

reflex agent - doesn't think about consequences; bases only on current state  
 planning agent: maintains model; thinks ahead  
 worldstate - all info about state search - info necessary for planning  
 state space graph - states = nodes  
 - edges = actions  
 - directed edges from states to successors  
 search tree - node encodes path from start state to given state  
 Tree Search: partial plan (n) of full, replace on fringe w/ children (n+1)  
 Completeness - guaranteed to find soln  
 optimality - lowest cost path  
 branching factor - increase in # nodes on fringe each time fringe node is discovered.  
 DFS - deepest fringe node LIFO  
 - not complete (cycles)  
 - not optimal (leftmost)  
 -  $O(b^m)$ , depth = m  
 - space  $O(bm)$   
 BFS - shallowest fringe node FIFO  
 - complete only  
 - optimal (leftmost) when all edge weights are =  
 - shallowest soln depth = s time  $O(b^s)$   
 - space  $O(b^s)$   
 UCS - lowest cost fringe node  
 right = path from s to v  
 - complete  
 - optimal, if edges  $\geq 0$   
 - time  $O(b^{C^*})$   
 $C^*$  is optimal path cost, minimal cost between nodes = e  
 - space:  $O(b^{C^*})$   
 Greedy  
 - uses heuristic  
 - not complete/optimal  
 A\*  
 - heuristic: path cost + heuristic  
 - complete + optimal  
 Admissibility  $\Rightarrow$  optimal A\* tree search  
 $h^*(n)$  = true optimal forward cost from node n  
 $h_n, 0 \leq h(n) \leq h^*(n)$   
 "underestimates" true cost  
 Graph Search  
 - kept track of expanded nodes  
 - ensure next node isn't in closed set  
 Consistency  $\Rightarrow$  graph search optimality/completeness  
 - underestimate difference in path cost between two nodes  
 $\forall A, C, h(A) - h(C) \leq cost(A, C)$   
 consistency  $\Rightarrow$  admissibility  
 Heuristic A is dominant over B if estimated goal distance for A is greater than estimated goal distance for B for every node in state space graph  
 Max (admissible/consistent) = admissible/consistent  
 Search problems: state space, successor fn, start state, goal test  
 consistency  $\Rightarrow$  e value along path never decreases

CSPs  
 - variables:  $X_1, \dots, X_n$   
 - domain  
 - constraints  
 Unary Constraint: single var  
 binary - two vars  
 Backtracking - assign only if value violates no constraints  
 Filtering - pre-dominates of assigned variable by removing values that lead to backtracking  
 Forward Checking - pre-dominates of all vars that share a constraint w/ current var.  
 Arc Consistency: catching stuff early, remove value for  $X_i$  if  $X_i$  is violated by  $X_j$  if  $X_i \rightarrow X_j$  is violated by  $X_j$  if  $X_j$  is violated by  $X_i$  to queue  
 AC3:  $O(ed^3)$ , e = # arcs, d = size (largest domain)  
 AC: fewer backtracks, more holistic, more computation  
 Minimum Remaining Values (MRV) C1  
 - select variable w/ fewest remaining valid values to assign next  
 Least Constraining Value (LCV) C2  
 - choose which value to assign next, select value that prunes fewest values from domains of remaining unassigned values  
 Local Search: selection that violates most constraints and reject to value that violates fewest constraints (min-conflicts heuristic)  
 - incomplete, suboptimal  
 Minimax  
 - opponent behaves optimally  
 Utility - best possible outcome  
 terminal utility - deterministic known value  
 infinite game property  
 if agent controlled state,  $V(s) = \max_{s'} V(s')$   
 if opp. controlled state,  $V(s) = \min_{s'} V(s')$   
 terminal states  $V(s)$  known.  
 - behaves similarly to DFS:  $O(b^m)$   
 $\alpha$ -B pruning:  $\geq O(b^{m/2})$   
 - m is depth of terminal nodes  
 Eval fn: take in state and output estimate of true minimax value of node  
 - used in depth-limited minimax  
 - removes guarantee of optimal play  
 Eval fn:  $= w_1 f_1(s) + \dots + w_n f_n(s)$   
 $f_i$ : feature from input state  
 $w_i$ : weight of  $f_i$   
 - higher scores for better positions  
 Expectimax  
 - chance nodes: average case  
 $\forall$  chance states,  $V(s) = \sum_{s'} P(s'|s) V(s')$   
 - not much pruning except when ? known, finite bounds on possible node vals.  
 General Games  
 MDPs  
 - set of states S  
 - set of actions A  
 - start state  
 -  $\geq 1$  terminal state  
 - discount factor  $\gamma$   
 - transition fn  $T(s, a, s')$   
 - reward fn  $R(s, a, s')$   
 s - states / action states = chance nodes  
 $C(s, a)$   
 - discounted utility if finite-valued  
 if  $\gamma < 1$  Markov Property  
 $T(s, a, s') = P(s' | s, a)$

Solving MDP = find optimal policy  $V^* = \max$  expected utility over all possible actions from s  
 $\pi^*: S \rightarrow A$   
 $V^*(s)$  - optimal value of s (starting in s)  
 $Q^*(s, a)$  - optimal value of s, if taking action a  
 Bellman Eqn - condition for optimality  
 $V^*(s) = \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$   
 $Q^*(s, a) = \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$   
 Expected utility  
 utility attained by acting optimally, not acting in  $s'$  from  $s$ -state (s, a)  
 Value Iteration - run until convergence (OCS2A)  
 initialize  $V_0(s) = 0 \forall s \in S$   
 $\forall s \in S, V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$   
 Convergence:  $\forall s \in S, V_k(s) = V^*(s) = V_{k+1}(s)$   
 Policy Extraction - optimal policy if values are optimal  
 $\forall s \in S, \pi^*(s) = \arg \max_a Q^*(s, a) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]$   
 Policy Iteration  
 - policy converges faster than values  
 1) define initial policy  
 2) until convergence:  
 - policy evaluation: compute  $V^\pi(s) \forall s$  in states when following  $\pi$   
 $V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$   
 $V^\pi$  = policy at iteration i.  
 only one action at each iteration, no need for max.  
 $V^\pi(s) = \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi(s')]$  } policy eval  
 - policy improvement  
 $\pi_{i+1}(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^\pi(s')]$  } policy extraction  
 $\pi_{i+1} = \pi_i \Rightarrow$  convergence  $\Rightarrow \pi_{i+1} = \pi_i = \pi^*$   
 RL  
 MDP: define planning  $\rightarrow$  knowledge of transition and reward fn  $C(s, a, s', r) :=$  Example  
 Model Based - estimate transition/reward fns  $\rightarrow$  solve MDP  
 "Free": estimate values/q-values directly  
 - get samples from episode, estimate transition + reward } Model based  
 - expensive to maintain counts for every  $(s, a, s')$  seen  
 Model Free  
 Direct Eval  
 - passive (same policy)  
 - fix policy  $\pi$  and have agent experience episodes from episodes  
 - maintain total utility obtained from each state, # visits per state  
 - value = utility / visits  
 - one "bad" sample can logote process  
 need optimal policy to converge to  $V^*(s)$ .  
 TD (Temporal Difference)  
 - passive  
 - learn from every experience  
 - uses exponential moving average  
 Sample =  $R(s, \pi(s), s') + \gamma V^\pi(s')$ . new estimate for  $V^\pi(s)$   
 $V^\pi(s) \leftarrow (1 - \alpha) V^\pi(s) + \alpha \text{sample}$ ,  $\alpha$  = learning rate  $0 \leq \alpha \leq 1$   
 - typically start  $\alpha$  at 1, slowly zero out  
 - older samples given exponentially less weight  
 Q-Learning  
 - find out an optimal policy must visit all state, action pairs.  
 Q-value iteration:  
 $Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q_k(s', a')]$   
 - max changes bc we transition before selecting new action in a state.  
 Q-value sample:  $R(s, a, s') + \gamma \max_{a'} Q(s', a')$   
 $Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha \text{sample}$  } update  
 $\rightarrow$  off-policy learning: learn optimal policy by taking suboptimal or random actions  
 Approximate Q-Learning  
 - learn about few general situations and extrapolate.  
 - state represented as feature vector  
 $V(s) = w_1 \cdot f_1(s) + \dots + w_n \cdot f_n(s) = \vec{w} \cdot \vec{f}(s)$   
 $Q(s, a) = w_1 \cdot f_1(s, a) + \dots + w_n \cdot f_n(s, a)$  } update  
 $Q(s, a) \leftarrow Q(s, a) + \alpha \text{difference}$   
 $\text{difference} = [R(s, a, s') + \gamma \max_{a'} Q(s', a')] - Q(s, a)$  same for all weights  
 $w_i \leftarrow w_i + \alpha \text{difference} \cdot f_i(s, a)$  } update  
 - store only single weight vector, compute Q-values as needed  
 - more memory efficient.

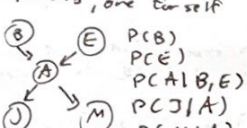


E-greedy  
 - explore w/ prob  $\epsilon$   
 - exploit w/ prob  $(1-\epsilon)$

Exploration Functions  
 $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha [R(s,a) + \max_{a'} V(s,a')]$   
 e.g.  $Q(s,a) = \frac{Q(s,a)}{N(s,a)}$

$K :=$  predetermined,  $N(s,a) :=$  # time  $(s,a)$  has been visited

Bayes Nets

- DAG  
 - node is conditionally independent of all ancestor nodes in graph given all parents  
 - arrows represent conditional prob dist.  
 - node:  $PC(X|A_1, \dots, A_n)$ ;  $A_i$  is its parent of  $X$   
 - each node has a CPT  $C(x|c)$   
 - n parents, one for self  
 e.g.  
  
 $P(S)$   
 $P(B)$   
 $P(E)$   
 $P(A|B, E)$   
 $P(C|A)$   
 $P(M|A)$

Inference - calculate joint PDF for some query var based on some set of observed var

Prior Sampling - sample many times BUT: generate too many samples

Rejection: reject some inconsistent w/ evidence

D-Separation

- all tuples in a path are active  
 $\Rightarrow$  path is active, d-connects  $X$  to  $Y$   
 $\Rightarrow$   $X$  is not conditionally independent of  $Y$  given observed nodes  
 - all paths  $X-Y$  are inactive  
 $\Rightarrow$   $X$  and  $Y$  are CI given observed  
 1 tuple is inactive

Active

not guaranteed

C2 X12

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

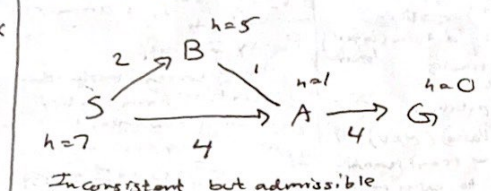
$P(X|Y, Z) = P(X|Y)$

$P(X|Y, Z) = P(X|Y)$

Marginal Distribution

$P(S) = \sum_x P(x, S)$   
 $P(a|b) = \frac{P(a, b)}{P(b)}$   
 $P(x, y) = P(x|y)P(y) = P(y|x)P(x)$   
 $P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$   
 $P(x_1, \dots, x_n) = \prod P(x_i|x_1, \dots, x_{i-1})$   
 $P(x|y) = \frac{P(y|x)P(x)}{P(y)}$   
 $x$  and  $y$  are independent if  $P(x, y) = P(x)P(y)$ ,  $\forall (x, y)$   
 $x$  and  $y$  are CI given  $z$  iff  $P(x, y|z) = P(x|z)P(y|z)$   
 $\forall x, y, z : P(x, y|z) = P(x|z)P(y|z)$

CSP: r vars, d possible val - backtracking may backtrack: O(d<sup>n</sup>)



DFS, BFS, UCS  
 - search is equivalent if edge weights are changed in same way (multiplication).  
 - nodes have costs are added/subtracted by same number  
 - find by doing sum  
 $h(j) - h(i)$  is telescoping  
 - heuristic must return 0 in goal state  
 - consistent!  
 doesn't decrease from state to state by more than it actually costs to go from state to state  
 - fully connected (complete) graph can represent joint distribution  
 - marginalization renders neighbors of marginalized out var dependent

condition - ancestors dependent

valid joint distr.

- each variable appears on left of 1 once

- no cyclic conditioning

nondeterministic transition  $\Rightarrow$  iteration

d-B

d: MAX's best option on path to root

B: MIN's best option on path to root

def max\_value(state, d, B):

initialize  $v = -\infty$

for each suc of state:

$v = \max(v, \text{value}(\text{suc}, d, B))$

if  $v \geq B$  return  $v$

$d = \arg \max(d, v)$

return  $v$

min

max

min

max

min

max

min

max

min

max

min

max

min

max

min

max

min

max

min

max

$X \perp Y | Z$

$P(X|Y, Z) = P(X|Z)$

$P(X, Y, Z) = \frac{P(X, Z)P(Y, Z)}{P(Z)}$

d-B pruning:

- node is pruned from under max node if it knows that min node above it has smaller value to pick than the value max node just found

- node is pruned from min node if it knows that max node above it has larger value to pick than the value min node just found

Infinite state space

$h(G) = 0$   
 $G \leftarrow S \xrightarrow{1} S_1 \xrightarrow{0.5} S_2 \xrightarrow{0.25} S_3 \xrightarrow{0.125} S_4 \xrightarrow{0.0625} S_5 \xrightarrow{0.03125} S_6 \xrightarrow{0.015625} S_7 \xrightarrow{0.0078125} S_8 \xrightarrow{0.00390625} S_9 \xrightarrow{0.001953125} S_{10} \xrightarrow{0.0009765625} S_{11} \xrightarrow{0.00048828125} S_{12} \xrightarrow{0.000244140625} S_{13} \xrightarrow{0.0001220703125} S_{14} \xrightarrow{0.00006103515625} S_{15} \xrightarrow{0.000030517578125} S_{16} \xrightarrow{0.0000152587890625} S_{17} \xrightarrow{0.00000762939453125} S_{18} \xrightarrow{0.000003814697265625} S_{19} \xrightarrow{0.0000019073486328125} S_{20} \xrightarrow{0.00000095367431640625} S_{21} \xrightarrow{0.000000476837158203125} S_{22} \xrightarrow{0.0000002384185791015625} S_{23} \xrightarrow{0.00000011920928955078125} S_{24} \xrightarrow{0.000000059604644775390625} S_{25} \xrightarrow{0.0000000298023223876953125} S_{26} \xrightarrow{0.00000001490116119384765625} S_{27} \xrightarrow{0.000000007450580596923828125} S_{28} \xrightarrow{0.0000000037252902984619140625} S_{29} \xrightarrow{0.00000000186264514923095703125} S_{30} \xrightarrow{0.000000000931322574615478515625} S_{31} \xrightarrow{0.0000000004656612873077392578125} S_{32} \xrightarrow{0.00000000023283064365386962890625} S_{33} \xrightarrow{0.000000000116415321826934814453125} S_{34} \xrightarrow{0.0000000000582076609134674072265625} S_{35} \xrightarrow{0.00000000002910383045673370361328125} S_{36} \xrightarrow{0.000000000014551915228366851806640625} S_{37} \xrightarrow{0.0000000000072759576141834259033203125} S_{38} \xrightarrow{0.00000000000363797880709171295166015625} S_{39} \xrightarrow{0.000000000001818989403545856475830078125} S_{40} \xrightarrow{0.0000000000009094947017729282379150390625} S_{41} \xrightarrow{0.00000000000045474735088646411895751953125} S_{42} \xrightarrow{0.000000000000227373675443232059478759765625} S_{43} \xrightarrow{0.0000000000001136868377216160297393798828125} S_{44} \xrightarrow{0.00000000000005684341886080801486968994140625} S_{45} \xrightarrow{0.000000000000028421709430404007434844970703125} S_{46} \xrightarrow{0.0000000000000142108547152020037174224853515625} S_{47} \xrightarrow{0.00000000000000710542735760100185871124267578125} S_{48} \xrightarrow{0.000000000000003552713678800500929355621337890625} S_{49} \xrightarrow{0.0000000000000017763568394002504646778106689453125} S_{50} \xrightarrow{0.00000000000000088817841970012523233890533447265625} S_{51} \xrightarrow{0.000000000000000444089209850062616169452667236328125} S_{52} \xrightarrow{0.0000000000000002220446049250313080847263336181640625} S_{53} \xrightarrow{0.00000000000000011102230246251565404236316680908203125} S_{54} \xrightarrow{0.000000000000000055511151231257827021181583404541015625} S_{55} \xrightarrow{0.0000000000000000277555756156289135105907917022705078125} S_{56} \xrightarrow{0.00000000000000001387778780781445675529539585113525390625} S_{57} \xrightarrow{0.000000000000000006938893903907228377647697925567626953125} S_{58} \xrightarrow{0.0000000000000000034694469519536141888238489627838134765625} S_{59} \xrightarrow{0.00000000000000000173472347597680709441192448139190673828125} S_{60} \xrightarrow{0.000000000000000000867361737988403547205596240695953369140625} S_{61} \xrightarrow{0.0000000000000000004336808689942017736027981203479766845703125} S_{62} \xrightarrow{0.00000000000000000021684043449710088680139906017398834228515625} S_{63} \xrightarrow{0.000000000000000000108420217248550443400699530086994171142578125} S_{64} \xrightarrow{0.0000000000000000000542101086242752217003497650434970855712890625} S_{65} \xrightarrow{0.00000000000000000002710505431213761085017488252174854278564453125} S_{66} \xrightarrow{0.000000000000000000013552527156068805425087441260874271392822265625} S_{67} \xrightarrow{0.0000000000000000000067762635780344027125437206304371356964111328125} S_{68} \xrightarrow{0.00000000000000000000338813178901720135627186031521856784820556640625} S_{69} \xrightarrow{0.000000000000000000001694065894508600678135930157609283924102783203125} S_{70} \xrightarrow{0.0000000000000000000008470329472543003390677950788046419620513916015625} S_{71} \xrightarrow{0.00000000000000000000042351647362715016953389753940232098102569580078125} S_{72} \xrightarrow{0.000000000000000000000211758236813575084766948769701160490512847900390625} S_{73} \xrightarrow{0.0000000000000000000001058791184067875423834743848505802452564239501953125} S_{74} \xrightarrow{0.00000000000000000000005293955920339377119171723742529012262821197509765625} S_{75} \xrightarrow{0.000000000000000000000026469779601696885595858618712645061314105987548828125} S_{76} \xrightarrow{0.0000000000000000000000132348898008484427979293093563225306570529937744140625} S_{77} \xrightarrow{0.00000000000000000000000661744490042422139896465467816126532852649688720703125} S_{78} \xrightarrow{0.000000000000000000000003308722450212110699482327339080632664263248443603515625} S_{79} \xrightarrow{0.0000000000000000000000016543612251060553497411636695403163321316242218017578125} S_{80} \xrightarrow{0.0000000000000000000000008271806125530276748705818347701581660658121109008828125} S_{81} \xrightarrow{0.00000000000000000000000041359030627651383743529091738507908303290605545044140625} S_{82} \xrightarrow{0.000000000000000000000000206795153138256918717645458692539541516453027725220703125} S_{83} \xrightarrow{0.0000000000000000000000001033975765691284593588227293462697707582265138626103515625} S_{84} \xrightarrow{0.00000000000000000000000005169878828456422967941136467313488537911325693130517578125} S_{85} \xrightarrow{0.000000000000000000000000025849394142282114839705682336567442689556628465652587890625} S_{86} \xrightarrow{0.0000000000000000000000000129246970711410574198528411682837213447783142328262939453125} S_{87} \xrightarrow{0.00000000000000000000000000646234853557052870992642058414186067238915711641314692265625} S_{88} \xrightarrow{0.000000000000000000000000003231174267785264354993210292073558285593208558206573461328125} S_{89} \xrightarrow{0.00000000000000000000000000161558713389263217749660514603677914279660427910328671875} S_{90} \xrightarrow{0.0000000000000000000000000008077935669463160887483025730183895713983021395516406564453125} S_{91} \xrightarrow{0.0000000000000000000000000004038967834731580443741512865091947859966510697758203282265625} S_{92} \xrightarrow{0.00000000000000000000000000020194839173657902218707564325459739299832553488791016411328125} S_{93} \xrightarrow{0.000000000000000000000000000100974195868289511093537821627298696499162767443955082056640625} S_{94} \xrightarrow{0.0000000000000000000000000000504870979341447555467719108136493482495813837219775410283203125} S_{95} \xrightarrow{0.00000000000000000000000000002524354896707237777338595540682467412479069186098877051416015625} S_{96} \xrightarrow{0.000000000000000000000000000012621774483536188886692977703412337062395345930494385257080078125} S_{97} \xrightarrow{0.0000000000000000000000000000063108872417680944433464888501706185311976729652471926285400390625} S_{98} \xrightarrow{0.00000000000000000000000000000315544362088404722167324442508530926559883648262359631427001953125} S_{99} \xrightarrow{0.000000000000000000000000000001577721810442023610836622212542654632799418241311798157135009765625} S_{100} \xrightarrow{0.0000000000000000000000000000007888609052210118054183111062713273163997091206558990785675048828125} S_{101} \xrightarrow{0.00000000000000000000000000000039443045261050590270915555313566365819985456032794953928375244140625} S_{102} \xrightarrow{0.000000000000000000000000000000197215226305252951354577776567831829099927280163974769641876220703125} S_{103} \xrightarrow{0.0000000000000000000000000000000986076131526264756772888882839159145499636400819873848209381103515625} S_{104} \xrightarrow{0.00000000000000000000000000000004930380657631323783864444414195795727498182004099369241046905517578125} S_{105} \xrightarrow{0.000000000000000000000000000000024651903288156618919322222070978978637490910020496846205234527587890625} S_{106} \xrightarrow{0.0000000000000000000000000000000123259516440783094596611110354894893187454550102484231026172637939453125} S_{107} \xrightarrow{0.00000000000000000000000000000000616297582203915472983055551774474465937272750512421155130863189697265625} S_{108} \xrightarrow{0.000000000000000000000000000000003081487911019577364915277758872372329686363752562105775654315948486328125} S_{109} \xrightarrow{0.0000000000000000000000000000000015407439555097886824576388794361861648431818762810528878271579742431640625} S_{110} \xrightarrow{0.00000000000000000000000000000000077037197775489434122881943971809308242159093814052644391357898712158203125} S_{111} \xrightarrow{0.0000000000000$



run sampling - normal sampling  
 Rejection - early reject sample inconsistent w evidence

Likelihood Weighting - manually set all variables equal to evidence in grey  
 - use weight for each sample!  
 $P(c | \text{sample})$

Iterate thru vars:  
 - sample value of var = evidence  
 - change weight of sample if var seen

Elitism - first set all vars to some random val  
 - repeatedly pick one var at a time, clear val, resample it gives info about all other assigned values

Decision between  
 - chance nodes  $\circ$   
 - action  $\square$   
 - utility  $\diamond$

$EV(a|e) = \sum_{x_1, \dots, x_n} P(x_1, \dots, x_n | e) U(a, x_1, \dots, x_n)$   
 action  $a$ , evidence  $e$ , chance nodes  $x_1, \dots, x_n$   
 $MEU = \max_a EV(a|e)$

VPI: amount MEU is expected to increase if observe new evidence  
 Expected value of MEU:  
 $MEU(e, e') = \sum_{e'} P(e' | e) MEU(e, e')$   
 $MEU(e, e') = \max_a \sum_{e'} P(e' | e) U(a, e')$

$VPI(e' | e) = MEU(e, e') - MEU(e)$   
 - nonnegative: always isolated value  
 - can always be zero  
 $VPI(e_j | e_k) \neq VPI(e_k | e_j)$   
 - observing  $e_j$  might cause us to not care about  $e_k$

- order independent: order of observations doesn't matter  
 $VPI(e_j | e_k) = VPI(e_j | e) + VPI(e_k | e, e_j)$   
 $= VPI(e_k | e) + VPI(e_j | e, e_k)$

Markov Assumption  
 - chain-like, finite length Bayes net  
 - time dependent  
 - Markov (memoryless) properties  
 - stationary transition model

represented w/  $Pr(W_0)$ ,  $Pr(W_{t+1} | W_t)$   
 MiniFind Algo  
 $Pr(W_{t+1}) = \sum_{W_t} Pr(W_t) Pr(W_{t+1} | W_t)$   
 $= \sum_{W_t} Pr(W_{t+1} | W_t) Pr(W_t)$

→ look at prob distr at time  $t$ , advance model by 1 timestep using  $Pr(W_{t+1} | W_t)$

Stationary Distr  
 $Pr(W_{t+1}) = Pr(W_t) = \sum_{W_t} Pr(W_{t+1} | W_t) Pr(W_t)$   
 solve w/ system:  $\forall i, Pr(W_{t+1}) = Pr(W_t)$

HMM  
 - allows us to observe evidence at each timestep  
 $W_0 \rightarrow W_1 \rightarrow W_2 \rightarrow W_3 \rightarrow$   
 $\downarrow \quad \downarrow \quad \downarrow$   
 $F_1 \quad F_2 \quad F_3$

$W_t$ : state var  
 $F_t$ : evidence var  
 $F_t \perp\!\!\!\perp W_0, \dots, W_{t-1} | W_t$   
 $W_t \perp\!\!\!\perp \{W_0, \dots, W_{t-1}, F_1, \dots, F_{t-1}\} | W_{t-1}$   
 $F_t \perp\!\!\!\perp \{W_0, \dots, W_{t-1}, F_1, \dots, F_{t-1}\} | W_t$

Stationary:  $Pr(W_{t+1} | W_t) = Pr(W_t | W_{t-1})$   
 $Pr(F_t | W_t)$  - sensor model

Belief Distr:  $BC(W_t) = Pr(W_t | F_1, \dots, F_t)$  all evidence observed  
 $B'(W_t) = Pr(W_t | F_1, \dots, F_t)$   $F_1, \dots, F_t$  obs.

Find Algo  
 $B'(W_{t+1}) = Pr(W_{t+1} | F_1, \dots, F_{t+1}) = \sum_{W_t} Pr(W_{t+1} | W_t, F_{t+1}) Pr(W_t | F_1, \dots, F_t)$   
 $= \sum_{W_t} Pr(W_{t+1} | W_t, F_{t+1}) Pr(W_t | F_1, \dots, F_t)$   
 $= \sum_{W_t} Pr(W_{t+1} | W_t) Pr(W_t | F_1, \dots, F_t)$

$BC(W_{t+1}) = Pr(W_{t+1} | F_1, \dots, F_{t+1}) = \sum_{W_t} Pr(W_{t+1} | W_t, F_{t+1}) BC(W_t)$   
 $= \sum_{W_t} Pr(W_{t+1} | W_t) BC(W_t)$   
 $= \sum_{W_t} Pr(W_{t+1} | W_t) Pr(W_t | F_1, \dots, F_t)$

$\propto Pr(W_{t+1} | F_1, \dots, F_{t+1})$   
 $= Pr(W_{t+1} | F_1, \dots, F_t) Pr(F_{t+1} | W_{t+1}, F_1, \dots, F_t)$   
 $= Pr(F_{t+1} | W_{t+1}) B'(W_{t+1})$

$B(W_{t+1}) \propto Pr(F_{t+1} | W_{t+1}) \sum_{W_t} Pr(W_t | W_{t+1}) B(W_t)$   
 Time Elapse:  $B'(W_{t+1})$  from  $BC(W_t)$   
 observation:  $BC(W_{t+1})$  from  $B'(W_{t+1})$   
 - normalize

Particle Filtering  
 - approximate desired distr  
 - store in particles; in one of  $d$  states  
 -  $n \ll d$

Time Elapse  
 - particle in state  $t_i$   
 - sample from  $Pr(T_{t+1} | t_i)$

Observation  
 - assign weight of  $Pr(F_{t+1} | t_i)$  for particle in state  $t_i$  w/ resampling  $F_t$   
 - calculate total weight for each state

- normalize distr, resample

Supervised - input/output  
 - predict outputs  
 Unsupervised - doesn't have corresponding output training data - used to map inputs to outputs  
 Validation data - measure model performance by making predictions on inputs

test set - final exam  
 - adjust hyperparameters

Naive Bayes  
 - classification - group into classes (class)

Features - attributes of data used to learn  
 - Naive: each feature  $F_i$  is CI of all other features given class label  $Y$   
 - table:  $PC(Y) = 2$  entries  
 $PC(F_i | Y) = 4$  entries

total 4 + 2 entries,  $n = \#$  Features  
 prediction  $(F_1, \dots, F_n) = \arg \max_y P(Y | F_1, \dots, F_n)$   
 $= \arg \max_y PC(Y) \prod_{i=1}^n P(F_i | Y)$   
 $= \arg \max_y PC(Y) \prod_{i=1}^n P(F_i | F_1, \dots, F_n)$

observations drawn from distr parameterized by  $\theta$   
 Maximum Likelihood Estimate learns  $\theta$   
 - each  $x_i$  is independent, identically distributed  
 - all values of  $\theta$  are equally likely (uniform prior)

- Likelihood  $L(\theta)$ : fn that represents probability of having drawn sample from distr  
 $L(\theta) = P_\theta(x_1, \dots, x_n) = \prod_{i=1}^n P_\theta(x_i)$  sample

MLE:  $\frac{\partial}{\partial \theta} L(\theta) = 0$   
 fitting: building a model that doesn't generalize well to unseen data

Laplace Smoothing:  
 - strength  $k$  - seen  $k$  extra of each outcome  
 $P_{Lap}(x) = \frac{\text{count}(x) + k}{N + k |X|}$

$|X|$ : number of classes eg words  
 $P_{Lap,k}(x|y) = \frac{\text{count}(x,y) + k}{\text{count}(y) + k |X|}$   
 $P_{Lap,k}(x) = \frac{1}{|X|} \sum_y P_{Lap,k}(x|y)$

$N$ : sample size  
 e.g. words in data

Linear Classifier classify using linear comb. of features (activation)

- takes in data, multiplies each feature of data point by corresponding weight  
 activation  $w(x) = \sum w_i f_i(x)$   
 classify  $(x) = \begin{cases} 1 & \text{if } w(x) > 0 \\ -1 & \text{if } w(x) < 0 \end{cases}$

does feature vector point in same dir ( $< 90^\circ$ ) as predefined weight vector?  
 classify  $(x) = \begin{cases} 1 & \text{if } w(x) > 0 \\ -1 & \text{if } w(x) < 0 \end{cases}$   
 if  $\text{act} = \|w\| \|f(x)\| \cos \theta > 0$ ,  $\cos \theta = 0$ , feature vec  $\perp w$

- line/vector  $\perp w$  separates  $+1$  - boundary  
 "decision boundary"

Binary Perceptron  
 - find decision boundary that separates data  
 - find best possible weights,  $\vec{w}$ , st any training point is correctly classified

Algo:  
 1) initialize all weights to 0.  
 2) For each sample: w/ feature  $F$   
 - classify sample using weight  $y = \text{classify}(x)$   
 - if  $y \neq \text{true}$  (n set  $y^*$ )  
 $w \leftarrow w + y^* F(x)$   
 3) terminate when all samples don't require weight adjustment  
 update = 0  
 1) misclassified + as -  
 $w \leftarrow w + F(x)$   
 2) misclassified - as +  
 $w \leftarrow w - F(x)$

Biases: formen boundary doesn't go thru origin  
 - add feature (except feature vector that is always 1, add extra weight, it might vector  $w^T F(x) + b \geq 0$

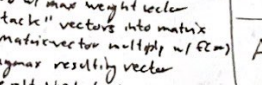
Multiclass  
 - one weight vector for each class  
 classify  $(x) = \max_i w_i^T F(x)$   
 - choose max weight vector  
 - stack "vectors" into matrix  
 - matrix-vector multiply w/  $F(x)$   
 - argmax resulting vector

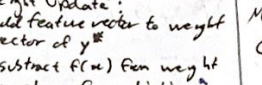
Weight Update:  
 - add feature vector to weight vector of  $y^*$   
 - subtract  $F(x)$  from weight vector of prediction  $\hat{y}$

e.g.  $W_0 = \begin{bmatrix} -2 & 2 & 1 \end{bmatrix}$   
 $W_1 = \begin{bmatrix} 0 & 3 & 4 \end{bmatrix}$   
 $W_2 = \begin{bmatrix} 1 & 4 & -2 \end{bmatrix}$   
 $W = \begin{bmatrix} -2 & 2 & 1 \\ 0 & 3 & 4 \\ 1 & 4 & -2 \end{bmatrix}$

→ reward correct weights, punish misclassifying one

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

  
 ReLU  
 $\ell(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$

  
 Softmax  
 $\sigma(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} = P(\text{class } j | z)$   
 $= P(\text{class } j | F(x))$

Likelihood of weight  $w$  explaining observation

$\mathcal{L}(w) = \prod_{i=1}^m P(y_i | F(x_i); w)$   
 - want weight that maximizes - log-likelihood  
 $\mathcal{L}(w) = \log \mathcal{L}(w)$   
 $= \sum_{i=1}^m \log P(y_i | F(x_i); w)$

$\nabla_w \mathcal{L}(w) = \left[ \frac{\partial}{\partial w_1} \mathcal{L}(w), \dots, \frac{\partial}{\partial w_n} \mathcal{L}(w) \right]$

Gradient Descent  
 - find direction of steepest descent  
 - Initialize weights  $w$   
 for  $i = 0, 1, 2, \dots$   
 $w \leftarrow w - \alpha \nabla_w \mathcal{L}(w)$

Backprop  
 - efficiently calculate gradients  
 For each parameter in NN  
 - value after each node is partial deriv of last node's value w.r.t. to that node's value  
 - gradients of nodes w/ multiple children are sum of gradients of children

$x \rightarrow h \dots f$   
 $\rightarrow i \dots f$

$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial h} \frac{\partial h}{\partial x} + \frac{\partial \mathcal{L}}{\partial i} \frac{\partial i}{\partial x}$

In NN, want to find parameters  $w$  that maximize likelihood of the class probabilities - minimize loss

MLE  
 maximize  $P(\text{data} | \text{evidence})$

$A_{jk} = A^T_{kj}$

Matrix Multiplication!  
 $C_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$

Matrix Vector Multiplication  
 $Ax = \sum_{j=1}^m A_{ij} x_j$

$\left[ \sum_{j=1}^m (A_{ij} + A_{ij}^T) x_j \right]$

$= (A + A^T) x$

Chain Rule  
 $P(A_4 | A_3, A_2, A_1)$   
 $= P(A_4 | A_3, A_2, A_1) \cdot P(A_3 | A_2, A_1)$   
 $\cdot P(A_2 | A_1)$   
 $\cdot P(A_1)$

$\left[ \sum_{j=1}^m (A_{ij} + A_{ij}^T) x_j \right]$

$= (A + A^T) x$

Chain Rule

$P(A_4 | A_3, A_2, A_1)$

$= P(A_4 | A_3, A_2, A_1) \cdot P(A_3 | A_2, A_1)$

$\cdot P(A_2 | A_1)$

$\cdot P(A_1)$



Naïve Bayes - overconfident

→ low entropy posteriors

the more naïve, the more overconfident,  
the lower entropy posterior is

posterior - conditional probability  
after relevant evidence  
is taken into account

$$\sum_B P(B|A) = P(+b|A) + P(-b|A) = 1$$

backward pass step for a node  
depends on node's inputs  
and gradients computed  
downstream of current node.

Viterbi: finding most likely  
path to a node

$$m_t[X_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t}) \\ = P(e_t | x_t) \max_{x_{1:t-1}} P(x_t | x_{t-1}) m_{t-1}[x_{t-1}]$$

Most likely sequence of hidden states  $X_{1:N}^*$

for  $t = 1$  to  $N$ :

for  $x_t \in X(\text{states at } t)$ :

if  $t = 1$ :

$$m_t[X_t] = P(x_t) P(e_1 | x_t)$$

else

$$a_t[X_t] = \arg\max_{x_{t-1}} P(x_t | x_{t-1}) m_{t-1}[x_{t-1}]$$

$$m_t[X_t] = P(e_t | x_t) P(x_t | a_t[X_t]) m_{t-1}[a_t[X_t]]$$

\* this gives path's most likely ending point

for  $t = N$  to  $2$ :

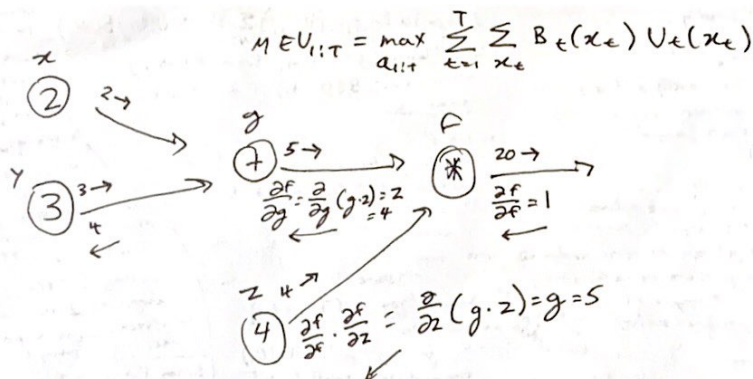
$$X_{t-1}^* = a_t[X_t^*]$$

Variables that are not in union of  
ancestors of query and ancestors of evidence  
can be pruned.

parameters - # of quantities needed to specify  
model ~~size of CPT~~  
(size of CPT)

inference by enumeration: full joint  
computed

output of softmax is probability distribution  
elements between 0 and 1



$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial y} = 1 \cdot \frac{\partial g}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} = 1 \cdot \frac{\partial g}{\partial x} = 1$$

$$g(x) = \frac{1}{1 + \exp(-x)}, \quad \frac{\partial g}{\partial x} = g(x)(1 - g(x))$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} = 1 \cdot g(2)(1 - g(2))$$

calculate joint: if binary.

size of

2 # vars that are not set

↑ #  
or, possible  
vals

Label

Class

$\begin{bmatrix} F_1 \\ \vdots \\ F_n \end{bmatrix}$

Feature  
of sample

$$P(Y | X_1, X_2) = \frac{P(Y, X_1, X_2)}{P(X_1, X_2)}$$

→ Naïve Bayes → Class Allocation

Given many samples, which you know true classifications of,  
how many times does Naïve Bayes accurately classify  
sample?

Particle Filtering /  $P_1: W_1 = 0$   $P_2: W_1 = 1$  *sum weights in state*

1. Compute weight of two particles after evidence  $O_1 = A$

$$w(P_1) = P(O_1 = A | W_1 = 0) = 0.9$$

$$w(P_2) = P(O_1 = A | W_2 = 1) = 0.5$$

2. Using random number, resample  $P_1$  and  $P_2$  based on weights.

Normalize weights...  $P_1: [0, 0.643]$   $P_2: [0.643, 1]$

$P_1 = \text{Sample}(w_{\text{weights}}, 0.22) = 0$

$P_2 = \text{Sample}(w_{\text{weights}}, 0.05) = 0$

3. Elapse Time: Sample  $P_1$  and  $P_2$  from applying time updates

$$P_1 = \text{Sample}(P(W_{t+1} | W_t = 0), 0.33) = 0$$

$$P_2 = \text{Sample}(P(W_{t+1} | W_t = 0), 0.2) = 0$$

4. Observe: Compute the weight of the two particles after evidence  $O_2 = B$

$$w(P_1) = P(O_2 = B | W_1 = 0) = 0.1$$

$$w(P_2) = P(O_2 = B | W_2 = 0) = 0.1$$

5. Resample  $P_1$  and  $P_2$  based on weights

- both  $P_1, P_2 = 0$ , so resampling leaves us w/  $P_1 = 0, P_2 = 0$

6. Estimated Distr For

$$P(W_2 | O_1 = A, O_2 = B):$$

$$P(W_2 = 0 | O_1 = A, O_2 = B) = 2/2$$

$$P(W_2 = 1 | O_1 = A, O_2 = B) = 0/2$$