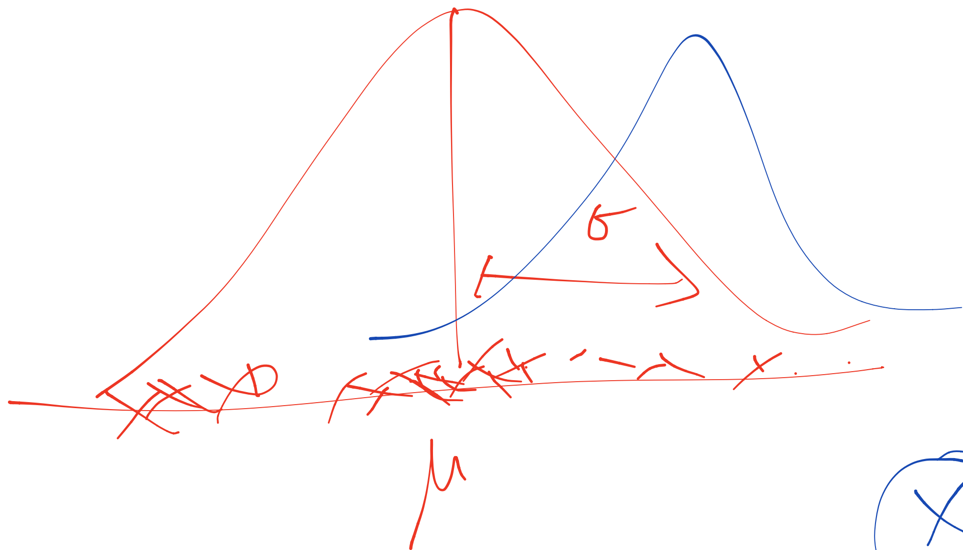
Learning for Gaussians

- Remember Gaussians?
  - mean: μ standard deviation: σ
- For a set of data **x**, we define the log likelihood given N(μ, σ) as

$$L = \sum_{j=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_j-\mu)^2}{2\sigma^2}}$$

$$= N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^{N} \frac{(x_j - \mu)^2}{2\sigma^2}$$

- Question: what values of μ, σ *maximize* the likelihood?
  - Can derive this through calculus



X

$$\underset{\mu,\sigma}{\operatorname{argmax}} \; P_{\mu,\sigma}(X_i^n) = \prod_{i=1}^{n} P(X_i)$$

$$\underset{\mu,\sigma}{\operatorname{argmax}} \; \log P(X_i^n) = \log \prod (P(x_i))$$

$$= \sum \log P(x_i)$$

$$P(x_i) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\log P(x_i) = \log\left(\frac{1}{\sqrt{2\pi}\,\sigma}\right) - \frac{(x-\mu)^2}{2\sigma^2}$$

Maximizing the likelihood

- The maximum is found by taking the derivative and setting it to zero

$$L = N(-\log\sqrt{2\pi} - \log\sigma) - \sum_{j=1}^{N}\frac{(x_j-\mu)^2}{2\sigma^2}$$

$$\frac{\partial L}{\partial \mu} = \frac{1}{\sigma^2}\sum_{j=1}^{N}(x_j-\mu) = 0 \Rightarrow \mu = \frac{\sum_{j=1}^{N} x_j}{N}$$

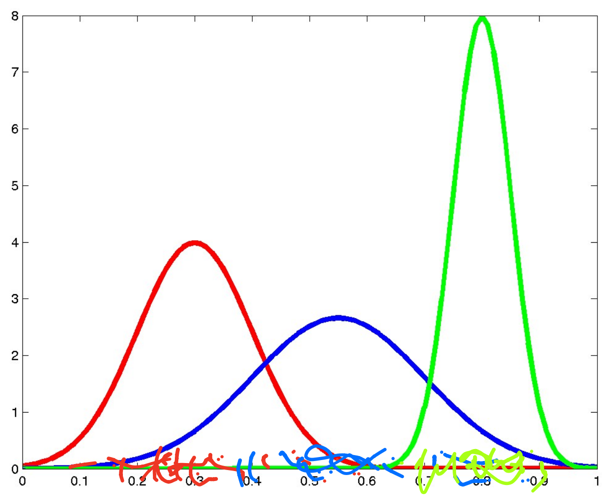
$$\frac{\partial L}{\partial \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{N}(x_j-\mu)^2 = 0 \Rightarrow \sigma = \sqrt{\frac{\sum_{j=1}^{N}(x_j-\mu)^2}{N}}$$

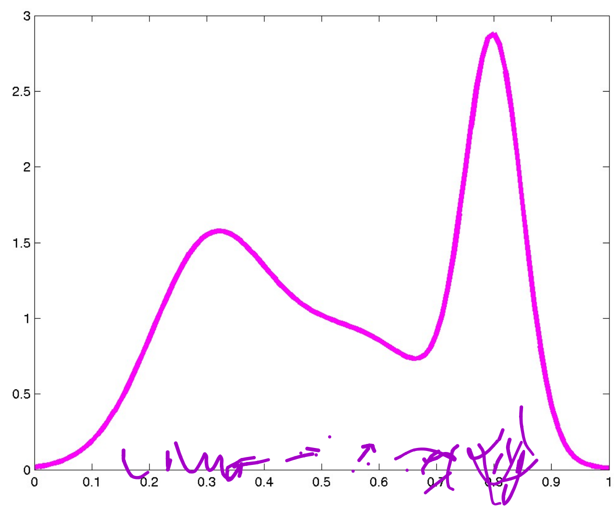
What does this all mean?

- The mean and standard deviation we define here is the one that maximizes the likelihood of the training data
- However, we can do this for any data
  - Example given in section 20.2 for linear Gaussian model

Mixture of Gaussians

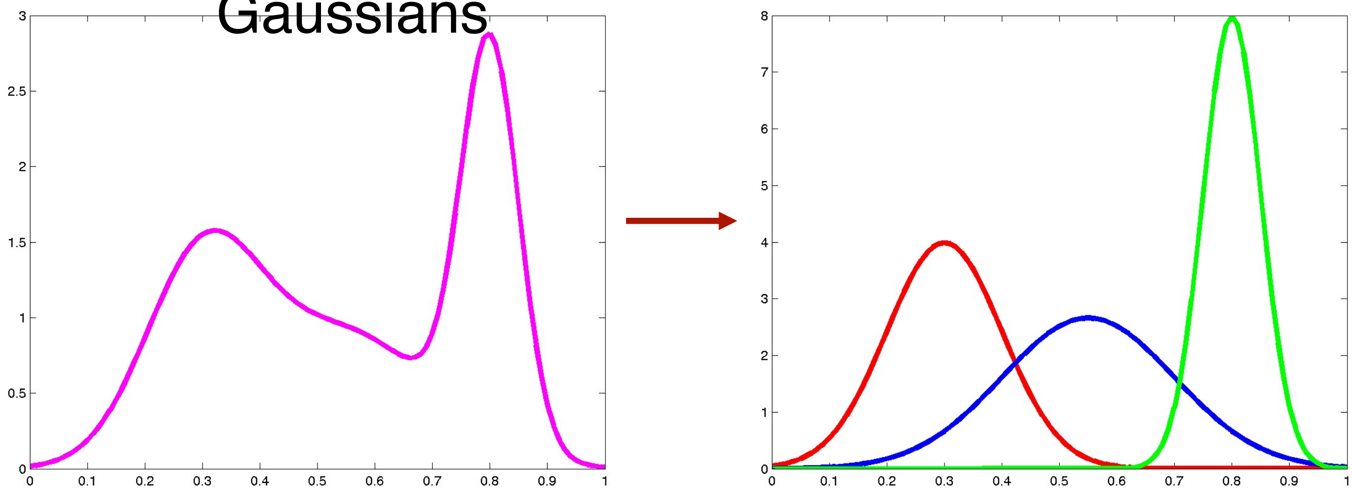
- Imagine if we had 3 Gaussians and mixed them up together





Mixture of Gaussians

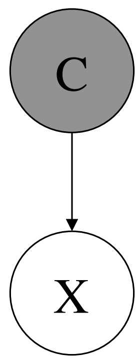
- Can we recover the underlying Gaussians given some data?
  - Each data point is “generated” by one of the Gaussians



Mixture of Gaussians

- MOG is a small Bayes net, where the mixture is a hidden variable

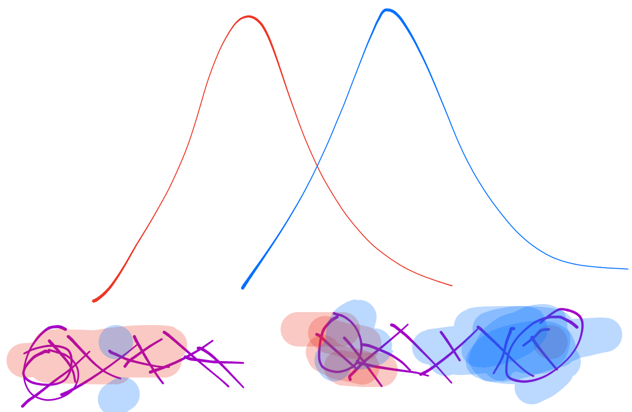
$$P(X) = \sum_{c \in C} P(c)P(X \mid c)$$



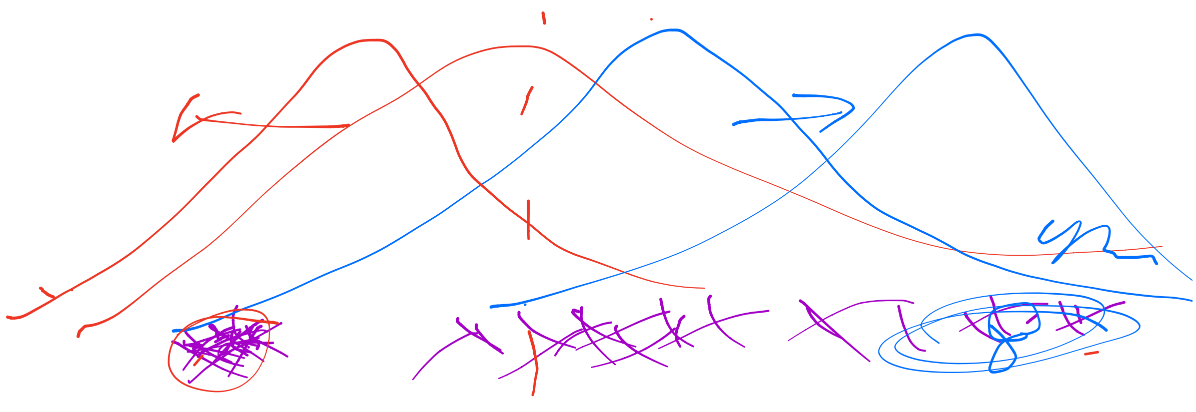
$P(c_1)=a$, $P(c_2)=b$,…

C	P(X|c)
c_1	$N(\mu_1,\sigma_1)$
c_2	$N(\mu_2,\sigma_2)$
…	…

$P(Red, Blue) = \langle .5, .5 \rangle$



$P(Blue|X) = \alpha P(x|Blue) P(Blue)$



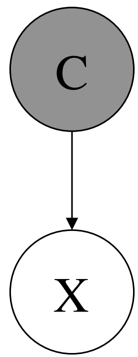
Mixture of Gaussians

- MOG is a small Bayes net, where the mixture is a hidden variable

$$P(X) = \sum_{c \in C} P(c)P(X \mid c)$$

which mixture?

likelihood given mixture

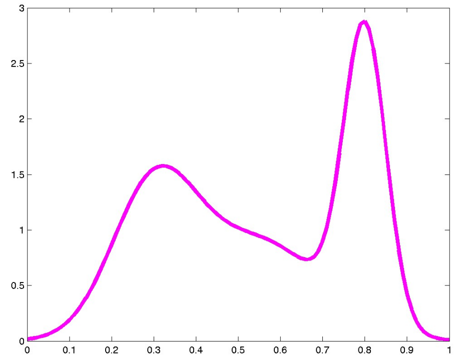


$P(c_1)$=a, $P(c_2)$=b,…

C	P(X|c)
c_1	$N(\mu_1,\sigma_1)$
c_2	$N(\mu_2,\sigma_2)$
…	…

Learning MOGs

- We only have data like this:



- How do we learn which mixture things come from?
  - **Expectation-maximization algorithm**

Expectation-maximization (EM) algorithm

- Two parts, done over and over again
- Part 1: Expectation
  - What's our best guess for every data point as to which cluster it comes from
  - In general, compute the probability of hidden variables
- Part 2: Maximization:
  - Given our expectations, figure out the parameters for the gaussian distributions
  - In general, compute new parameters based on the probability of the hidden variables

MOG: initialization

- Set $P(c_i)$ to be random distributions
  - Make sure that P(all classes) sums to 1
- Set means, standard deviations to be random
  - Helps to get them started in the right area
  - A good initialization is to take the global mean and variance and add/subtract a small random number

MOG: Expectation

- Question: for every point X_j, what is the probability that class_i generated that point?

$$P_{ij}(c_i \mid X_j) = \alpha P(X_j \mid c_i) P(c_i) = P_{ij}$$

$$N_i = \sum_j P_{ij}$$

MOG: Maximization

- For every class, compute a new class prior, mean, and standard deviation

$$\hat{\mu}_i = \frac{\sum_j P_{ij} x_j}{N_i}$$

new mean: weighted average of points assigned to class i

$$\hat{\sigma}_i = \sqrt{\frac{\sum_j P_{ij} x_j^2}{N_i} - \left(\frac{\sum_j P_{ij} x_j}{N_i} \right)^2}$$

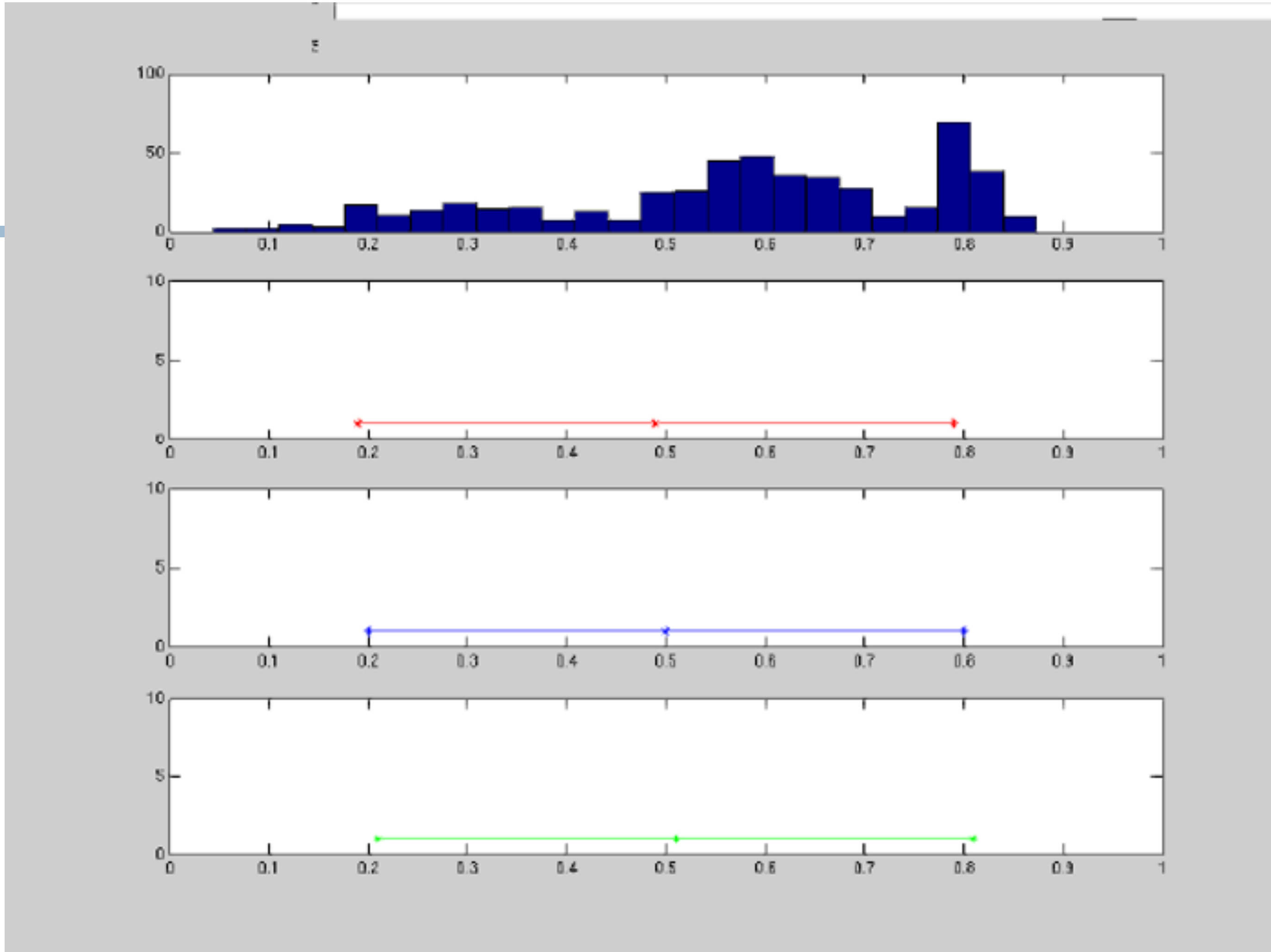
new standard deviation: calculated in same weighted manner

$$\hat{P}(c_i) = \frac{N_i}{\sum_j N_j} = N$$

new class prior: proportion of weighted samples attributed to class

MOG Example

- Data generated from three mixtures
- Initialize three mixtures with
  - even mixture priors
  - similar means (center of line)
  - similar standard deviations (width of line)
- Movie shows progression of mean, standard deviation as a function of EM iteration


100
50
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
10
5
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
10
5
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
10
5
0
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1

General form of EM

- Given:
  - Θ^t, parameters at time t,
  - Z, hidden variables
  - X, observed variables

$$\Theta^{(t+1)} = \arg\max_{\hat{\Theta}} \sum_{z} P(Z = z \mid X, \Theta^t) L(X, Z = z \mid \Theta)$$

- E-step: calculate probabilities of Z
- M-step: calculate most likely Θ

Important points about EM

- At every step, EM is guaranteed not to decrease the likelihood of the data
- May run into local maxima in the likelihood space
  - What the space looks like depends on the problem
  - Local maxima are rare in MOG problems

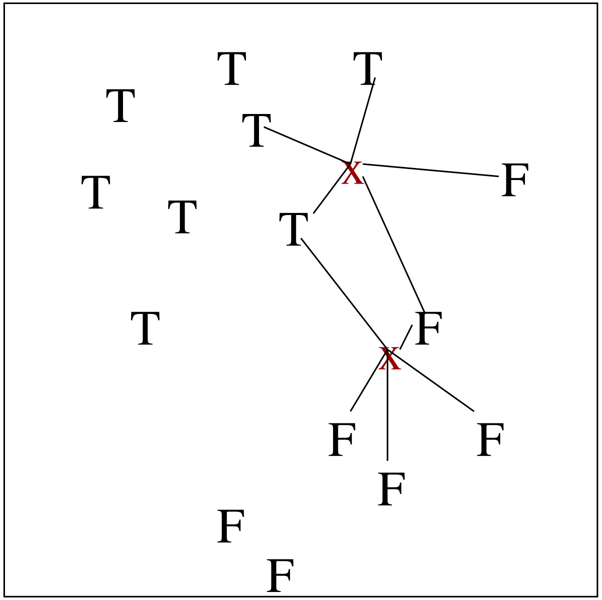
Non-parametric learning

- Neural networks and Gaussians are parametric learners
  - Restricted number of parameters according to a particular form
    - weights, means, variances
- Non-parametric learners use the data directly to derive classifications
  - Nearest neighbors / k-nearest neighbors

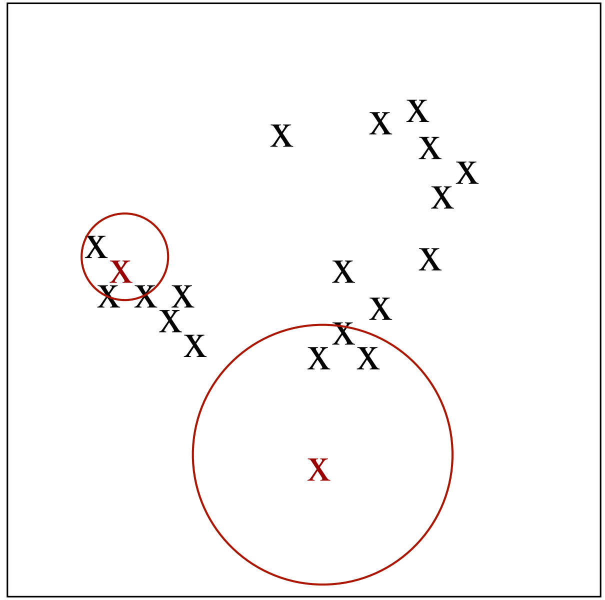
Nearest neighbor
k-nearest neighbor

- Idea: points are likely to be clustered together
  - Classification: similar classes cluster together
  - Probability density: instances likely to cluster together
- To figure out what to do with a point, look at neighboring points
  - Classification: neighbors “vote”
  - Density estimation: how far out do you need to go to get k neighbors?
- As you get more data, it gets harder to decide who your neighbors are

k-nearest neighbor



5-nearest neighbor
classification

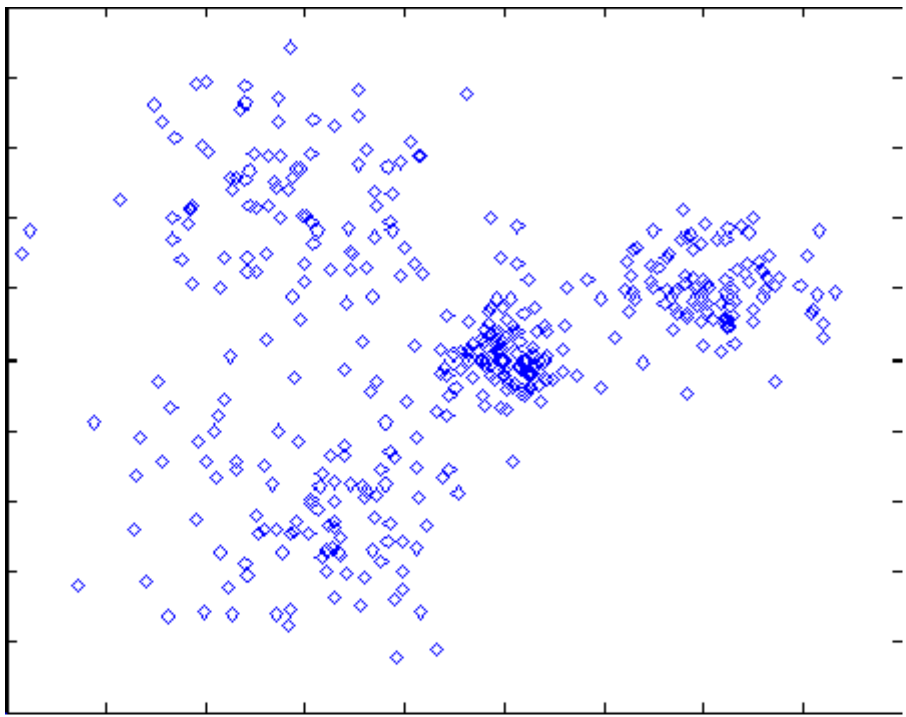


3-nearest neighbor
density estimation

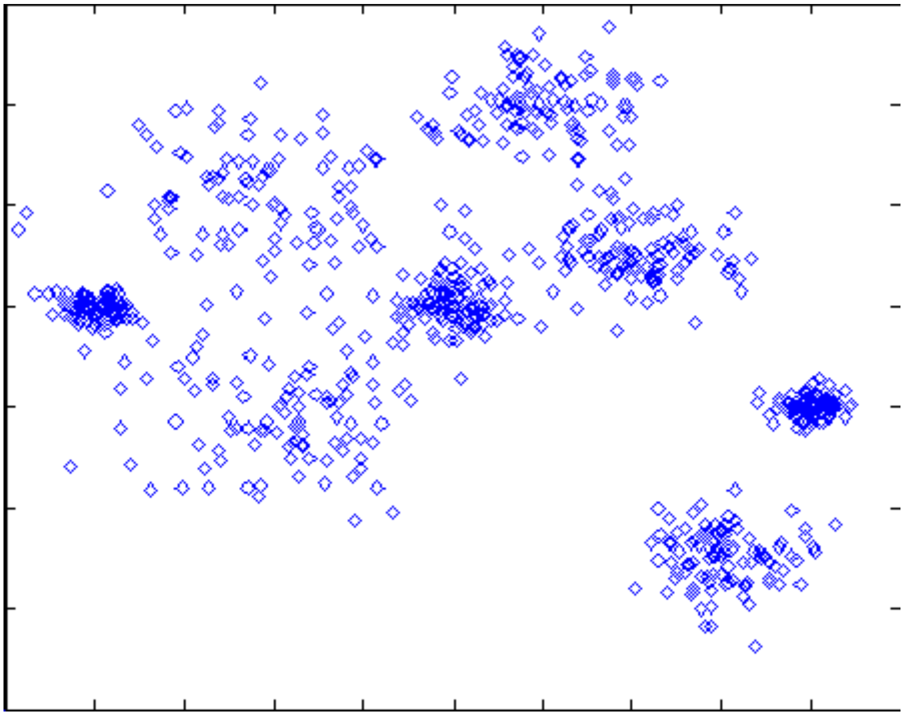
k-means clustering

- Not a classification technique — unsupervised learning
- Similar idea to k-nearest neighbors
  - Choose n cluster mean points randomly
  - Assign each data point to a cluster mean
    - instead of closest neighbors to point, closest mean to point
  - Recalculate mean, iterate to previous step
    - Stop when means don't move around (converge)

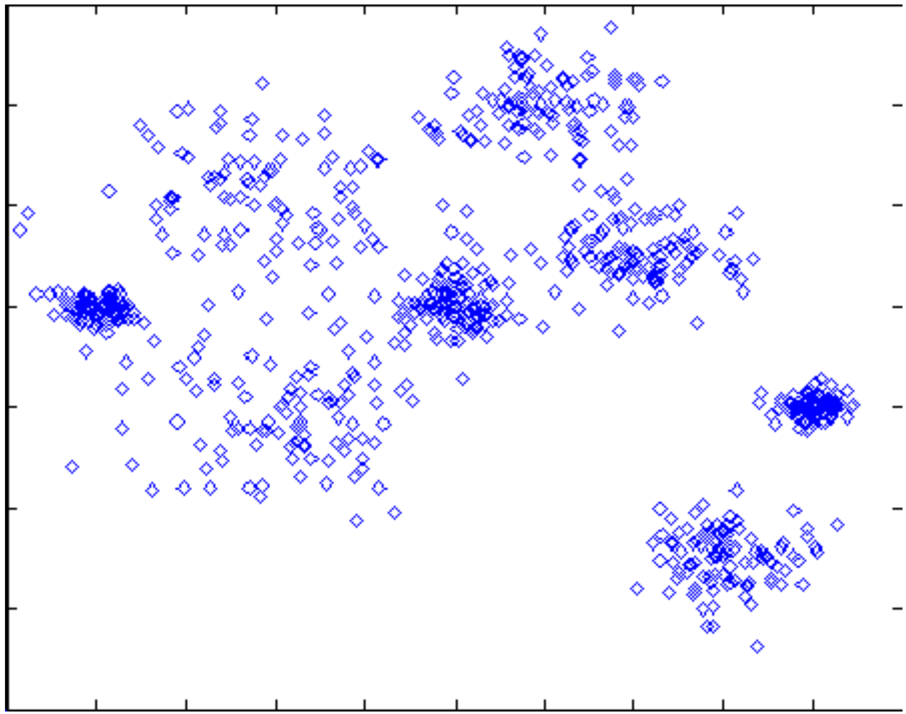
4 means



8 means – version 1



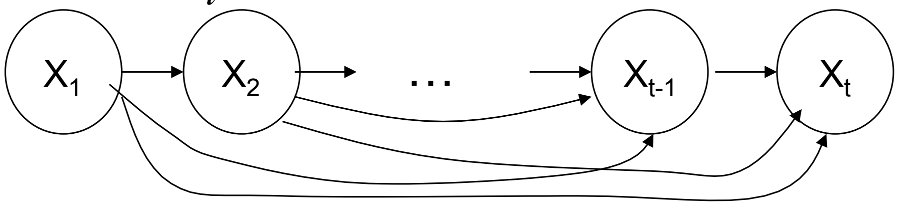
8 means – version 2



Modeling time

- Often we have a sequence of events/ observations/random variables
- We have a distribution (given chain rule):

$$\prod_t P(X_t \mid X_{t-1}, \ldots, X_1)$$



- Examples?

Markov Models

- Model a time sequence of variables
- Markov assumption:

$$P(X_t \mid X_{t-1}, \ldots, X_1) = P(X_t \mid X_{t-1})$$

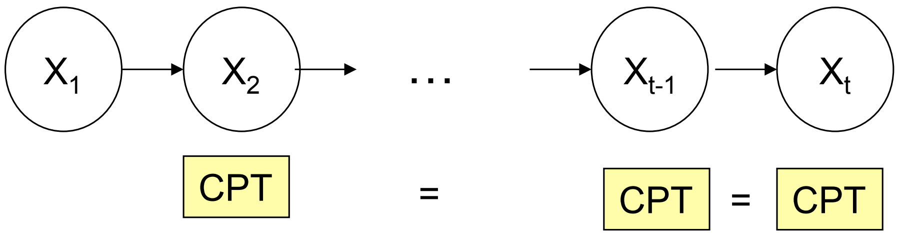
  - Current X only depends on previous X

Markov Models

- Stationary process assumption:

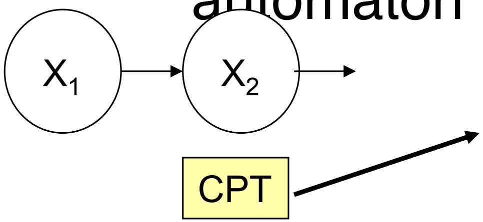
$$P(X_t \mid X_{t-1}) = P(X_{t-1} \mid X_{t-2})$$

  - Relationship between X and successor doesn't change over time



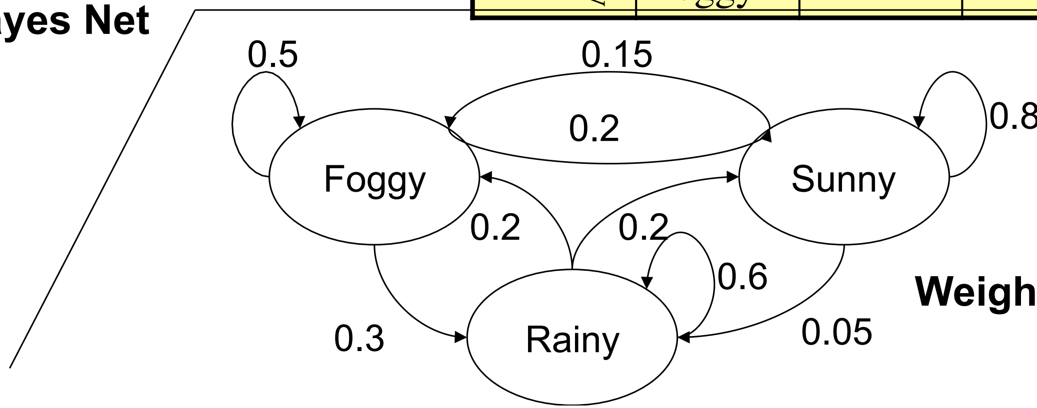
Markov Models

- Often represented as weighted finite state automaton



		Weather tomorrow		
Weather today		Sunny	Rainy	Foggy
	Sunny	0.8	0.05	0.15
	Rainy	0.2	0.6	0.2
	Foggy	0.2	0.3	0.5

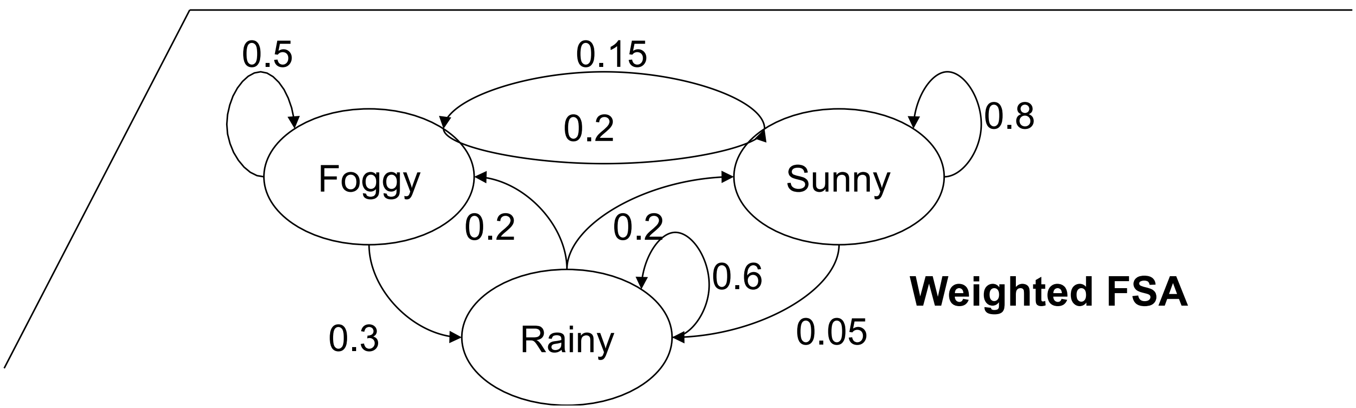
Bayes Net



Weighted FSA

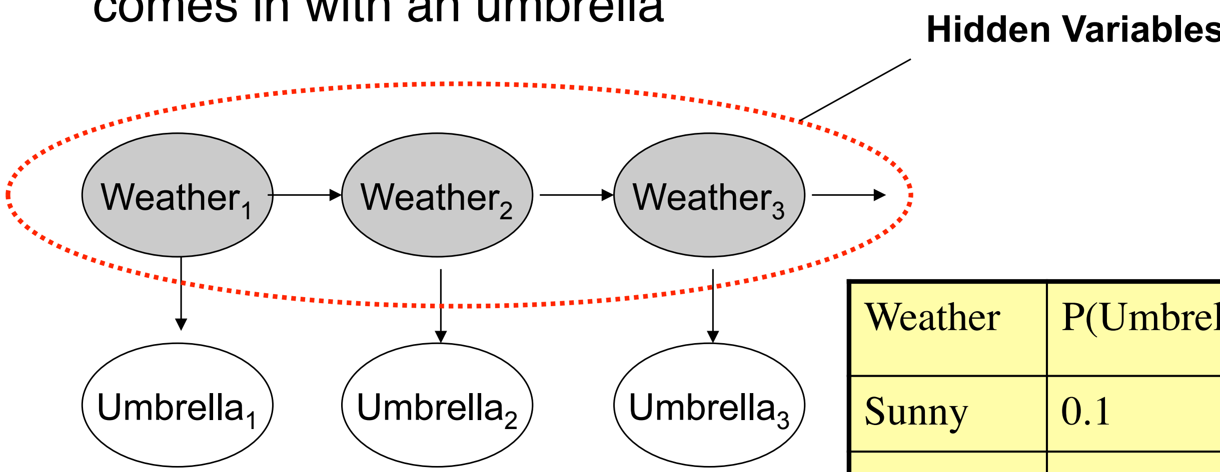
Markov Models

- If it's rainy today, what's the probability that it will be sunny for the next two days?
- If it's rainy today, what's the probability that it will be sunny two days from now?



Hidden Markov Models

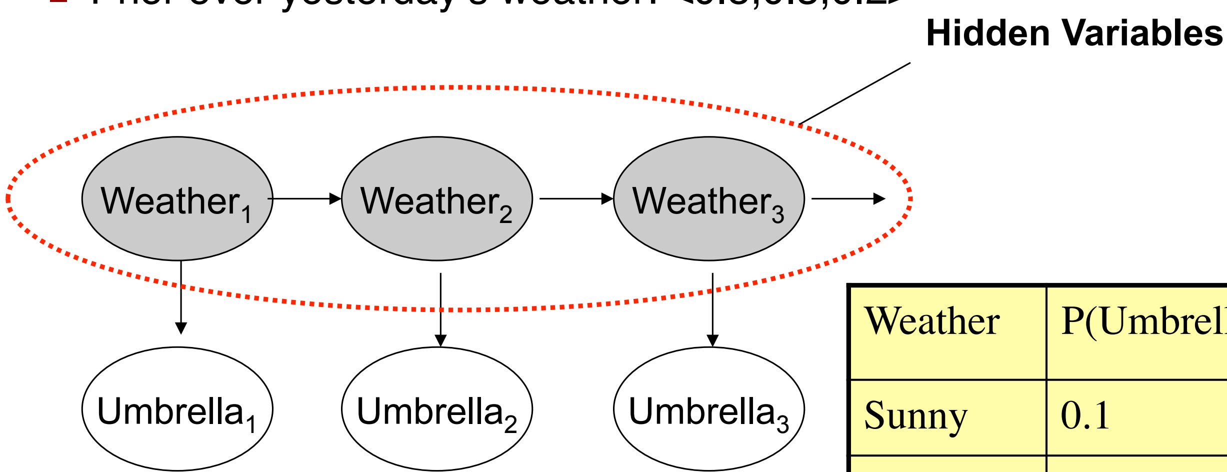
- What if you were locked in a room?
  - Can only tell what's going on by whether someone comes in with an umbrella



Weather	P(Umbrella)
Sunny	0.1
Rainy	0.8
Foggy	0.3

Hidden Markov Models

- If someone walked into a room with an umbrella yesterday and today, what's the probability that it's raining?
  - Prior over yesterday's weather: <0.5,0.3,0.2>

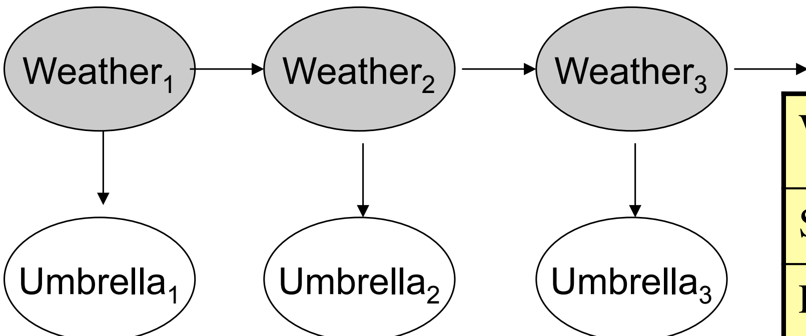


Weather	P(Umbrella)
Sunny	0.1
Rainy	0.8
Foggy	0.3

Hidden Markov Models

- Total probability sequence of hidden variables has compact form:

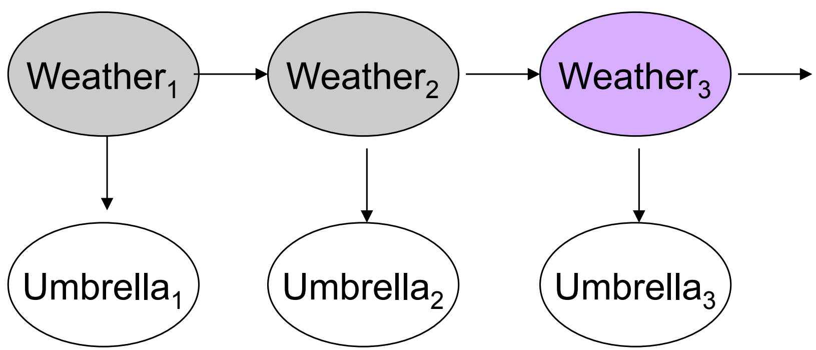
$$P(X_{1..t}, e_{1..t}) = \prod_{i=1}^{t} P(X_i \mid X_{i-1}) P(e_i \mid X_i)$$



Weather	P(Umbrella)
Sunny	0.1
Rainy	0.8
Foggy	0.3

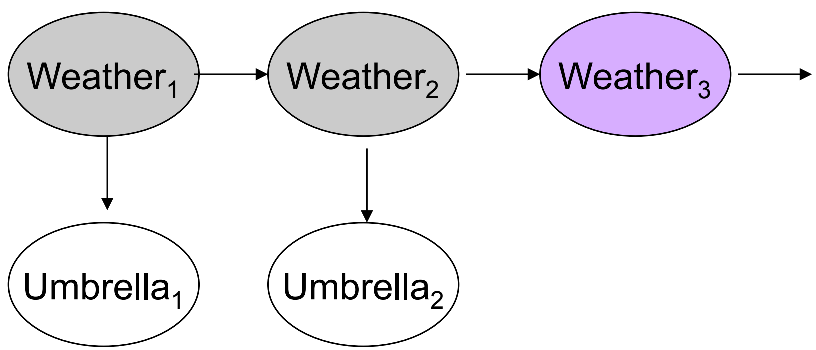
Temporal inference problems

- Filtering/monitoring: Compute $P(X_t|e_{1..t})$
  - What is today's weather like given the umbrella situation for the past three days?
  - Compute the current **belief state** of an agent



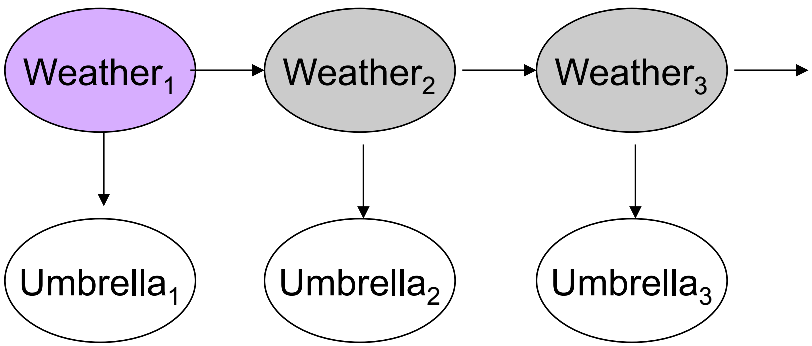
Temporal inference problems

- Prediction: Compute $P(X_{t+k}|e_{1..t})$, k>0
  - What is tomorrow's weather like given the umbrella situation for the past two days?
  - Similar to filtering, except we're looking ahead



Temporal inference problems

- Smoothing: Compute $P(X_{t-k}|e_{1..t})$, $k>0$
  - What is the weather like two days ago, given that the caretaker brought an umbrella the last three days?
  - Given newer evidence, this might change your opinion about past events



Temporal inference problems

- Find the most likely sequence:
  Compute $\mathrm{argmax}_t\, P(X_{1..t} | e_{1..t})$
  - What is the most likely weather sequence over the last three days?
  - Can be computed efficiently using the **Viterbi algorithm**
  - Important in speech recognition, communications