Probability

marginal: $\sum_a P(A=a, B) = P(B)$

Always formally write out what prob. you're solving for

1. sum $\sum_a P(A=a) = 1$
2. conditional $P(A|C) = \frac{P(A,C)}{P(C)}$
3. independence $P(A,B) = P(A) \cdot P(B)$
4. All of these still apply if all conditioned on D (i.e. sum $\sum_a P(A=a|D) = 1$)

VPI / Decision Networks

observing noisy variables is less valuable

$VPI(x) \triangleq MEU(x) - MEU(\{\})$

$VPI(Y|X) \triangleq MEU(X,Y) - MEU(X)$

$VPI(X,Y) = VPI(X) + VPI(Y|X)$

① If I have data, what will I do

Value iteration ⟷ variable elimination ⇒ computes full thing and then predicts → TOO much time

Product Rule ⇒ doesn't have to break it down all the way i.e. $P(A,B,C) = P(B|AC)P(AC)$

Inference by enumeration

1. select entries consistent w/ evidence
2. sum out H to get joint of query & evidence
3. Normalize

Product Rule

$P(y)P(x|y) = P(x,y)$

Chain Rule

$P(x_1, x_2, x_3, \dots, x_n) = \prod P(x_i | x_1, \dots, x_{i-1})$

(i.e. $P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\dots$)

Sampling

= inference: approx p then show converges in limit (consistent)

Independence

$P(x,y) = P(x)P(y)$

$P(x|y) = P(x)$

Bayes Rule

$P(x,y) = P(x|y)P(y) = P(y|x)P(x)$

Conditional Independence

$X \perp\!\!\!\perp Y | Z$

$P(x,y|z) = P(x|z)P(y|z)$

$P(x|y,z) = P(x|z)$

when combine factors don't include conditionals in fact?

Variable Elimination

* If variable appears in front of conditioning bar → in resulting factor; else behind; can't appear in front for more than one factor

→ join & eliminate till you get query

* if evidence, start w/ factors that select the evidence
* result is selected joint of query & evidence

$P(+r, L) \xrightarrow{\text{normalize}} P(L|+r)$ → by what you want to condition by

* Local CPTs instantiated w/ evidence
* join/elim hidden variables

* polytree always has efficient ordering → cutset to polytree

Prior Sampling

(~ direct evaluation)

- Start from top & move down, sampling at each node; total/normalize

Rejection Sampling

- if query doesn't appear, toss out

Bayes' Net

* Not every BN can represent every joint distribution
* arrows encode conditional independence (NOT necessarily causal structure; could be correlation)

Absolute independence

(x_1) (x_2) (x_3)

CPT count d^2 variables capacity $\times N$

- joint distribution over N variables 2^N
- N-node net $O(N \cdot 2^{k+1})$ save space

Causal

$X \to Y \to Z$

$P(x,y,z) = P(x)P(y|x)P(z|y)$

$X \perp\!\!\!\perp Z | Y$

common cause

$X \leftarrow Y \to Z$

$P(x,y,z) = P(y)P(x|y)P(z|y)$

$X \perp\!\!\!\perp Z | Y$

(w/o Y, observing X might suggest we could observe Z if Y does exist)

common effect

$X \to Z \leftarrow Y$

$X \perp\!\!\!\perp Y$ but NOT when $|Z$ b/c competing explanations

CPT from midterm

$X \to Y \to Z$

Probability tables

$P(X)$, $P(Y|X)$, $P(Z|Y)$ — choose CPTs based on how you marginalize

$P(Y|Z)$ ⇒ X, Y, Z

Probability of full assignment = product of local conditional distributions

Active / Inactive

Active: O→O→O

Inactive: O→●→O

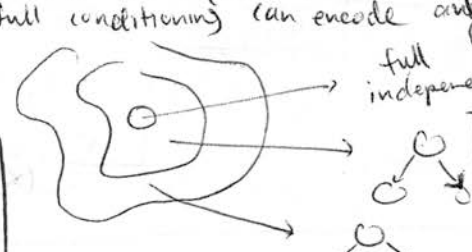
NO active paths b/w x & y (considering all undirected paths) ⇓ independence

gives precise indep info so can build joint dist using this

- Graph structure encodes certain (conditional) independences
- Adding arcs increases set of distributions but has costs (can represent more)

→ full conditioning can encode any distribution

won't know some independences unless inspect specific distribution (not from BN alone)



full independence

For likelihood, estimate probability by counting weights

Factor Zoo

of capitals = dimensionality of table

- Joint: $P(X,Y)$ → sum to 1
- Selected joint: $P(x,Y)$ → fixed x, all y → sums to $P(x)$
- Single conditional: $P(Y|x)$ → fixed x, all y → sums to 1
- Family of conditionals: → $P(X|Y)$ all x, all y → sums to $|Y|$ (i.e. $P(X|y) \to 1$, $|Y|$ times)
- Specified family $P(y|X)$ → fixed y, all X → sums to unknown

w/ factors ⇒ # of entries = 2^{size}

SVM

⇒ max margin

$\min_w \frac{1}{2}\|w\|^2$

$\forall i, y, w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$

Likelihood Weighting

* downside; only reflects in downstream stuff

→ Set evidence & sample rest (weight by probability of evidence)

$S_{WS}(z,e) \cdot w(z,e) = \prod_{i=1}^{l} P(z_i | Parents(z_i)) \prod_{i=1}^{m} P(e_i | Parents(e_i)) = P(z,e)$

Reflex Agent
→ preplanned behavior (acts on it)

DFS
- Stack
- "leftmost" solution
$O(b^m)$ time
fringe = $O(bm)$

More CSPs...?
Local search
→ improve single var until can't be better
(local) → not complete or optimal
(faster more mem efficient) → no change
Hill Climbing
→ move to best neighboring state
→ if none... quit (lol)
Simulated annealing
allow downhill to escape (local) [reuse of time]

BFS
- Queue
- optimal if all costs = 1 or a constant positive weight
- Shallowest level (for solution)
$O(b^s)$ search time
$O(b^s)$ fringe (opens everything)

Iterative Deepening
DFS memory but BFS time
DFS depth 1 2 ... 3
con: computationally redundant & most work done in lower levels anyway

Uniform Search Cost
→ w/ negative const. weight = DFS
- priority queue
- complete if min arc cost > 0 & finite cost
- optimal
- $g(n)$ = backward cost
- explore in every direction
cheapest solution = C^*
arcs @ least ϵ ... effective depth
$C^*/\epsilon \approx m$
time & depth $O(b^{C^*/\epsilon})$

Greedy
- looks only at heuristic (state that looks @ the closest state) $h(n)$ = forward cost
- badly-guided DFS (lol)

A*
(A^* optimal → admissible (tree))
- $f(n) = h(n) + g(n)$
- stop when dequeue goal
- expands toward the goal (not equally in all directions)
A^* optimal ⇓ consistent graph search

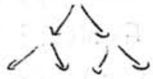
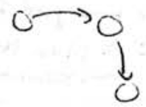
consistent
$A \xrightarrow{1} C$ $h=1$ $h(A) - h(C) \le cost(A \text{ to } C)$
$h=2$ ↓ 3
$h(A) \le cost(A \text{ to } C) + h(C)$
G
* f never decreases along path

Search state
* Based on what info is available
→ details necessary for planning
how many?
example = positions (x, y)
XY etc. (counting)

i.e. if bug locations unknown ⇒ Booleans of where bug might be

Search tree properties
- b is the branching factor
m = max depth
$O(b^m)$ = num nodes in entire tree

Forward checking ONLY b/w assigned & unassigned
Arc consistency (ALL pairs)

Search tree

Search Graph

Search graph = condensed search tree

func Tree-Search (prob, strategy)
init search tree
loop do
if there are no candidates for expansion
↳ failure
choose a leaf node for expansion
if node contains a goal state
return solution
else expand the node & add the resulting nodes to the search tree
end

→ add "visited" closed set for graph search (to avoid duplicate visitations)

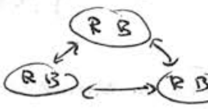
constraint satisfaction problems $O(d^n)$ d = domain
- unary constraint
- explicit: variable ∈ {values}
- binary constraint
- implicit VA ≠ WA
- backtracking (DFS w/ 2 improvements)
① 1 variable @ a time
② check constraints as you go (consider values w/ no conflict)

speed up: ① filtering: - forward checking (cross off values after current assignment, based on constraints) * X lose value, neighbors rechecked
(a.k.a. arc consistency)
* delete from tail
$X \mapsto Y$ consistent iff every x in tail, some y in head
* modify forward checking w/ constraint propogation
queue arcs ⇒ modify then enqueue neighbors
(binary CSP)
$O(n^2d^3) \to O(n^2d^2)$ [can be reduced]

① One solution
② multiple solutions
③ No solutions but doesn't know it yet

CSP ⇒ BREAK CYCLE

② Ordering
Minimum Remaining Values
① choose variable w/ fewest remaining values in domain


convert to tree if possible for analysis

Least constraining value
① One that rules out fewest values in remaining variables

K-consistency K = 2 arc consistency (binary CSP)
Tree structured CSPs $O(nd^2)$ (no loops)
Pick root, order o→o→o→o
① Remove backward
② Assign forward (if no sol, arc c was shown)

Iterative "local"
- no solution = run forever (lol)

Heuristic(s)
Admissible
$0 \le h(n) \le h^*(n)$
"optimistic"
→ relaxed problems
dominance: $h_a(n) \ge h_c(n)$ if true ∀ n
$h(n) = max(h_a(n), h_b(n))$
↳ closer = better directed search
Admissible ≤ = actual optimal cost current node to goal

- for graph search need consistent
consistency ⇒ admissibility

* Tree must be linear

Nearly Tree
⇒ assign all possible values for cutset the tree the rest
* make cutset (very fast for very small)
$O(d^c(n-c)d^2)$ [run-time]
* cutset backtracking ⇒ exponential

Iterative algorithm for CSPs
# live on the edge
while not solved:
randomly select conflicted variable
variable selection = min-conflicts heuristic (violate fewest constraints)

$R = \dfrac{\#\text{ of constraints}}{\#\text{ of values}}$

Performance:

Genetic Algorithms
→ Best N hypotheses (fitness test)
→ pairwise crossover operator

Entropy (Information Encoding)

- code of length $\log(1/p)$
- * more unlikely ⇒ more bits
- * more likely ⇒ less bits

$$E_p\left(\log_2 \frac{1}{p_i}\right) = \sum_{i=1}^{n} -p_i \log p_i$$

- more uniform ⇒ higher entropy
- more values ⇒ higher entropy
- more peaked ⇒ lower entropy
- rare values almost "don't count"

Information Gain

Gain = entropy before split − entropy after split

use * expected entropy

Pruning

- Build tree
- Start from bottom & prune according to $P_{chance} > \text{Max } P_{chance}$ ↳ delete this one

* some chance nodes not pruned if redeemed later

* pruning intended to improve held out accuracy

$\max P_{chance}$ = regularization param (held-out data)

Prevent overfitting

1. reduce hypothesis space
2. regularization

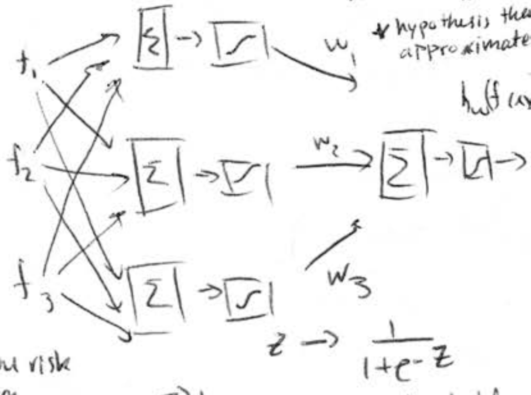
$$\min_{w} \sum_{i=1}^{m} (y^{(i)} - h_w(f(x^{(i)}$$

* hypothesis that approximate

$h_w(f(x$

Neural Networks

- * universal approximators
- * learning features



$z \to \frac{1}{1+e^{-z}}$

→ by not having threshold be strict edge then won't get stuck when hill climbing

* too many neurons & you risk overfitting

* each weight is a learned parameter

continuous ↦ discrete variables (Decision T)

- pick a point a to split on to give highest information gain (effective "Discrete")
- * if get to data missing in training (for particular attribute), then pick "most likely" ⇒ branch w/ node w/ greatest # of items

Classifiers + Kernels

PNo Bias (origin) — PBias (shifted hyperplane)

* activity on one side of decision tree shouldn't reflect on other side b/c they already diverged

Quadratic

Kernel maps to $[1, x_1, x_2, x_1x_2, x_1^2, x_2^2]$ & then projects back (linear in quadratic space)

- circles, ellipsoids & parabolas

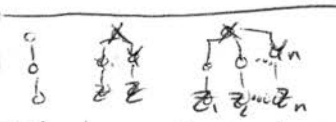
* just b/c the features exist in weights, doesn't mean they need a value

⇒ more training data always good

⇒ remove edges on Bayes means reduce hypothesis space

* growth is often roughly quadratic in # of features

* dual perception works best if more dimensionality of feature space than training data/points



Query: $P(Y_n \mid z_1, \ldots, z_n)$

Best ordering: $Y_1, \ldots Y_{n-1}, \cancel{X}$ ↳ constant # of entries

Worst Ordering: $X, Y_1, \ldots, Y_{n-1}$ ⇒ exponential growth

* admissible just needs to be less than true cost; underestimate

Bag of Words Model

- feature for each word in the document; value is the word @ that position in document (library **V** is ALL words)
- CPTs same for each feature; where word appears is irrelevant

* CSPs (w.s.Q.) to determine # of iterations determine worst case scenario? tree ordering? how many variables need to be reassigned?

Game trees ⇒ β resets itself for the new branch & only looks @ α & β relevant to itself

RL learns about agent's own optimal behavior

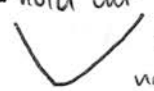
- if want to learn more about an adversary's behavior (which you know follows a certain rule) then state space search

EMV = expected monetary value (expectation)

equivalent monetary value "how much $ do I need to get offered whether lottery & $ are equivalent" ← need utility func

* If there is uncertainty in Pacman's position (non-deterministic behavior) ⇓ model w/ MDP

* hold out



is the norm

→ as training error improves so does hold out, but eventually error flattens, then classifier probably overfits

Ⓐ-Ⓑ-Ⓒ Ⓓ Ⓔ

This is not a tree can't run arc consistency, but if cutset & assign a, then can apply arc consistency

* If assign in linear order where node assigned before all children, then one arc consistency will prevent backtracking

how much would I pay questions

insurance − equivalent $ value

} b/c paying more means lottery is not worth it

How to determine CPTs

1. Independence
2. what do I need to marginalize

Handout Question w/ MRV & LCV:

1. LCV doesn't affect nodes expand b/c it just orders values (& we got to go through all)
2. MRV does affect b/c increase computation b/c of incompatible values remaining (used) w/ restricted variables

* If assign cutset ⇒ so it's not cycle anymore first then repeatedly apply arc consistency won't need to backtrack

* Naive Bayes is overconfident ⇒ lower entropy b/c variation in prob decreases

* Tabular Q-learning & full-depth minimax both compute exact value of all states

* Feature based Q learning is approximation of the states (won't match minimax)

Non linearly separable data, (1) Case-based reasoning

* Classification (labeled data)

Parametric models
1. Fixed set of parameters
2. more data => better settings (tune the parameters) (i.e. perceptron

Nearest neighbor (sim. function)

1-NN => most similar data pt

K-NN => pick K nearest neighbors & vote (need weighting scheme)

Dual Perceptron

$w_y = 0 + f(x_i) - f(x_j) + \dots$

perceptron is explicit learning b/c train parameters

$$w_y = \sum_i \alpha_{i,y} f(x_i)$$

$\alpha = \langle \alpha_{1,y}, \alpha_{2,y}, \dots \rangle$ dual representation

Score: $w_y \cdot f(x)$

$$= \left(\sum_i a_{i,y} f(x_i)\right) \cdot f(x)$$

$$= \sum_i a_{i,y} \left(f(x_i) \cdot f(x)\right)$$

$$= \boxed{\sum_i a_{i,y} K(x_i, X)}$$

trying to classify x

Kernel = similarity

* diff types of Kernel functions

* Kernel could be black box; don't even need to know weight vectors or feature vectors

Non-parametric
- complexity of classifier increases w/ data
- better in limit. worse in non-limit
  → when similarity is needed

Sim function: (1) dot product (2) invariant matrices (i.e. rotation) (3) data augmentation

no explicit learning

→ what you use to determine nearest neighbor

Infinite Dimension

$K(x, x')$
$= \exp(-\|x-$
$e^{-\|x-x'\|}$

Dual Perceptron } how to do similarity function perceptron
- one alpha vector per label; one alpha entry for each training data
* can learn to ignore noisy data
* allows to do nearest neighbor w/ perceptron ∴ allowing learning to happen (tune α's => parameters)

Kernels → non-linear separators

e.g. quadratic kernel

$$K(x, x') = (x \cdot x' + 1)^2$$

$$= \sum_{i,j} x_i x_j x'_i x'_j + 2\sum_i x_i x'_i + 1$$

→ so as not to represent according to vector full size of features

Kernel Classification:
1. start w/ zero
2. $y = \arg\max_y \sum_i \alpha_{i,y} K(x_i, X_n)$

$\alpha_{y,n} = \alpha_{y,n} - 1$

$\alpha_{y^*,n} = \alpha_{y^*,n} + 1$

Perceptron: use held out error to determine when to stop → prevent overfit

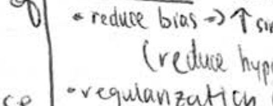
* It's ok to think of same looking points as different as long as classify

Agglomerative Clustering produce dendrogram

1. initially, each instance its own cluster

Repeat:
2. two closest clusters
3. Merge into new cluster
4. stop when one cluster left

* can pick which level of dendrogram to look @



"Closest": (1) closest pair (2) farthest pair (3) average of all pairs (4) Ward's method (min variance, like k means)

Clustering
- no labels yet
- unsupervised learning

K-Means (* class used euclidean dist)
1. pick K arbitrary centers
2. assign data to nearest center
3. average & reassign center → the assigned points
4. repeat till assignments don't change

(3) minimizes dist b/w diff pts by reassignment w/ Manhat dist => median

$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum \text{dist}(x_i, c_{a_i})$$

points, assignments, means

* this is a minimization problem so it will converge

- reassignment minimize according to d metric used

* initialization makes a difference (variance-based, split/merge, etc.)

Decision Trees
- compact representation of probability table
* automatically reduces size of hypothesis space
* allows to be more expressive than typical perceptron => conjunction features

Size of CPT

Binary classification w/ 3 variables (binary)

2^3 => variable combos. each of these 8 have 2 labels $(2^{2^3} = 2^8)$

- bias => keep getting same consistent error in estimation (too "biased")
- variance → overfitting → too much Δ b/w hypotheses
- reduce bias → ↑ simplicity (reduce hypothesis space)
- regularization (smoothing) can keep hypothesis space large but can't get to extremes
- variance ⟺ consistency

consistency & simplicity => no overfitting

have a g we need to approx from (x, g(x))
- hypothesis space, H (defined by training data)

→ classification } uses
→ regression } hypothesis space

Decision Tree Size

4^n: n = # of attributes

i.e. w/ 1 attribute

Y_0, Y_1, Y_0, Y_1
Y_0, Y_1, Y_1, Y_0 } 4 diff possibilities (binary decision)

Search Problem

start state → goal state

solution = a plan that does this transformation

Game algorithms

strategy (policy) for move to move

terminal state: value known

Value: best utility from that state

* deeper eval func → weighted linear combo features; quality matters less

minimax

→ minimax value: best achievable utility against a rational (optimal) adversary

like DFS: Time $O(b^m)$, Space $O(bm)$ } iterative deepening else too much

* eval func

```
def value (state):
    if terminal, return utility
    if next agent max: return max_val
    if next agent min: return min_val
```

α = MAX's best option on path to root

β = MIN's best option on path to root

```
def max-value (state, α, β):
    init v = -∞
    each successor
        v = max(v, val(succ., α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
min-value:
    init v = +∞
    each successor:
        v = min
        if v ≤ α return v
        β = min(β, v)
    return v
```

optimal max → highest val possible

optimal min → lowest val possible

alpha-Beta: no effect on minimax value for root

* alpha-beta
→ remember what each references (max → β, min → α)

expectimax (average case) ← replace min_value

* be careful w/ pruning for expectimax

* depending on adversary, expectimax can be optimistic

* minimax = pessimistic

* minimax, terminal function scale doesn't matter (insensitivity to monotonic transformations)

* expectimax ⇒ magnitudes must be meaningful

Preference $A \succ B$

Indifference $A \sim B$

$U(A) \geq U(B) \Leftrightarrow A \succeq B$

$U(L) = \sum_{x_i \in L} P(x_i) U(x_i)$

Axioms of Rationality

① orderability

$(A \succ B) \vee (B \succ A) \vee (A \sim B)$

② Transitivity

$(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

③ continuity

$A \succ B \succ C \Rightarrow \exists_p [p, A; 1-p, C] \sim B$

④ substitutability

$A \sim B \Rightarrow [p, A; 1-p, C] \sim [p, B; 1-p, C]$

⑤ Monotonicity

$A \succ B \Rightarrow$

$(p \geq q \Leftrightarrow [p, A; 1-p, B] \succeq [q, A; 1-q, B])$

Money

* pay in exchange for L

$EMV(L) = p \cdot X + (1-p) \cdot Y$

$L = [p, \$X; (1-p), \$Y]$

$U(L) = p * U(\$X) + (1-p) U(\$Y)$

$U(L) < U(EMV(L))$

MDP

* one way to solve is expectimax

* depends only on current state

* like search

* more, sooner

Policy = action for each state

Utility = sum of (discounted) rewards

* γ helps algorithm converge

Bellman equations (expected utility when act optimally)

$V^*(s) = \max_a Q^*(s,a)$

$Q^*(s,a) = \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$ (probabilities included)

$V^*(s) = \max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$

Time-limited

$V_k(s)$ = optimal value k more time steps

Policy Evaluation

all updates based on full iter

$V_0^\pi(s) = 0$

* Efficiency $O(s^2)$ per iter

$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_k^\pi(s')]$

Value Iteration:

* Repeat till converge

* complexity of iteration: $O(S^2A)$

$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V_k(s')]$

consider what reward is motivating

* policy may converge before optimal values

Policy Extraction

$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V^*(s')]$ (effectively max Q)

Policy Iteration

① Policy evaluation

② Policy improvement ⇒ use one-step look-ahead w/ resulting utilities

one more iteration, max (Q)

* converge much faster under some conditions

① is faster than value iteration

② is same pace as value iteration

learning → model-based
- look @ episodes
- norm to get probability
- discover rewards

⇒ Solve MDP

→ w/o model
① Passive
② Active

→ decreasing learning rate can give converging values

Episode = start to finish

Iter = each step in episode

① Passive (observe)

→ no choice in policy (given) ⇒ effectively evaluation ⇒ find value

Ⓐ Direct Evaluation

① Act according to π

② every time visit a state, write what discounted rewards turned out to be

③ average (equal weight) $\frac{1}{N} \sum a_i$

Ⓑ Temporal Difference Learning

$sample = R(s, \pi(s), s') + \gamma V^\pi(s')$

$V^\pi(s) \leftarrow (1-\alpha) V^\pi(s) + \alpha \cdot sample$

OR

$V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

② Active: get to choose actions; find policy [learn policies that maximize rewards, not values that predict them]

Ⓐ Q learning ⇒ will converge; off-policy learning

$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$

$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$

$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha(sample)$

* explore enough

* eventually make α small enough; not too quickly else conv. to sa

Exploration vs. Exploitation

ε-greedy

$P = \varepsilon \Rightarrow$ random

$1-\varepsilon \Rightarrow$ current policy

* to prevent thrashing over time lower ε over time

or exploration func

$f(u,n) = u + k/n$ → eventually stop exploring states; n = # of times visited → bonus to unknown

[$\gamma f(Q(s',a'), N(s',a'))$

$diff = [r + \gamma \max_{a'} Q(s',a')] - Q(s,a)$

$w_i \leftarrow w_i + \alpha [diff] f_i(s,a)$

weighted combo of feat = value

Approximate Q
- generalize states
- vector of features
- con: share feat but diff on value

Naive Bayes Nets

→ features are conditionally independent given label

→ info about f_1 doesn't give info about f_2 [i.e. aligned along rectangular axis]

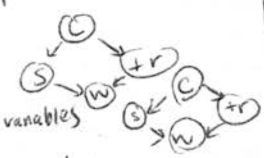
Training → small data (training) → less error
(reverse for test) ↳ also more overfit

Gibbs Sampling

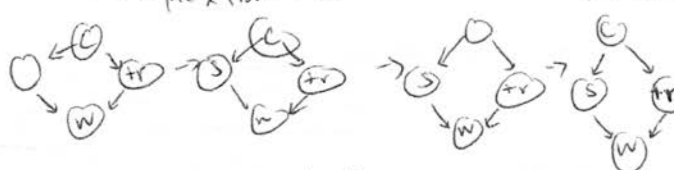
* sum of weights = how "effective" samples were obtained (want high)

* start w/ arbitrary instantiation consistent w/ evidence; sample one variable @ a time, conditioned on all the rest, but keep evidence fixed. Repeat

1. Fix evidence
2. initialize other variables
3. Repeat { choose nonevidence variable X; resample X from P(X | all other variable) }



i.e.



$P(S \mid +c, +r, -w) = \dfrac{P(s \mid +c)\, P(-w \mid s, +r)}{\sum_s P(s \mid +c)\, P(-w \mid s, +r)}$

⇒ only CPTs that have resampled variable need to be considered, & joined together

weak model underfits on large data

Model structure (& Laplace Smoothing)

→ smoothing decreases maximum likelihood expectation

→ if model represents a larger family of models then the MLE for that model IS greater

Particle Filtering (likelihood sampling)

→ calculating probabilities = normalizing the weights (counting weighted samples)

→ only count samples w/ the correct conditioning matching the query

Forward Algorithm = sum
Viterbi = max

1. Elapse time
$x' = \text{sample}(P(X' \mid x))$

2. Observe
$w(x) = P(e \mid x)$

3. Resample

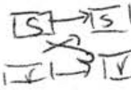
Dynamic Bayes Net

* create super nodes
* "unroll" for T time steps, eliminate variables, then eliminate till $P(X_T \mid e_{1:t})$

State Trellis

$\arg\max_{x_{1:t}} P(x_{1:t} \mid e_{1:t})$

- graph of states & transitions



- arc weight $= P(x_t \mid x_{t-1})\, P(e_t \mid x_t)$
- each transition $x_{t-1} \to x_t$ = arc
- product of weights = sequence's probability

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

$= P(e_t \mid x_t) \max_{x_{t-1}} P(x_t \mid x_{t-1})\, m_{t-1}[x_{t-1}]$

Laplace

$\dfrac{c(x,y)+k}{c(y)+k|X|}$

Naive Bayes

M1 parameters

$P(Y, F_1, \ldots, F_n) = P(Y) \prod_i P(F_i \mid Y)$ Y = label/parameters, F = evidence/parameter

$|Y| \times |F|^n$ values

$n \times |F| \times |Y|$ parameters (for naive)

$P(Y)$ = prior over labels

maximize likelihood of data = $L(x, \theta) = \prod_i P_\theta(x_i)$

↳ i.e. $\arg\max_\theta P(X \mid \theta)$

or, most likely parameter given data

$\arg\max P(\theta \mid X) = \theta_{MAP} \propto \arg\max_\theta P(X \mid \theta) P(\theta)$

Viterbi Algorithm

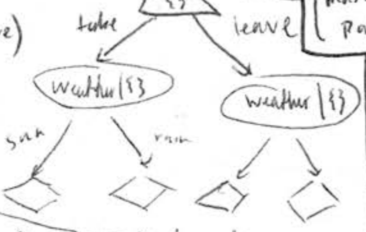
$P(X_{1:T} \mid e_{1:T}) \propto P(X_{1:T}, e_{1:T})$

maximize path

Inference

1. get joint probabilities of label & evidence for each label
2. sum to get probability of evidence
3. normalize

Decision networks



$\sum_w P(w)\, U(\text{leave}, w) = EU(\text{leave})$

$MEU = \max_a EU(a)$

$EU(\text{leave} \mid \text{bad}) = \sum_w P(w \mid \text{bad})\, U(\text{leave}, w)$ → w/ evidence

$MEU(F = \text{bad}) = \max_a EU(a \mid \text{bad})$

$VPI(E' \mid e) = \left(\sum_{e'} P(e' \mid e)\, MEU(e, e')\right) - MEU(e)$

↓ for every evidence, insert a step like this

VPI properties

If Parents(U) ⫫ Z | CurrentEvidence
Then VPI(Z | CurrentEvidence) = 0

1. Nonnegative
2. Nonadditive
$VPI(E_j, E_k \mid e) \neq VPI(E_j \mid e) + VPI(E_k \mid e)$
3. Order-Independent
$VPI(E_j, E_k \mid e) = VPI(E_j \mid e) + VPI(E_k \mid e, E_j)$
$= VPI(E_k \mid e) + VPI(E_j \mid e, E_k)$

Model based
label & features = RV
query for dist of label cond. on features

→ how much you pay for info = prob of not ideal · payoff for continuing knowing not ideal

weight vector = hyperplane

Binary Perceptron

$w = w + y^* \cdot f$

Multiclass

$w_y = w_y - f(x)$

$w_{y^*} = w_{y^*} + f(x)$

MIRA

$\min_w \frac{1}{2} \sum_y \|w_y - w'_y\|^2$

$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$

always changes for @ least 1 improvement

$w_y = w'_y - \tau f(x)$

$w_{y^*} = w'_{y^*} + \tau f(x)$

max step size

$\tau^* = \min\left(\dfrac{(w'_y - w'_{y^*}) f + 1}{2 f \cdot f}, C\right)$

Time elapse

$P(X_{t+1} \mid e_{1:t}) = \sum_{x_t} P(X_{t+1} \mid x_t)\, P(x_t \mid e_{1:t})$

Observation

$P(e_{t+1} \mid X_{t+1})\, P(X_{t+1} \mid e_{1:t}) = P(X_{t+1} \mid e_{1:t+1})$

Forward Algorithm

$P(x_t) = \sum_{x_{t-1}} P(x_t \mid x_{t-1})\, P(x_{t-1})$

Inferences

$P(X_1 \mid e_1) \propto P(X_1) P(e_1 \mid X_1)$

$P(X_2) = \sum_{x_1} P(x_1) P(X_2 \mid x_1)$

(base cases)

Stationary Dist

$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X \mid x)\, P_\infty(x)$

when learning select distribution that maximizes likelihood