Part 1: Search

relex agent: based on current State; planning agent: consequence of action, choose best (by simulating the world!)

A search Problem is consited of ① A state space, ② A set of actions

③ a transition model ④ An action cost ⑤ Start state ⑥ Goal state?

	property	Optimal: Tree	Optimal: Graph	Complete: Tree	Complete: Graph	Time	space
BFS	FIFO queue	✓ cost=1	✓	✓	✓	b^s	b^s
DFS	LIFO stack	X	X	X	✓	b^m	bm
Interative Deepening	each time search depth i = 1,2 ----	✓ cost=1		X		b^s	bm
uniform cost search	f(n)=g(n)	✓	✓	✓	✓	$b^{C^*/\varepsilon}$	$b^{C^*/\varepsilon}$
A^* Search	f(n) = g(n)+h(n)	✓		X			

① admissibility: $\forall n\ 0 \le h(n) \le h^*(n)$

② consistency: $\forall A, C,\ h(A) - h(C) \le \text{cost}(A, C)$.

③ dominance: $\forall n\ h_a(n) \ge h_b(n)$

Theorem 1: if h is admissible, then A^* yields an optimal solution in tree search.

Theorem 2: if h is consistent, then A^* yields an optimal solution in Graph search.

greedy search: select the one with lowest heurist value to expand.

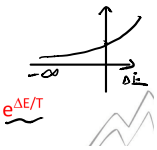
not complete & optimal.

Local search:

① Hill climbing search: best neighbor → not C & O

② Simulated Annealing Search:

```
function SIMULATED-ANNEALING(problem,schedule) returns a state
current ← problem.initial-state
for t = 1 to ∞ do
    T ←schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.value – current.value
    if ΔE > 0 then current ← next
        ΔE < 0 else current ← next only with probability e^{ΔE/T}
```



③ Local Beam search. best K result each time

④ Genetic Algorithm:

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

2. Game → zero-sum game. adversarial search problems

① Min-Max Tree

② Alpha-Beta Pruning: $O(b^{\frac{m}{2}})$ (all adversial game can use) Sometimes has upper bound Expected Max

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

← backward →

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

*Max pruning leaf = $(b-1)(m-1)$; remain the left branch of each branch from root

reorder leaves. # possible value = # leaf node - max pruning node.

③ ExpectiMax ; cut from ∇, cut K largest ; cut from Δ: cut K smallest

④ Monte Carlo Tree Search.

while

S1 Tree traversal: $UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{PARENT}(n))}{N(n)}}$ (U(n): # win of parent; N(n): # total rollout)

S2 Node Expansion.

S3 simulation (Rollout)

S4 Backpropagation

3 Logic

valid: true for all models

satisfiable ∃ model. true

unsatisfiable: not true for all models

conjunctive normal form: $[P_1 \vee \cdots \vee P_i) \wedge \cdots \wedge (P_j \vee \cdots P_n]$

entail: $A \models B$.

① $A \models B$ iff $A \Rightarrow B$ is valid ② $A \models B$ iff $A \wedge \neg B$ is unsatisifiable.

Model checking

DPLL → is a variant of DFS

```
function DPLL-SATISFIABLE?(s) returns true or false   sound (correct) & complete.
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(clauses, symbols, { })

function DPLL(clauses, symbols, model) returns true or false

  if every clause in clauses is true in model then return true
  if some clause in clauses is false in model then return false   Early Termination. (one symbol T)
  P, value ← FIND-PURE-SYMBOL(symbols, clauses, model)   Pure Symbol Heuristic (all A or ¬A and assign as T)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})
  P, value ← FIND-UNIT-CLAUSE(clauses, model)   Unit clause Heuristic Eg (B ∨ F ∨ F ∨ F)
  if P is non-null then return DPLL(clauses, symbols – P, model ∪ {P=value})   only valid in clause.
  P ← FIRST(symbols); rest ← REST(symbols)
  return DPLL(clauses, rest, model ∪ {P=true}) or
         DPLL(clauses, rest, model ∪ {P=false}))
```

three rules of inference

① Modus Ponens: if knowledge base contain $A \& A \Rightarrow B$, we can infer B

② And-Elimination: if knowledge base contain $A \wedge B$. we can infer A and B.

③ Resolution: if knowledge base contain A and $\neg B$. we can infer $A \wedge B$

Forward chaining KB?

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
    p ← POP(agenda)
    if p = q then return true
    if inferred[p] = false then
      inferred[p] ← true
      for each clause c in KB where p is in c.PREMISE do
        decrement count[c]
        if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

4 Bayes Net (tree-like net usually not holds ???)

mutually independent $(A \perp\!\!\!\perp B)$: $P(A, B) = P(A) \cdot P(B)$

Conditional independent $B \perp\!\!\!\perp A \mid C$. $P(A, B \mid C) = P(A \mid C)\, P(B \mid C)$. ; $P(A \mid B, C) = P(A \mid C)$; $P(B \mid A, C) = P(B \mid C)$

$P(Q_1 \cdots Q_K \mid e_1 \cdots e_k)$: Q_i: Query variable e_i: Evidence variable. Hidden variable...

① Inforce by Enumeration

1. Collect all the rows consistent with the observed evidence variables.
2. Sum out (marginalize) all the hidden variables.
3. Normalize the table so that it is a probability distribution (i.e. values sum to 1)

zero-sum game.

if one takes suboptimal other than opt. the oppent's utility ↑

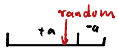
Bayes Network Representation. (DAG)

② Variable Elimination. $O(c^n)$

function ELIMINATION-ASK(X, **e**, bn) **returns** a distribution over X
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
  factors ← [] (unnormalized P)
  for each var **in** ORDER(bn.VARS) **do**
    factors ← [MAKE-FACTOR(var, **e**)|*factors*]
    if var is a hidden variable **then** *factors* ← SUM-OUT(var, *factors*)
  return NORMALIZE(POINTWISE-PRODUCT(*factors*))

order of elimination: Best → NP-hard problem:
Maybe good? ① least out-neighbor ② min-weight ③ min-fill: min size of factor
Approximate.

③ Prior Sampling



$P(\text{sample}) = \prod_{i}^{n} P(x_i \mid \text{Parent}(x_i))$

④ Rejection Sampling: consistent with given info? reject it!

⑤ Likelihood weight (evidence → P, Query → Sampling) | consistent evidence

function LIKELIHOOD-WEIGHTING(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  inputs: X, the query variable
    e, observed values for variables **E**
    bn, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$
    N, the total number of samples to be generated
  local variables: **W**, a vector of weighted counts for each value of X, initially zero

  for j = 1 to N **do**
    x, w ← WEIGHTED-SAMPLE(bn, **e**)
    W[x] ← **W**[x] + w where x is the value of X in **x**
  return NORMALIZE(**W**)

---

function WEIGHTED-SAMPLE(bn, **e**) **returns** an event and a weight
  w ← 1; **x** ← an event with n elements initialized from **e**
  foreach variable X_i **in** $X_1, \ldots, X_n$ **do**
    if X_i is an evidence variable with value x_i in **e**
      then $w \leftarrow w \times P(X_i = x_i \mid parents(X_i))$
      else **x**[i] ← a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return **x**, w

Gibbs Sampling: (consistent) ~> evidence

function GIBBS-ASK(X, **e**, bn, N) **returns** an estimate of $\mathbf{P}(X|\mathbf{e})$
  local variables: **N**, a vector of counts for each value of X, initially zero
    Z, the nonevidence variables in bn
    x, the current state of the network, initially copied from **e**

  initialize **x** with random values for the variables in **Z**
  for j = 1 to N **do**
    for each Z_i in **Z** **do**
      set the value of Z_i in **x** by sampling from $\mathbf{P}(Z_i|mb(Z_i))$ (markov_blanket: child, parents, child's parent)
      N[x] ← **N**[x] + 1 where x is the value of X in **x**
  return NORMALIZE(**N**)

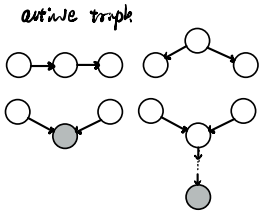
Type of Environment in AI

① Fully observable / Partially Observable: sense the complete state at each point time
② Static / dynamic: the environment keep change?
③ stochastic / deterministic: a uniqueness in the agent's current state determine next state
④ know physics: agent know the transition model:

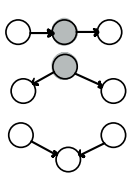
Conditional independence.

1. Shade all observed nodes $\{Z_1, \ldots, Z_k\}$ in the graph.
2. Enumerate all undirected paths from X to Y.
3. For each path:
   (a) Decompose the path into triples (segments of 3 nodes).
   (b) If all triples are active, this path is active and *d-connects* X to Y.
4. If no path d-connects X and Y, then X and Y are d-separated, so they are conditionally independent given $\{Z_1, \ldots, Z_k\}$

active graph



inactive triples



Markov Models

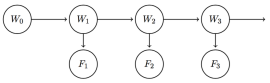
Memoryless / Markov Property: $W_{i+1} \perp \{W_0, \ldots, W_{i-1}\} \mid W_i$

Stationary Assumption: for all values of i: $P(W_{i+1} \mid W_i)$ is identical.

the Mini-Forwarding Algorithm: $P(W_{i+1}) = \sum_{w_i} P(W_{i+1} \mid w_i) \cdot P(w_i)$

we must compute the Stationary distribution: $P(W_{t+1}) = P(W_t)$

Hidden Markov Models



Markov Property: ① $F_1 \perp W_0 \mid W_1$: F_1 only depends on W_1
② $\forall i = 2, \ldots, n$, $W_i \perp \{W_0, \ldots, W_{i-2}, F_1, \ldots, F_{i-1}\} \mid W_{i-1}$
③ $\forall i = 2, \ldots, n$, $F_i \perp \{W_0, \ldots, W_{i-1}, F_1, \ldots, F_{i-1}\} \mid W_i$

Assumptions: ① $P(W_{i+1} \mid W_i)$ is stationary ② sensor model $P(F_i \mid W_i)$ is stationary.

Forwarding Algorithm: $B'(W_{i+1}) = P(W_{i+1}|f_1, \ldots, f_i) = \sum_{w_i} P(W_{i+1}, w_i|f_1, \ldots, f_i)$ $\quad B(W_i) = P(W_i|f_1, \ldots, f_i)$

① time elapse update: $B'(W_{i+1}) = \sum_{w_i} P(W_{i+1}|w_i)B(w_i)$

② observation update: $B(W_{i+1}) \propto P(f_{i+1}|W_{i+1})B'(W_{i+1}) = P(f_{i+1}|W_{i+1}) \sum_{w_i} P(W_{i+1}|w_i)B(w_i)$

Viterbi Algorithm: $m_t(x_t) = \max_{x_{1:t}} P(x_{1:t}, e_{1:t})$

$= \max_{x_{1:t-1}} P(e_t|x_t) P(x_t|x_{t-1}) P(x_{1:t-1}, e_{1:t-1})$

$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) \max_{x_{1:t-2}} P(x_{1:t-1}, e_{1:t-1})$

$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) \cdot m_{t-1}(x_{t-1})$

Result: Most likely sequence of hidden states $x^*_{1:N}$

```
/* Forward pass                                                    */
for t = 1 to N do
    for x_t ∈ 𝒳 do
        if t = 1 then
            m_t[x_t] = P(x_t)P(e_0|x_t)
        else
            a_t[x_t] = arg max_{x_{t-1}} P(x_t|x_{t-1})m_{t-1}[x_{t-1}]; -> best 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]];
        end
    end
end
/* Find the most likely path's ending point                        */
x*_N = arg max_{x_N} m_N[x_N];
/* Work backwards through our most likely path and find the hidden
   states                                                          */
for t = N to 2 do
    x*_{t-1} = a_t[x*_t];
end
```

Particle Filtering: ?

- *Time Elapse Update* - Update the value of each particle according to the transition model. For a particle in state t_i, sample the updated value from the probability distribution given by $P(T_{i+1}|t_i)$. (Sample all ⟼ t_i)
- *Observation Update* - During the observation update for particle filtering, we use the sensor model $P(F_i|T_i)$ to weight each particle according to the probability dictated by the observed evidence and the particle's state. Specifically, for a particle in state t_i with sensor reading f_i, assign a weight of $P(f_i|t_i)$. The algorithm for the observation update is as follows:

  1. Calculate the weights of all particles as described above.
  2. Calculate the total weight for each state.
  3. If the sum of all weights across all states is 0, reinitialize all particles.
  4. Else, normalize the distribution of total weights over states and resample your list of particles from this distribution.

Utility & Rationality, decision.

Axioms of Rationality:

- *Orderability*: $(A \succ B) \vee (B \succ A) \vee (A \sim B)$
  A rational agent must either prefer one of A or B, or be indifferent between the two.
- *Transitivity*: $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
  If a rational agent prefers A to B and B to C, then it prefers A to C.
- *Continuity*: $A \succ B \succ C \Rightarrow \exists p\, [p, A;\ (1-p), C] \sim B$
  If a rational agent prefers A to B but B to C, then it's possible to construct a lottery L between A and C such that the agent is indifferent between L and B with appropriate selection of p.
- *Substitutability*: $A \sim B \Rightarrow [p, A;\ (1-p), C] \sim [p, B;\ (1-p), C]$
  A rational agent indifferent between two prizes A and B is also indifferent between any two lotteries which only differ in substitutions of A for B or B for A.
- *Monotonicity*: $A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A;\ (1-p), B] \succeq [q, A;\ (1-q), B]$
  If a rational agent prefers A over B, then given a choice between lotteries involving only A and B the agent prefers the lottery assigning the highest probability to A.

Value of perfect information:

$$VPI(E'|e) = MEU(e, E') - MEU(e)$$

$$MEU(e) = \max_a \sum_s P(s|e)U(s, a)$$

$$MEU(e, E') = \sum_{e'} P(e'|e)MEU(e, e')$$

$$MEU(e, e') = \max_a \sum_s P(s|e, e')U(s, a)$$

Property:

① **Nonnegativity.** $\forall E', e\ VPI(E'|e) \geq 0$

② **Nonadditivity.** $VPI(E_j, E_k|e) \neq VPI(E_j|e) + VPI(E_k|e)$ in general.

③ **Order-independence.** $VPI(E_j, E_k|e) = VPI(E_j|e) + VPI(E_k|e, E_j) = VPI(E_k|e) + VPI(E_j|e, E_k)$

Markov Decision Processes

Bellman equation: rT. -> focus on long term.

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

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

① Value Iteration

1. $\forall s \in S$, initialize $U_0(s) = 0$. This should be intuitive, since setting a time limit of 0 timesteps means no actions can be taken before termination, and so no rewards can be acquired.
2. Repeat the following update rule until convergence: -> $U_{k+1}(s) = U_k(s)$

$$\forall s \in S,\ U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U_k(s')]$$

② Policy Extraction

$$\forall s \in S,\ \pi^*(s) = \operatorname*{argmax}_a Q^*(s,a) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^*(s')]$$

③ Q-Value Iteration:

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

④ Policy Iteration:

1. Define an *initial policy*. This can be arbitrary, but policy iteration will converge faster the closer the initial policy is to the eventual optimal policy.
2. Repeat the following until convergence:
   - Evaluate the current policy with **policy evaluation**. For a policy π, policy evaluation means computing $U^\pi(s)$ for all states s, where $U^\pi(s)$ is expected utility of starting in state s when following π:
   $$U^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma U^\pi(s')]$$
   Define the policy at iteration i of policy iteration as π_i. Since we are fixing a single action for each state, we no longer need the max operator which effectively leaves us with a system of $|S|$ equations generated by the above rule. Each $U^{\pi_i}(s)$ can then be computed by simply solving this system. Alternatively, we can also compute $U^{\pi_i}(s)$ by using the following update rule until convergence, just like in value iteration:
   $$U^{\pi_i}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s')[R(s,\pi_i(s),s') + \gamma U^{\pi_i}_k(s')]$$
   However, this second method is typically slower in practice.
   - Once we've evaluated the current policy, use **policy improvement** to generate a better policy. Policy improvement uses policy extraction on the values of states generated by policy evaluation to generate this new and improved policy:
   $$\pi_{i+1}(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma U^{\pi_i}(s')]$$
   If $\pi_{i+1} = \pi_i$, the algorithm has converged, and we can conclude that $\pi_{i+1} = \pi_i = \pi^*$.

Reinforcement Learning:

- Passive RL -> policy evaluation
  - Direct Evaluation
  - TD - Learning
- Active RL -> learn policy
  - model - based. RL
  - model - free RL -> Q - Learning

① Direct Learning

Reward(A) = $\frac{\sum R(A, a, s')}{\#\text{ time A appears}}$

② Temporal Difference Learning

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

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

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

③ Q-Learning:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot \text{sample}$$

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

Approximating Q-Learning -> feature-based representation.

$$V(s) = w_1 \cdot f_1(s) + w_2 \cdot f_2(s) + \ldots + w_n \cdot f_n(s) = \vec{w} \cdot \vec{f}(s)$$

$$Q(s,a) = w_1 \cdot f_1(s,a) + w_2 \cdot f_2(s,a) + \ldots + w_n \cdot f_n(s,a) = \vec{w} \cdot \vec{f}(s,a)$$

$$\text{difference} = [R(s,a,s') + \gamma \max_{a'} Q(s',a')] - Q(s,a)$$

$$w_i \leftarrow w_i + \alpha \cdot \text{difference} \cdot f_i(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \text{difference}$$

④ Model-Based Learning

$\hat{T}(s,a,s') = \frac{\#(s,a,s')}{\#(s,a)}$

Generate $\hat{T}$ explore by value / policy iteration

ε-Greedy Policy <- explore with ε, exploit with 1 - ε

ε: too large -> behave randomly -> init ε large, then tune ε; too small -> less exploration -> slow to reach optimal policy

To avoid manual tuning: exploration function: deterministic way to explore.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \cdot [R(s,a,s') + \gamma \max_{a'} f(s',a')]$$

$$f(s,a) = Q(s,a) + \frac{k}{N(s,a)} \quad \rightarrow \text{fixed}$$

model-based RL: estimate the $\hat{T}$ & $\hat{R}$ function with sample attained during exploration. Before using these estimates to solve MDP with value / policy iteration.

model-free RL: estimate V or Q-value of state directly.
- not estimate $\hat{T}$ / $\hat{R}$
- ① can be offline / online RL.

Both of them ① use exploration to learn about environment (R & T).
- ② require online interaction with environment.

TD learning -> offline.

Q learning -> online.

Machine Learning

- Supervised learning
- unsupervised learning

procedure: train; validation. -> can modify hyperparameters; test

Naive Bayes -> classification problem

$$\text{prediction}(f_1, \ldots f_n) = \operatorname*{argmax}_y P(Y=y \mid F_1=f_1, \ldots F_n=f_n)$$

$$= \operatorname*{argmax}_y P(Y=y, F_1=f_1, \ldots F_n=f_n)$$

$$= \operatorname*{argmax}_y P(Y=y) \prod_{i=1}^n P(F_i=f_i \mid Y=y)$$

Parameter Estimation -> MLE -> special case of MAP (max a priori)

assumptions: iid — ① x_i is identical distributed. ② Each sample x_i is conditionally independent of the others. ③ All possible θ are equally likely before we seen any data (uniform prior)

$\frac{d}{d\theta} L(\theta) = 0$

Maximum likelihood for Naive Bayes (Bernoulli case)

$$L(\theta) = \prod_{j=1}^{m} P(F_i = f_i^{(j)} \mid Y = \text{ham}) = \prod_{j=1}^{m} \theta^{f_i^{(j)}} (1-\theta)^{1-f_i^{(j)}}$$

$$\log L(\theta) = \log\theta \sum_{j=1}^{m} f_i^{(j)} + \log(1-\theta) \sum_{j=1}^{m} (1 - f_i^{(j)})$$

$$\frac{d}{d\theta} \log L(\theta) = 0 \Rightarrow \theta = \frac{1}{m} \sum_{j=1}^{m} f_i^{(j)}$$

Smoothing: $P_{MLE}(A) = \frac{\text{Count}(A)}{N}$

$P_{LAP,k}(X) = \frac{\text{Count}(A)+k}{N+k|X|}$ $\quad P_{LAP,R}(X|Y) = \frac{\text{Count}(X,Y)+R}{\text{Count}(Y)+K|X|}$ $\quad P_{LAP,\infty} = \frac{1}{|X|}$

note: ① can reduce overfitting ② will not change the results?

Linear Regression

$$\text{loss}(h_w) = \frac{1}{2} \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \frac{1}{2} \|y - Xw\|_2^2$$

$$\Rightarrow \hat{w} = (X^T X^{-1}) X^T y$$

Perceptron:

Linear Classifier: activation(x) = $h_w(x) = w^T f(x)$ $\begin{cases} + & \text{if } h_w(x) > 0 \\ - & \text{if } h_w(x) < 0 \end{cases}$

Binary Perceptron:

1. Initialize all weights to 0: $\mathbf{w} = \mathbf{0}$
2. For each training sample, with features $\mathbf{f}(\mathbf{x})$ and true class label $y^* \in \{-1, +1\}$, do:
   (a) Classify the sample using the current weights, let y be the class predicted by your current $\mathbf{w}$:
   $$y = \text{classify}(x) = \begin{cases} +1 & \text{if } h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) > 0 \\ -1 & \text{if } h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{f}(\mathbf{x}) < 0 \end{cases}$$
   (b) Compare the predicted label y to the true label y^*:
   - If $y = y^*$, do nothing
   - Otherwise, if $y \neq y^*$, then update your weights: $\mathbf{w} \leftarrow \mathbf{w} + y^*\mathbf{f}(\mathbf{x})$ (mistake positive as negative: $y^* = 1$; mistake negative as positive: $y^* = -1$)
3. If you went through **every** training sample without having to update your weights (all samples predicted correctly), then terminate. Else, repeat step 2

bias: $W^T f(x) + b = 0$.

optimization:

① gradient based method — obj

- Gradient ascent. — maximize. $w \leftarrow w + \alpha \nabla_w f(w)$
- Gradient descent — minimize. $w \leftarrow w - \alpha \nabla_w f(w)$
- stochastic gradient descent. — each iteration. use one sample
- Mini-batch gradient descent. — use size m sample.
- Least squares gradient descent: $\text{loss}(h_w) = \frac{1}{2}\|y - Xw\|_2^2$; $w \leftarrow w - \alpha(-X^T y + X^T X w)$.

② Logistic Regression.

logistic function: $h_w(x) = \frac{1}{1+e^{-w^T x}}$

Softmax function: $P(y=i \mid f(x); w) = \frac{e^{w_i^T f(x)}}{\sum_{k=1}^{K} e^{w_k^T f(x)}}$

$$\ell(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{i=1}^{n} P(y_i \mid \mathbf{f}(\mathbf{x}_i); \mathbf{w}) = \prod_{i=1}^{n} \prod_{k=1}^{K} \left(\frac{e^{\mathbf{w}_k^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^{K} e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right)^{t_{i,k}}$$

$\sum_k t_{i,k} = 1$.

$$\log \ell(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{i=1}^{n} \sum_{k=1}^{K} t_{i,k} \log \left(\frac{e^{\mathbf{w}_k^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^{K} e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right) \quad \nabla_{\mathbf{w}_j} \log \ell(\mathbf{w}) = \sum_{i=1}^{n} \nabla_{\mathbf{w}_j} \sum_{k=1}^{K} t_{i,k} \log \left(\frac{e^{\mathbf{w}_k^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^{K} e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right) = \sum_{i=1}^{n} \left(t_{i,j} - \frac{e^{\mathbf{w}_j^T \mathbf{f}(\mathbf{x}_i)}}{\sum_{\ell=1}^{K} e^{\mathbf{w}_\ell^T \mathbf{f}(\mathbf{x}_i)}} \right) \mathbf{f}(\mathbf{x}_i).$$

Neural Network

Theorem. (Universal Function Approximators) A two-layer neural network with a sufficient number of neurons can approximate any continuous function to any desired accuracy.

Measuring Accuracy:

binary: $l^{acc}(w) = \frac{1}{n} \sum_{i=1}^{n} \text{sgn}(w \cdot f(x_i)) == y_i)$

P of class i: $\sigma(\mathbf{x}_i)_j = \frac{e^{\mathbf{f}(\mathbf{x}_i)^T \mathbf{w}_j}}{\sum_{\ell=1}^{N} e^{\mathbf{f}(\mathbf{x}_i)^T \mathbf{w}_\ell}} = P(y_i = j \mid \mathbf{f}(\mathbf{x}_i); \mathbf{w})$ $\qquad \ell(\mathbf{w}) = \prod_{i=1}^{n} P(y_i \mid \mathbf{f}(\mathbf{x}_i); \mathbf{w})$

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

ReLu: $f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$

How to choose activation func?

if we want to fit a non-linear func. use non-linear activation func.

CS 188 Final Study Guide — Jeffrey Shen

1 Dynamic Bayes Nets
tracks multiple variables over time, using multiple sources evidence

Idea: Repeat fixed Bayes net structure at each time

Particle Filtering simulating motion of set of particles through state graph to approx prob distr

1) Prediction Step: sample new state from transition model
   particle j state $x_t^{(j)}$ $\quad x_{t+1}^{(j)} \sim P(X_{t+1} \mid x_t^{(j)})$
2) Update Step: weight each sample on evidence, then normalize
   $w^{(j)} = P(e_{t+1} \mid x_{t+1}^{(j)})$

Exact Inference: Calc exact probability more accurate but hard w/ large belief distribution

2 Rational Decisions
Principle of Maximum Expected Utility (MEU): rational agents always select action to max utility

Preferences:

$A \succ B$: prefers A over B $\qquad A \sim B$: indifferent between A and B

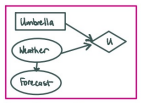
Lottery: A recieved w/ probability p, B w/ probability $1-p$

$L = [p, A; (1-p), B]$

Axioms of Rationality:
1) Orderability: must prefer A or B, or be indifferent — $(A \succ B) \vee (B \succ A) \vee (A \sim B)$
2) Transitivity: if prefers A to B and B to C, prefers A to C — $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$
3) Continuity: if prefers A to B, B to C, lottery L w/ A,C possible st. indifferent between L and B for some p — $A \succ B \succ C \Rightarrow \exists p\, [p, A; (1-p), C] \sim B$
4) Substitutability: indifferent between A and B also indifferent for lottery substitutions — $A \sim B \Rightarrow [p, A; (1-p), C] \sim [p, B; (1-p), C]$
5) Monotonicity: prefers A over B, chooses lottery w/ highest A prob — $A \succ B \Rightarrow (p \geq q \Leftrightarrow [p, A; (1-p), B] \geq [q, A; (1-q), B])$

3 Decision Networks
used to model state, actions, utilities of subsequent states

Chance Nodes: outcome has associated probability
Action Nodes: choices between actions we can choose
Utility Nodes: children of chance & action representing utility



Value of Perfect Information (VPI) expected improvement in decision quality from observing value

VPI $(E' \mid e)$ is value of observing new evidence E' given current evidence e

$$VPI(E' \mid e) = MEU(e, E') - MEU(e) = \left[\sum_{e'} P(e' \mid e) \max_a \sum_s P(s \mid e, e') U(s,a)\right] - \max_a \sum_s P(s \mid e) U(s,a)$$

VPI Properties
1) Nonnegativity: more informed decision — $\forall E', e,\ VPI(E' \mid e) \geq 0$
2) Nonadditivity: can change how much we care about E_k — $VPI(E_j, E_k \mid e) \neq VPI(E_j \mid e) + VPI(E_k \mid e)$
3) Order-independence: order of observation doesn't matter — $VPI(E_i, E_j \mid e) = VPI(E_i, E_i \mid e)$

4 Markov Decision Process (MDP)
Solving nondeterministic search problems

Markov Decision Process properties:
- State Space S
- Actions A
- Transition function $T(s,a,s')$ probability taking action a at state s results in s' $P(s' \mid s, a)$
- Reward function $R(s,a,s')$ pos/neg reward for each step
- terminal state(s)
- start state s_0
- discount factor γ multiply to reward for exponential decay in reward "time constant"

Policy π : $S \to A$, mapping each state to an action
State Optimal Value $U^*(s)$: expected utility of optimal agent from state s
Optimal Q Value $Q^*(s,a)$: expected utility of optimal agent from state s taking action a
Policy Extraction: used to determine policy given some state value function

Value Iteration DP algorithm to compute MDP until convergence of values

1) $\forall s \in S$, set $U_0(s) = 0$
2) Repeat until convergence:

$$\forall s \in S,\ U_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma U_k(s')\right]$$

Runtime: $O(|S|^2 |A|)$

Policy Iteration more efficient method to converge to optimal policy

1) Define an initial policy
2) Use policy evaluation for current policy
3) Use policy improvement to find better policy

$$U^{\pi}(s) = \sum_{s'} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma U^{\pi}(s')\right]$$

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma U^{\pi}(s')\right]$$

5 Machine Learning
learning parameters of specified model given data

Decision Trees simple nonlinear classifier but susceptible to over-fitting

Entropy: uncertainty about variable ↑ entropy ↑ uncertainty

$$H(V) = -\sum_k P(v_k) \log_2 P(v_k)$$

entropy of binary variable: $B(q) = -q \log_2 q - (1-q) \log_2 (1-q)$

Information Gain: info gained by splitting on that feature

$$Gain(Type) = B\left(\frac{p}{p+n}\right) - \sum_{k=1}^{3} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k+n_k}\right)$$

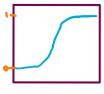
Linear Regression linear classifier for continous var output

L2 Loss function: metric to measure our model using squared difference $(y - h(x))^2$
Least squares: $\vec{w} = (X^T X)^{-1} X^T y$

Logistic Regression linear classifier for categorical variables

Logistic function: $h_w(x) = \frac{1}{1 + e^{-wx}}$

Find optimal using gradients



Models $P(c \mid x)$ where c: class, x: evidence

Perceptron

Linear classifier: classification using linear combination of features
Activation: $activation_w(x) = h_w(x) = w \cdot f(x)$

$$classify(x) = \begin{cases} + & \text{if } h_w(x) > 0 \\ - & \text{if } h_w(x) < 0 \end{cases}$$

Perceptron Algorithm:
1) Initialize all weights to 0: $w = 0$
2) Classify y and compare to true label y^*
   if $y = y^*$, do nothing
   if $y \neq y^*$, update $w \leftarrow w + y^* f(x)$

Naive Bayes model features as Bayes Net, assumes each feature is independent from others

Prediction for class label becomes $prediction(F) = \arg\max_{y_i} P(Y = y_i) \prod_j P(F_j = f_j \mid Y = y_i)$, normalize

Maximum Likelihood Estimation (MLE): given i.i.d., method to learn probability, count

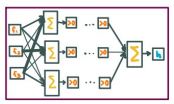
Likelihood: $L(\theta) = \prod_{i=1}^{n} P_\theta(x_i)$ take gradient $\frac{\partial}{\partial \theta} L(\theta) = 0$ to get max, log likelihood $\log L(\theta)$

Laplace smoothing: mitigate overfitting, assume seen k extra each of each outcome

$$P_{LAP,k}(x) = \frac{count(x) + k}{N + k|X|}$$

Neural Networks

Multi-layer perceptron map data to higher dimension then classify
Universal func approximater: two layer NN w/ sufficient neurons can approx any cont. func
can use indicator function of sgn threshold, softmax func to classify
Log Likelihood: $\log L(w) = \prod_{i=1}^{n} \log P(y_i \mid f(x_i); w)$



Activation functions

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

ReLU: $f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$

Backpropogation determine gradient of output w/ respect to each of inputs
1) Forward Pass: compute values through computation graphs
2) Backwards Pass: compute gradients taking advantage of chain rule

6 Reinforcement Learning
method for solving MDP w/o transition & reward functions

Episode: collection of samples which are (s,a,s',r) tuples
Model-based learning: estimate transition, reward functions w/ samples before using policy/value iter
- count times arrived in state and normalize counts, Law of Large Numbers will converge on optimal

Model-free learning: estimate Q values directly w/o constructing model

↳ Passive reinforcement learning: agent given policy to follow and learns values of state exploring
1) Direct Evaluation: fix policy π and have agent experience, can compute by averages
2) Temporal Difference (TD) learning: learning from every experience

$$sample = R(s, \pi(s), s') + \gamma V^{\pi}(s')$$
$$V_n^{\pi}(s) \leftarrow (1-\alpha) V_{n-1}^{\pi}(s) + \alpha \cdot sample$$

↳ Active Reinforcement learning: learning agent can use feedback to iteratively update policy
3) Q-Learning: Bypass model by directly learning Q values

Q-Value Iteration:

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

Q value samples:

$$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$
$$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha \cdot sample$$

Feature Based Representation: allow model to generalize learning experiences, store as linear func

Weight update

$$w_i \leftarrow w_i - \alpha \cdot sample \frac{\partial Q_w}{\partial w_i}$$

negative for gradient decent

Exploration vs. Exploitation

ε-greedy: $0 \leq \varepsilon \leq 1$, act randomly w/ prob ε, should be lowered over time

Exploration functions: modified update

$$Q(s,a) = (1-\alpha) Q(s,a) + \alpha \left[R(s,a,s') + \gamma \max_{a'} f(s',a')\right]$$
$$f(s,a) = Q(s,a) + \frac{k}{N(s,a)}$$

k ← predetermined; $N(s,a)$ ← # times Q state visited

Regret: metric to measure model, difference between total reward for optimal and reward in our model