CS47100
Introduction to Artificial Intelligence

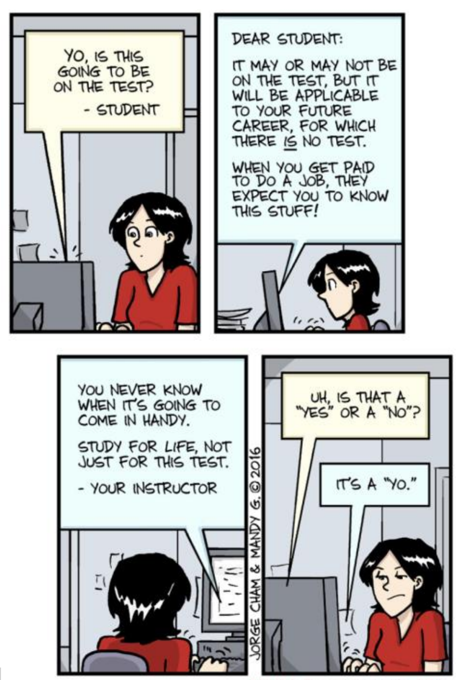
Lecture 39: Final Review

Dec 6th, 2024

1

Final Exam

- **Time:**
  - Thu Dec 12, 8:00 am – 10:00 am
- **Location:**
  - BHEE 129
- **What to bring:**
  - Cheat Sheet
  - Calculator
  - University ID
  - Pen, Pencil



2

Final Exam

- **6 Problems (Subject to change, but small variations):**
  - First order logic
  - Bayes Net
  - MDP
  - RL
  - Supervised Learning
  - Multiple Choice

Exam Tips

- Answer box's size does not correlate with the solution
- Don't have to write the exam in order
  - Its order is based on course content
- Start with the content you are most familiar with
- Study the examples covered in lecture & homework

What to write on the page of note?

Recommend:

- Definitions
- Equations
- Algorithms / Procedure to a type of problem
- General facts/summary about a topic
- Organize the note based on topics

Not recommended:

- Print this slide into one page…
- Putting homework problems/solutions
- Writing too much (Too difficult to find during an exam)

First order logic

Basic syntactic elements of FOL

- Constants John, Richard, 2,...
- Predicates Brother, >,...
- Functions Sqrt, LeftLegOf,...
- Variables x, y, a, b,...
- Connectives $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- Equality =
- Quantifiers $\forall, \exists$

Constants: objects

Relation: set of tuples of objects that are related e.g., {<Richard,John>, <John, Richard>}

Function: relations where a given set of objects are related to exactly one object e.g., {<John>→John's left leg, <Richard>→Richard's left leg}

Universal quantification

- $\forall$<variables> <sentence>
- Everyone in CS471 is smart: $\forall x$ in(x, CS471) $\Rightarrow$ Smart(x)
- $\forall x$ P is true in a model m iff P is true with x interpreted as *each* possible object in the model
- Roughly speaking, equivalent to the **conjunction** of **instantiations** of P

$$
\begin{aligned}
& In(John, CS471) \Rightarrow Smart(John) \\
\wedge \quad & In(Jane, CS471) \Rightarrow Smart(Jane) \\
\wedge \quad & In(CS471, CS471) \Rightarrow Smart(CS471) \\
\wedge \quad & \ldots
\end{aligned}
$$

Existential quantification

- ∃<variables> <sentence>
- Someone in CS471 is smart: ∃x In(x, CS471) ∧ Smart(x)
- ∃x P is true in a model m iff P is true with x interpreted as *some* possible object in the model
- Roughly speaking, equivalent to the **disjunction** of **instantiations** of P

In(John, CS471) ∧ Smart(John)
∨ In(Jane, CS471) ∧ Smart(Jane)
∨ In(CS471 , CS471) ∧ Smart(CS471)
∨ …

Unification

- Two sentences α, β can be **unified** with substitution θ if
  - $\mathrm{SUBST}(\theta, \alpha) = \mathrm{SUBST}(\theta, \beta)$
  - $\theta = \{x/v\}$ is a substitution of variable x with v

α	β	θ
Knows(John,x)	Knows(John, Jane)	{x/Jane}
Knows(John,x)	Knows(y, Steve)	{x/Steve,y/John}
Knows(John,x)	Knows(y, Mother(y))	{y/John,x/Mother(John)}
Knows(John,x)	Knows(x,Steve)	{fail}

Resolution

- **First convert to CNF** (as with propositional logic)
  - Conjunction clauses of disjunction terms
  - ($Term_1 \lor Term_2 \lor Term_3$) ∧ ($Term_4 \lor Term_5$)

- **Negate the conclusion** α and add it to the KB, i.e., add $\neg\alpha$ to KB

- **Resolution Rule**

$$\frac{\ell_1 \lor \ldots \lor \ell_k, \quad m_1 \lor \ldots \lor m_n}{\ell_1 \lor \ldots \lor \ell_{i-1} \lor \ell_{i+1} \lor \ldots \lor \ell_k \lor m_1 \lor \ldots \lor m_{j-1} \lor m_{j+1} \lor \ldots \lor m_n}$$

- where ℓ_i and m_j are **complementary** given a **substitution.**
  - i.e., ¬SUBST(θ, l_i) = SUBST(θ, m_j)
- Apply resolution and **find contradiction**, i.e., arrive at an empty clause

$(\alpha \land \beta) \equiv (\beta \land \alpha)$	commutativity of $\land$
$(\alpha \lor \beta) \equiv (\beta \lor \alpha)$	commutativity of $\lor$
$((\alpha \land \beta) \land \gamma) \equiv (\alpha \land (\beta \land \gamma))$	associativity of $\land$
$((\alpha \lor \beta) \lor \gamma) \equiv (\alpha \lor (\beta \lor \gamma))$	associativity of $\lor$
$\neg(\neg\alpha) \equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \lor \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \land \beta) \equiv (\neg\alpha \lor \neg\beta)$	De Morgan
$\neg(\alpha \lor \beta) \equiv (\neg\alpha \land \neg\beta)$	De Morgan
$(\alpha \land (\beta \lor \gamma)) \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma))$	distributivity of $\land$ over $\lor$
$(\alpha \lor (\beta \land \gamma)) \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma))$	distributivity of $\lor$ over $\land$

Resolution --- Example

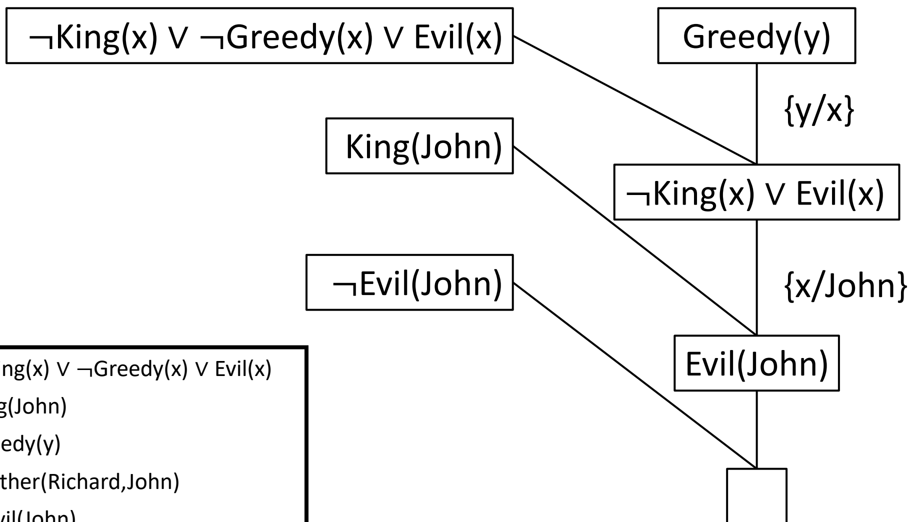
- ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
- King(John)
- ∀x Greedy(x) = ∀y Greedy(y) (**Standardizing apart**: eliminates overlap of variables)
- Brother(Richard,John)

α: Evil(John)

Convert to CNF

- ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
  - ¬(King(x) ∧ Greedy(x)) ∨ Evil(x) **(Implication Elimination)**
  - ¬King(x) ∨ ¬Greedy(x) ∨ Evil(x) **(De Morgan)**

Resolution --- Example



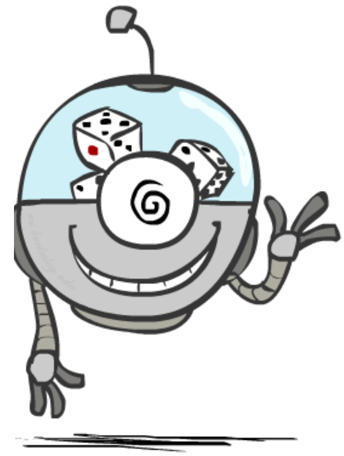
¬King(x) ∨ ¬Greedy(x) ∨ Evil(x)
King(John)
Greedy(y)
Brother(Richard,John)
¬Evil(John)

Probability

Probabilities

Probability space $(\Omega, \mathcal{F}, P)$

- **Sample Space** Ω (Possible outcomes)
- $\mathcal{F}$ **= Events** = a set of subsets of Ω
  - An **event** is a subset of Ω
- P = **Probability measure** on $\mathcal{F}$
- $P(A)$ = Probability of event A.



Dice roll:

Ω={1,2,3,4,5,6}

Event A = Getting 1 = {1}. $P(A) = P(\{1\}) = \frac{1}{6}$

Event B = Getting 1 or 3 = {1,3} $P(B) = P(\{1,3\}) = \frac{2}{6}$

Expectations

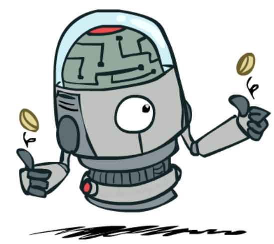
- The **expected value** of a random variable is the average, weighted by the probability distribution over outcomes
- **Example:** How long to get to the airport?

Time:	65 min		75 min		90 min		
	x	+	x	+	x	➔	76.25 min
Probability:	0.25		0.50		0.25		



Independence

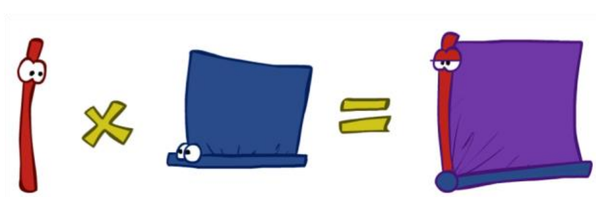
- Two variables are **independent** if:
  - This says that their joint distribution factors into a product two simpler distributions $\forall x, y : P(x, y) = P(x)P(y)$
  - Another form: $\forall x, y : P(x|y) = P(x)$
  - We write: $X \perp\!\!\!\perp Y$
- Independence is a simplifying modeling assumption
  - Empirical joint distributions at best “close” to independent
  - What could we assume for {Weather, Traffic, Cavity, Toothache}?



The Product Rule

- Sometimes have conditional distributions but want the joint distribution

$$P(y)P(x|y) = P(x, y) \qquad \Longleftrightarrow \qquad P(x|y) = \frac{P(x, y)}{P(y)}$$



The Chain Rule

- More generally, can always write any joint distribution as an incremental product of conditional distributions

$$P(x_1, x_2, x_3) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$$

$$P(x_1, x_2, \ldots x_n) = \prod_i P(x_i|x_1 \ldots x_{i-1})$$

- Why is this always true?
  - Repeated application of product rule

Bayes' Rule

- Two ways to factor a joint distribution over two variables:

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

- Dividing P(y), we get:

$$P(x|y) = \frac{P(y|x)}{P(y)}P(x)$$

- Why is this at all helpful?
  - Let's us build one conditional from its reverse
  - Often one conditional is tricky but the other one is simple
  - Foundation of many systems we'll see later

Reverend Thomas Bayes

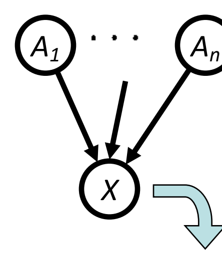
Bayesian Network

Bayesian Network (BN) Semantics

- A set of nodes, one per random variable X
- A directed, acyclic graph
- A conditional distribution for each node
  - A collection of probability distributions over X, one for each combination of parents' values

$$P(X|a_1 \ldots a_n)$$

  - CPT: conditional probability table
  - Description of a noisy "causal" process



$$P(X|A_1 \ldots A_n)$$

Bayesian network = Topology (graph) + Local Conditional Probabilities

Probabilities in BNs

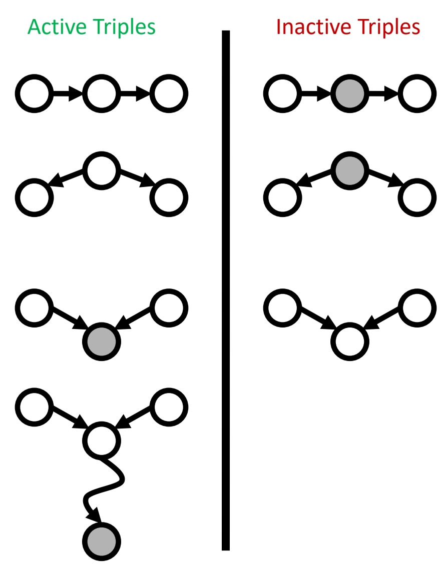
- Why are we guaranteed that BN results in a proper joint distribution?

$$P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i | \mathit{parents}(X_i))$$

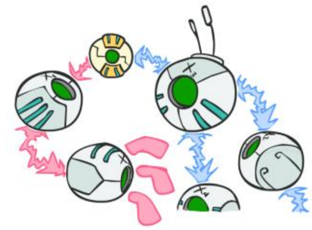
- Chain rule (valid for all distributions): $P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i | x_1 \ldots x_{i-1})$
- Assume conditional independence: $P(x_i | x_1, \ldots x_{i-1}) = P(x_i | \mathit{parents}(X_i))$
  - → Consequence: $P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i | \mathit{parents}(X_i))$
- A BN cannot represent all possible joint distributions
  - The topology enforces certain conditional independencies

Active / Inactive Paths

- **Question:** Are X and Y conditionally independent given evidence variables {Z}?
  - Yes, if X and Y "d-separated" by Z
  - Consider all (undirected) paths from X to Y
  - No active paths = independence!
- A path is active if each triple is active:
  - Causal chain $A \rightarrow B \rightarrow C$ where B **is unobserved** (either direction)
  - Common cause $A \leftarrow B \rightarrow C$ where B **is unobserved**
  - Common effect (aka v-structure) $A \rightarrow B \leftarrow C$ where B or one of its descendants is **observed**
- In a path, if one triple is inactive, the entire path is inactive



D-Separation



- Query: $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \ldots, X_{k_n}\}$?
- Check all (undirected!) paths between X_i and X_j
  - If one or more active, then independence **not guaranteed**

    $X_i \not\perp\!\!\!\perp X_j | \{X_{k_1}, \ldots, X_{k_n}\}$ Maybe?
  - Otherwise (i.e., if all paths are **inactive**), then **independence is guaranteed**

    $X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \ldots, X_{k_n}\}$

Simple sampling

- Given an empty network (no evidence)
  - Begin with nodes without parents
  - Sample from CPDs sequentially to instantiate all nodes
- This will produce one sample from the joint distribution
- Do this many times to produce an empirical distribution that approximates the full joint distribution.

Rejection sampling

- Sample the network as before
  - but **discard samples** that don't correspond with the evidence.
- Similar to real-world estimation procedures, which use observation
- However, it is hopelessly expensive for large networks where $P(e)$ is small
  - Very few observations consistent with evidence…

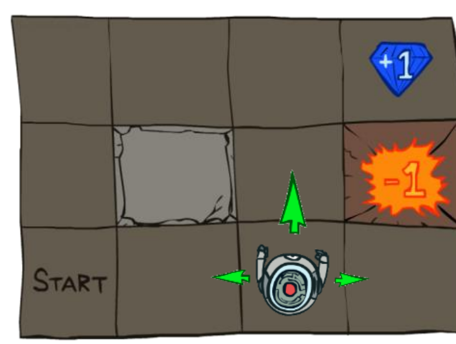
Likelihood weighting

- Do simple sampling as before…
  - But only generate samples that are consistent with the evidence
  - And weight the likelihood of each sample based on the evidence
- More efficient than rejection-based sampling since we use all the samples generated
- … but performance degrades as number of evidence variables increases
- Cannot deal with complex evidence (e.g., rain or sprinkler on).

Markov Decision Process

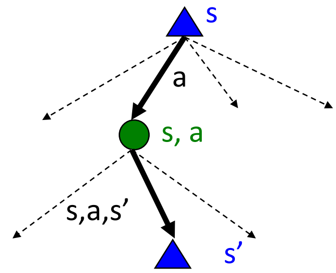
Markov Decision Processes

- An MDP is defined by:
  - Set of states $s \in S$
  - Set of actions $a \in A$
  - Transition function $T(s, a, s')$
    - Probability that a from s leads to s', i.e., $P(s'|\, s, a)$
    - Also called the model or the dynamics
  - Reward function $R(s, a, s')$
    - Sometimes just $R(s)$ or $R(s')$
  - Start state s_0
  - Maybe a terminal state
- MDPs are non-deterministic search problems



Values of States

- Fundamental operation: compute the (expectimax) value of a state
  - Expected utility under optimal action
  - Average sum of (discounted) rewards
  - This is just what expectimax computed



- Recursive definition of value (Bellman Equation):

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

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

Computing Actions from Values

- Let's imagine we have the optimal values $V^*(s)$

- How should we act?
  - Not so obvious…?

- We need to do a mini-expectimax (one step)

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma V^*(s')\right]$$

- This is called policy extraction, since it gets the policy implied by the values

Policy Iteration

- Evaluation: For fixed current policy π, find values with policy evaluation:
  - Solve n linear equations with n unknowns
  - Iterate until values converge:

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} P(s'|s, \pi_i(s)) \left[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')\right]$$

- Improvement: For fixed values, get a better policy using policy extraction
  - One-step look-ahead:

$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma V^{\pi_i}(s')\right]$$

Reinforcement Learning

Reinforcement Learning

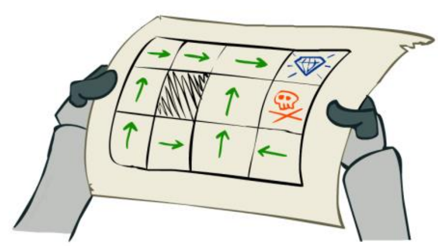
- Assume a Markov decision process (MDP):
  - A set of states S
  - A set of actions (per state) A
  - A model $T(s, a, s')$
  - A reward function $R(s, a, s')$

- Still looking for a policy $\pi(s)$

- **New twist: T or R is unknown**
  - I.e., we don't know which states are good or what the actions do
  - Must **try** out actions and states **to learn**

Passive Reinforcement Learning

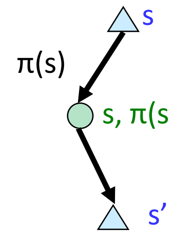
- **Simplified task:** policy evaluation
  - **Input:** a fixed policy $\pi(s)$
  - You don't know the transitions $T(s, a, s')$
  - You don't know the rewards $R(s, a, s')$
  - **Goal:** learn the state values $V^{\pi}(s)$



- **In this case:**
  - Learner is "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - This is NOT offline planning. Agent **must take** actions in the world.

Temporal Difference (TD) Learning

- **Big idea:** learn from every experience
  - Update $V(s)$ each time we experience a transition (s, a, s', r)
  - Likely outcomes s' will contribute updates more often



- Temporal difference learning of values
  - Policy still fixed, still doing evaluation!
  - Move values toward value of whatever successor occurs: running average

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

Update to V(s): $V^{\pi}(s) \leftarrow (1 - \alpha)V^{\pi}(s) + (\alpha) sample$

Same update: $V^{\pi}(s) \leftarrow V^{\pi}(s) + \boxed{\alpha(sample - V^{\pi}(s))}$ "Temporal Difference"

Q-Learning

- Q-Learning: sample-based Q-value iteration

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

- Learn $Q(s,a)$ values as you go
  - Receive a sample (s, a, s', r)
  - Consider your old estimate: $Q(s,a)$
  - Consider your new sample estimate:

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

  - Incorporate the new estimate into a running average:

$$Q(s,a) \leftarrow (1 - \alpha)Q(s,a) + (\alpha)\left[sample\right]$$

this is TD Q-learning

Active Reinforcement Learning

- **Full Reinforcement learning:** find optimal policies (like value iteration)
  - You don't know the transitions $T(s, a, s')$
  - You don't know the rewards $R(s, a, s')$
  - You choose the actions now
  - **Goal: learn the optimal policy / values**

- **In this case:**
  - Learner makes choices
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning. You must take actions in the world and find out what happens…

How to Explore?

- Several schemes for forcing exploration
  - Simplest: random actions (ε-greedy)
    - Every time step, flip a coin
    - With (small) probability ε, act randomly
    - With (large) probability 1-ε, act on current policy

- Problems with random actions?
  - You do eventually explore the space but keep acting randomly once learning is done..
  - **One solution:** lower ε over time
  - Another solution: exploration functions

Exploration Functions

- **When to explore?**
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose value has **not (yet) established**, eventually stop exploring
- Exploration function
  - Takes a value estimate u and a visit count n, and returns an optimistic utility (k>0), e.g., $f(u,n) = u + k/n$

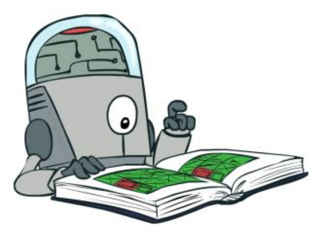
Regular Q-Update: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$

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

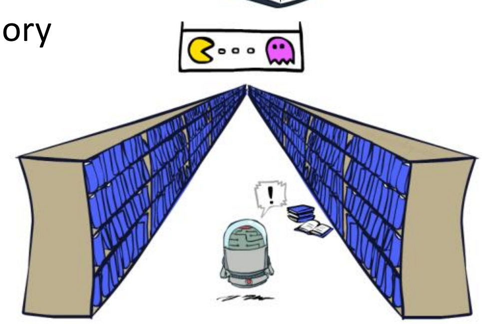
- **Note:** this propagates the "bonus" back to states that lead to unknown states as well!

41

Generalizing Across States



- Basic Q-Learning keeps a table of all Q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the Q-tables in memory



- Instead, we want to **generalize**:
  - Learn about some small number of training states from experience
  - Generalize that experience to **new, similar situations**
  - This is the fundamental idea in machine learning

42

Linear Value Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- Advantage: our experience is summed up in a few powerful numbers

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Disadvantage: states may share features but are very different in value!

Approximate Q-Learning

- Q-learning with linear Q-functions:

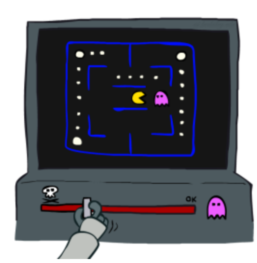
$$\boxed{Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)}$$

transition $= (s, a, r, s')$

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

$Q(s,a) \leftarrow Q(s,a) + \alpha$ [difference] Exact Q's

$w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$ Approximate Q's



- **Intuitive interpretation:**
  - Adjust weights of active features, e.g., if something unexpectedly bad happens, blame the features that were on: "dis-prefer" all states with that state's features

Supervised Learning

K-Nearest Neighbor Algorithm

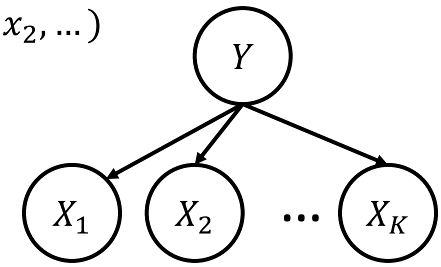
- **Classification Model**:
  - Training set $\mathcal{D} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$
- In Nearest Neighbor, $\theta = \mathcal{D}$ (memorize the training set!)
- Given a "distance metric" $d(x, x')$
  - Find the K closest example to x, denoted as $\{n\} = S_K(x, \mathcal{D})$
- Model:
  - $f_\theta(y|x) = \arg\max_y P(Y = y|x)$
  - $P(Y = y|x; D) = \frac{1}{K} \sum_{n \in S_K} \mathbf{1}[y^{(n)} = y]$

$\mathbf{1}$(statement) = 1 if statement is true, else 0

Naive Bayes --- Prediction

- **Naïve Bayes**: features are conditionally independent given target

- How to make a prediction?
  - Compute the probability $\mathrm{arg}max_y \; P(Y = y|x_1, x_2, \dots)$

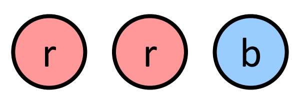


- How to compute $P(Y|x_1, x_2, \dots)$?
  - $P(Y|x_1, x_2, \dots) = P(Y, x_1, x_2, \dots)/P(x_1, x_2, \dots)$
    $\propto P(Y, x_1, x_2, \dots)$

  - $P(Y, x_1, x_2, \dots) = P(Y) \prod_k P(x_k|Y)$

Laplace's estimate (extended)

- Pretend you saw every outcome k extra times



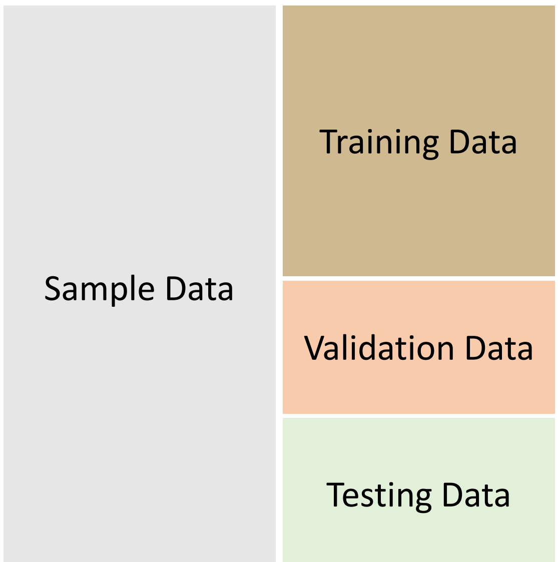
$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|}$$

  - What's Laplace with $k = 0$? $P_{LAP,0}(X) =$
  - k is the strength of the prior $P_{LAP,1}(X) =$ $P_{LAP,100}(X) =$

- Laplace for conditionals:
  - Smooth each condition separately:

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

Training/validation/test cycle



Estimate model parameters θ from training data

$$P_{\text{ML}}(x) = \frac{\text{count}(x)}{\text{total samples}}$$

Evaluation function (applied to training data)

$$L(x, \theta) = \prod_i P_\theta(x_i)$$

Estimate model hyperparameters from validation data

Evaluation function (applied to validation data)

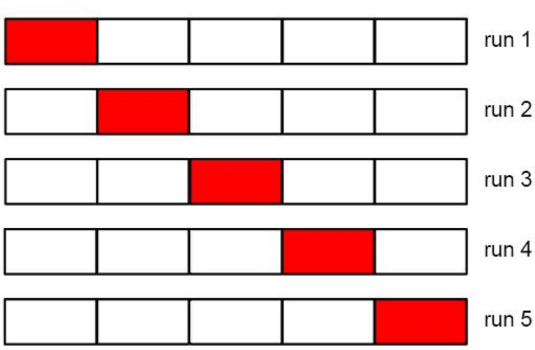
$$\frac{1}{N}\sum_{n=1}^{N} \mathbf{1}(y^{(n)} = f_{\theta,k}(x^{(n)}))$$

Apply model to test data (using chosen θ and hyperparameters)

Evaluation function (applied to test data)

$$\frac{1}{N}\sum_{n=1}^{N} \mathbf{1}(y^{(n)} = f_{\theta,k}(x^{(n)}))$$

S-fold cross validation



White: training data; red: validation data.

***Every* data point** serves in the training and the validation dataset:

- Split data into ***S*** equal parts.
- Use each part **in turn** as a validation dataset and use the others as a training dataset.
- Choose the hyperparameter leading to **best average performance**.

- **Leave-one-out cross validation**: every data point is used as the validation once.
  - I.e., $S = |\mathcal{D}|$

Last slide!

- **Course Evaluation**
  - Survey End Date: 12/8/2024
- **Time:**
  - Thu 12/12 08:00AM - 10:00AM
- **Location:**
  - BHEE 129
- **What to bring:**
  - Cheat Sheet
  - Calculator
  - University ID
  - Pen, Pencil