Markov Decision Processes (MDPs)

Machine Learning – CSE546

Carlos Guestrin
University of Washington

December 2, 2013

Markov Decision Process (MDP) Representation

position of peasant

- State space:
  - Joint state **x** of entire system

  $x = (x_1, \dots x_n)$ ↑ gold

- Action space:
  - Joint action **a**= {a_1,…, a_n} for all agents

  build castle …

- Reward function:
  - Total reward R(**x**,**a**)
    - sometimes reward can depend on action

  positive reward when x is win game

- Transition model:
  - Dynamics of the entire system P(**x**'|**x**,**a**)

  have gold no castle

  build castle

  have castle but no gold

want a policy
$\pi(x) \Rightarrow a$
what action at each state

Discount Factors $\gamma \in [0,1)$

People in economics and probabilistic decision-making do this all the time.

The "Discounted sum of future rewards" using discount factor γ" is

(reward now) +

γ (reward in 1 time step) +

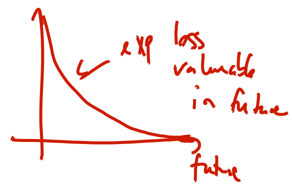
γ^2 (reward in 2 time steps) +

γ^3 (reward in 3 time steps) +

:

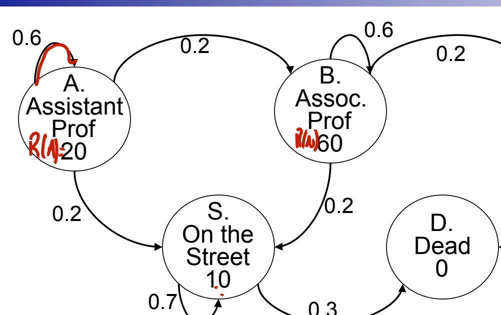
: (infinite sum)

maximize sum of discounted rewards



The Academic Life

Assume Discount Factor $\gamma = 0.9$



Define:

V_A = Expected discounted future rewards starting in state A $= 20 + \gamma(0.6 V_A + 0.2 V_B + 0.2 V_S)$

V_B = Expected discounted future rewards starting in state B $= 60 + \gamma(0.6 V_B + 0.2 V_S + 0.2 V_T)$

V_T = " " " " " " " T

V_S = " " " " " " " S

V_D = " " " " " " " D

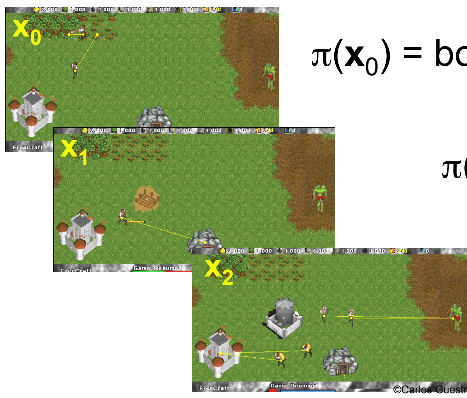
n states.
n unknowns
n equations
linear
⇒ e.g. matrix inversion

How do we compute V_A, V_B, V_T, V_S, V_D ?

Policy

Policy: $\pi(\mathbf{x}) = \mathbf{a}$

At state $\mathbf{x}$, action $\mathbf{a}$ for all agents



$\pi(\mathbf{x}_0)$ = both peasants get wood

$\pi(\mathbf{x}_1)$ = one peasant builds barrack, other gets gold

$\pi(\mathbf{x}_2)$ = peasants get gold, footmen attack

Value of Policy

or $\gamma < 1$

Value: $V_\pi(\mathbf{x})$

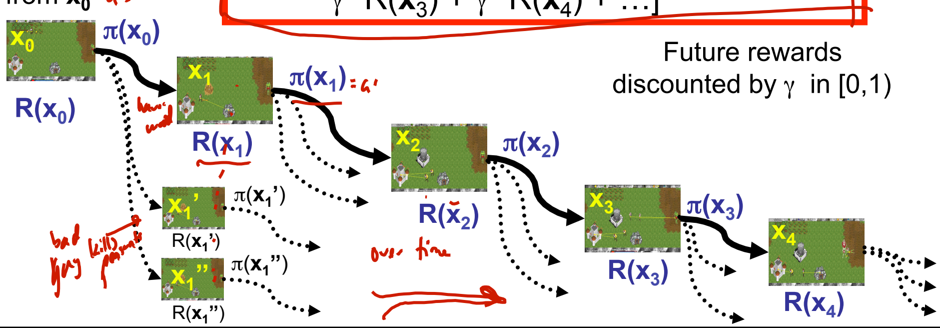
Expected long-term reward starting from $\mathbf{x}$

and act according to π

formal view of recursion

$$V_\pi(\mathbf{x}_0) = E_\pi[R(\mathbf{x}_0) + \gamma\, R(\mathbf{x}_1) + \gamma^2\, R(\mathbf{x}_2) + \gamma^3\, R(\mathbf{x}_3) + \gamma^4\, R(\mathbf{x}_4) + \ldots]$$

Start from $\mathbf{x}_0$ a:



Future rewards discounted by γ in [0,1)

Computing the value of a policy

$\boxed{V_\pi(\mathbf{x_0}) = \mathbf{E}_\pi[R(\mathbf{x}_0) + \gamma\, R(\mathbf{x}_1) + \gamma^2\, R(\mathbf{x}_2) + \gamma^3\, R(\mathbf{x}_3) + \gamma^4\, R(\mathbf{x}_4) + \ldots]}$

- Discounted value of a state:
  - value of starting from x_0 and continuing with policy π from then on

$$\begin{aligned} V_\pi(x_0) &= E_\pi[R(x_0) + \gamma R(x_1) + \gamma^2 R(x_2) + \gamma^3 R(x_3) + \cdots] \\ &= E_\pi[\sum_{t=0}^{\infty} \gamma^t R(x_t)] \end{aligned}$$

- A recursion!

$$V_\pi(x_0) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R(x_t)\right] = E_\pi\left[R(x_0) + \gamma \underbrace{\sum_{t=1}^{\infty} \gamma^{t-1} R(x_t)}\right]$$

$$= R(x_0) + \gamma \underbrace{E_\pi\left[R(x_1) + \sum_{t=2}^{\infty} \gamma^{t-1} R(x_t)\right]}_{V_\pi(x_1)}$$

$$= R(x_0) + \gamma \sum_{x_1} P(x_1 \mid x_0, \underset{a=}{\pi(x_0)})\, V_\pi(x_1)$$

n equations
n unknowns
linea eq's

what's V_1?
don't know, must average

Simple approach for computing the value of a policy: Iteratively

$$V_\pi(x) = R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x')$$

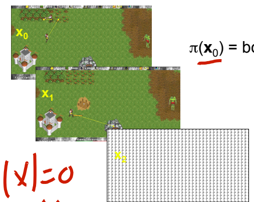
- Can solve using a simple convergent iterative approach: (a.k.a. dynamic programming) eg. $V^0(x) = R(x)$
  - Start with some guess V^0
  - Iteratively say:
    - $V_\pi^{t+1}(x) \leftarrow R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi^t(x')$
  - Stop when $||V^{t+1}-V^t||_\infty < \varepsilon$
    - means that $||V_\pi-V^{t+1}||_\infty < \varepsilon/(1-\gamma)$

iterate

But we want to learn a **Policy**

- So far, told you how good a policy is…
- But how can we choose the best policy???

Policy: $\pi(\mathbf{x}) = \mathbf{a}$ → At state **x**, action **a** for all agents



$\pi(\mathbf{x}_0)$ = both peasants get wood

$\pi(\mathbf{x}_1)$ = one peasant builds barrack, other gets gold

$\pi(\mathbf{x}_2)$ = peasants get gold, footmen attack

- Suppose there was only one time step:
  - world is about to end!!! ← $V^{t+1}(x) = 0 \ \forall x$
  - select action that maximizes reward! $V(x) = \max_a R(x,a)$

Unrolling the recursion

- Choose actions that lead to best value in the long run
  - Optimal value policy achieves optimal value V*

$$V^*(x_0) = \max_{a_0} R(x_0,a_0) + \gamma E_{a_0}[\max_{a_1} R(x_1) + \gamma^2 E_{a_1}[\max_{a_2} R(x_2) + \cdots]]$$

$$V^*(x_0) = \max_{a_0} R(x_0,a_0) + \gamma E_{x_1|a_0,x_0}[V^*(x_1)] \quad V^*(x_1)$$

$\Downarrow$

$$V^*(x_0) = \max_{a_0} R(x_0,a_0) + \gamma \sum_{x_1} P(x_1|x_0,a_0)\, V^*(x_1)$$

n states

n equations ← one per state

n unknowns ← $V^*(x_i)$

non-linear eqns.

⇒ can't use matrix inversion

Bellman equation

- Evaluating policy π:

$$V_\pi(x) \;=\; R(x) + \gamma \sum_{x'} P(x' \mid x, a = \pi(x)) V_\pi(x')$$

- Computing the optimal value V^* - Bellman equation

$$\underbrace{V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')}$$

if you can solve for $V^*(x)$ $\forall x$

Optimal Long-term Plan

Optimal value function $V^*(\mathbf{x})$ → Optimal Policy: $\pi^*(\mathbf{x})$

Optimal policy:

$$\pi^*(\mathbf{x}) = \arg\max_{a} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

Interesting fact – Unique value

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

- *Slightly surprising fact*: There is only one V^* that solves Bellman equation! argmax may not be unique
  - there may be many optimal policies that achieve V^*
- *Surprising fact*: optimal policies are good everywhere!!!

$$V_{\pi^*}(x) \geq V_\pi(x), \quad \forall x, \quad \forall \pi$$

any optimal policy

Solving an MDP

Solve Bellman equation → Optimal value $V^*(\mathbf{x})$ → Optimal policy $\pi^*(\mathbf{x})$

$$V^*(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^*(\mathbf{x}')$$

Bellman equation is non-linear!!!

Many algorithms solve the Bellman equations:

- Policy iteration [Howard '60, Bellman '57]
- Value iteration [Bellman '57]
- Linear programming [Manne '60]
- …

Value iteration (a.k.a. dynamic programming) – the simplest of all

next state

$$V^*(x) = \max_a R(x,a) + \gamma \sum_{x'} P(x' \mid x, a \cancel{= \pi(x)}) V^*(x')$$

- Start with some guess V^0 e.g., $(x) = \max_a R(x,a)$
- Iteratively say:
  - $V^{t+1}(x) \leftarrow \max_a R(x,a) + \gamma \sum_{x'} P(x' \mid x,a) V^t(x')$ iterate
- Stop when $||V^{t+1}-V^t||_\infty < \varepsilon$ ← iterations don't change much
  - means that $||V^*-V^{t+1}||_\infty < \varepsilon/(1-\gamma)$ ⇓ close to true value

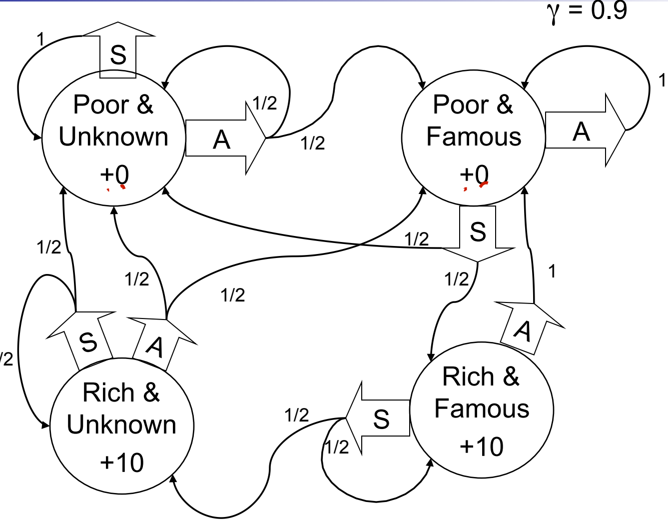
no local optima problem

A simple example

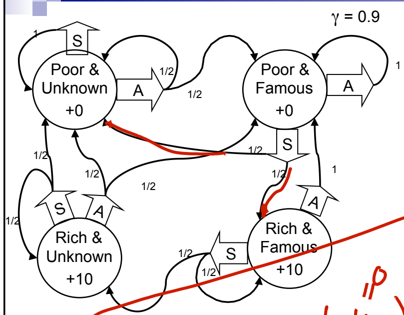
$\gamma = 0.9$

You run ~~a~~ ~~startup~~ your ~~company~~. like

In every state you must choose between Saving :invest money or Advertising.



Let's compute $V_t(x)$ for our example



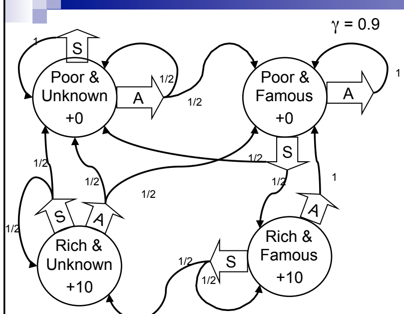
t	V^t(PU)	V^t(PF)	V^t(RU)	V^t(RF)
1	0	0	10	10
2		4.5		
3				
4				
5				
6				

$$V^2(PF) = \max_a \begin{cases} A: & \underbrace{0 + 0.9\,(V^1(PF))}_{0} \\ S: & 0 + 0.9\,(0.5\,V^1(PU) + 0.5\,V^1(RF)) = 4.5 \end{cases}$$

$V^1(PU) = 0$, $V^1(RF) = 10$

$$V^{t+1}(\mathbf{x}) = \max_{\mathbf{a}} R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^{t}(\mathbf{x}')$$

Let's compute $V_t(x)$ for our example



t	V^t(PU)	V^t(PF)	V^t(RU)	V^t(RF)
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	9.46	17.44	25.08
4	5.17	13.61	20.17	29.13
5	8.45	16.91	22.88	32.19
6	11.41	19.62	25.43	34.78
∞	31.59	38.60	44.02	54.02

$V^*(x)$

RF

$$V^{t+1}(\mathbf{x}) = \overset{\text{arg}}{\max_{\mathbf{a}}}\, R(\mathbf{x},\mathbf{a}) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x},\mathbf{a}) V^{t}(\mathbf{x}')$$

What you need to know

- What's a Markov decision process
  - state, actions, transitions, rewards
  - a policy
  - value function for a policy
    - computing V_π
- Optimal value function and optimal policy
  - Bellman equation
- Solving Bellman equation
  - with value iteration, policy iteration and linear programming

Acknowledgment

- This lecture contains some material from Andrew Moore's excellent collection of ML tutorials:
  - http://www.cs.cmu.edu/~awm/tutorials

Announcement

- Poster session 3-5pm CSE Atrium:
  - Arrive 15mins early
  - Everyone must attend
  - Write project number on poster
  - Prepare 2-3 minutes overview of what you did
  - At least 2 instructors will see your project
- Final Project Report
  - Due Monday 9th at 9am
  - See website for details (maximum 8 pages)
  - Be clear about what you did
  - Make it read like a paper

assumed $R(x,a)$
$P(x'|x,a)$ $\Rightarrow$ learned π
don't know

Reinforcement Learning

Machine Learning – CSE546
Carlos Guestrin
University of Washington

December 3, 2013

The Reinforcement Learning task

World: You are in state 34.
Your immediate reward is 3. You have possible 3 actions.

Robot: I'll take action 2.
World: You are in state 77.
Your immediate reward is -7. You have possible 2 actions.

Robot: I'll take action 1.
World: You're in state 34 (again).
Your immediate reward is 3. You have possible 3 actions.

Formalizing the (online) reinforcement learning problem

- Given a set of states **X** and actions **A**
  - in some versions of the problem size of **X** and **A** unknown
- Interact with world at each time step *t*:
  - world gives state $\mathbf{x}_t$ and reward r_t
  - you give next action $\mathbf{a}_t$
- **Goal**: (quickly) learn policy that (approximately) maximizes long-term expected discounted reward

The “Credit Assignment” Problem

I’m in state 43,	reward = 0,	action = 2
“ “ “ 39,	“ = 0,	“ = 4
“ “ “ 22,	“ = 0,	“ = 1
“ “ “ 21,	“ = 0,	“ = 1
“ “ “ 21,	“ = 0,	“ = 1
“ “ “ 13,	“ = 0,	“ = 2
“ “ “ 54,	“ = 0,	“ = 2
“ “ “ 26,	“ = 100,	

Yippee! I got to a state with a big reward! But which of my actions along the way actually helped me get there??

This is the Credit Assignment problem.

Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best I can hope for???
- **Exploitation**: should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at the risk of missing out on some large reward somewhere
- **Exploration**: should I look for a region with more reward?
  - at the risk of wasting my time or collecting a lot of negative reward

Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment, then learn model ($P(\mathbf{x}'|\mathbf{x},\mathbf{a})$ and $R(\mathbf{x},\mathbf{a})$) (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - works quite well in practice when state space is manageable
- Model-free approach:
  - don't learn a model, learn value function or policy directly
  - leads to weaker theoretical results
  - often works well when state space is large

Rmax – A model-based approach

Given a dataset – learn model

Given data, learn (MDP) Representation:

- Dataset: $x_t, a_t \rightarrow r_t, x_{t+1}$
- Learn reward function:
  - R(**x**,**a**) $R(x_t, a_t) = r_t$
- Learn transition model: MLE
  - P(**x'**|**x**,**a**) $\overset{MLE}{=} \dfrac{Count(X_{t+1}=x', X_t=x, A_t=a)}{Count(X_t=x, A_t=a)}$

next state

current state

action

Planning with insufficient information

- Model-based approach:
  - estimate R(**x**,**a**) & P(**x'**|**x**,**a**)
  - obtain policy by value or policy iteration, or linear programming
  - No credit assignment problem!
    - learning model, planning algorithm takes care of "assigning" credit
- What do you plug in when you don't have enough information about a state?
  - don't reward at a particular state $R(\bar{x}, \bar{a})$?
    - plug in 0?
    - plug in smallest reward (R_{min})? ← pessimistic
    - plug in largest reward (R_{max})? ← optimistic
  - don't know a particular transition probability?

Some challenges in model-based RL 2: Exploration-Exploitation tradeoff

- A state may be very hard to reach
  - waste a lot of time trying to learn rewards and transitions for this state
  - after a much effort, state may be useless
- A strong advantage of a model-based approach:
  - you know which states estimate for rewards and transitions are bad
  - can (try) to plan to reach these states
  - have a good estimate of how long it takes to get there

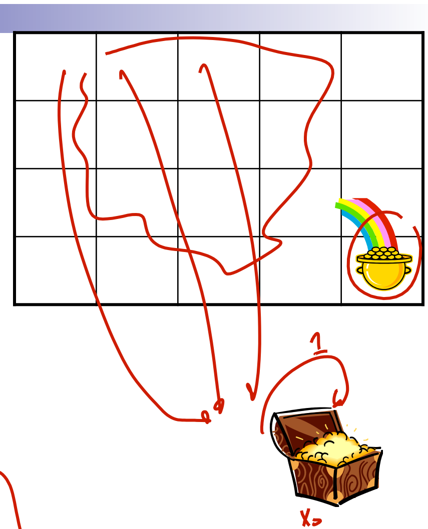
A surprisingly simple approach for model based RL – The Rmax algorithm [Brafman & Tennenholtz]

- **Optimism in the face of uncertainty!!!!**
  - heuristic shown to be useful long before theory was done (e.g., Kaelbling '90)
- If you don't know reward for a particular state-action pair, set it to R_{max}!!!
- If you don't know the transition probabilities $P(\mathbf{x}'|\mathbf{x},\mathbf{a})$ from some some state action pair $\mathbf{x},\mathbf{a}$ assume you go to **a magic, fairytale** new state $\mathbf{x}_0$!!!
  - $R(\mathbf{x}_0,\mathbf{a}) = R_{max}$
  - $P(\mathbf{x}_0|\mathbf{x}_0,\mathbf{a}) = 1$

discount factor γ

Understanding R_{max}

- With R_{max} you either:
  - **explore** – visit a state-action pair you don't know much about
    - because it seems to have lots of potential
  - **exploit** – spend all your time on known states
    - even if unknown states were amazingly good, it's not worth it



- Note: you never know if you are exploring or exploiting!!!

Implicit Exploration-Exploitation Lemma

- **Lemma**: every T time steps, either:
  - **Exploits**: achieves near-optimal reward for these T-steps, or
  - **Explores**: with high probability, the agent visits an unknown state-action pair
    - learns a little about an unknown state
  - T is related to *mixing time* of Markov chain defined by MDP
    - time it takes to (approximately) forget where you started

optimistic → optimal

learn something new

don't know which

The Rmax algorithm

- **Initialization**:
  - Add state $\mathbf{x}_0$ to MDP
  - $R(\mathbf{x},\mathbf{a}) = R_{max}$, $\forall \mathbf{x},\mathbf{a}$
  - $P(\mathbf{x}_0|\mathbf{x},\mathbf{a}) = 1$, $\forall \mathbf{x},\mathbf{a}$
  - all states (except for $\mathbf{x}_0$) are **unknown**

fairy tale

- Repeat
  - obtain policy for current MDP and Execute policy
  - for any visited state-action pair, set reward function to appropriate value
  - if visited some state-action pair $\mathbf{x},\mathbf{a}$ enough times to estimate $P(\mathbf{x}'|\mathbf{x},\mathbf{a})$
    - update transition probs. $P(\mathbf{x}'|\mathbf{x},\mathbf{a})$ for $\mathbf{x},\mathbf{a}$ using MLE
    - recompute policy

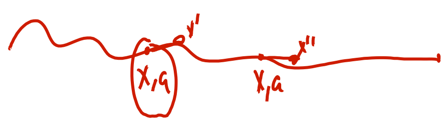
Visit enough times to estimate $P(\mathbf{x}'|\mathbf{x},\mathbf{a})$?

a sample

- How many times are enough?
  - use Chernoff Bound!
- **Chernoff Bound**:
  - $X_1,\ldots,X_n$ are i.i.d. Bernoulli trials with prob. θ
  - $P(|1/n \sum_i X_i - \theta| > \varepsilon) \leq \exp\{-2n\varepsilon^2\}$

every time you see

pick an ε, δ $\Rightarrow$ you known number of samples



Putting it all together

- **Theorem**: With prob. at least 1-δ, Rmax will reach a ε-optimal policy in time polynomial in: num. states, num. actions, T, 1/ε, 1/δ
  - Every T steps:
    - achieve near optimal reward (great!), or
    - visit an unknown state-action pair ! num. states and actions is finite, so can't take too long before all states are known

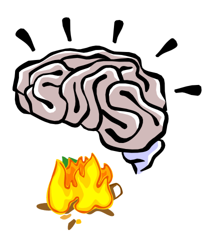
What you need to know about RL…

- Neither supervised, nor unsupervised learning
- Try to learn to act in the world, as we travel states and get rewards
- Model-based & Model-free approaches
- Rmax, a model based approach:
  - Learn model of rewards and transitions
  - Address exploration-exploitation tradeoff
  - Simple algorithm, great in practice

Closing….

What you have learned this quarter

- Learning is function approximation
- Point estimation
- Regression
- LASSO
- Subgradient
- Stochastic gradient descent
- Coordinate descent
- Discriminative v. Generative learning
- Naïve Bayes
- Logistic regression
- Bias-Variance tradeoff
- Decision trees
- Cross validation
- Boosting
- Instance-based learning
- Perceptron
- SVMs
- Kernel trick
- PAC learning
- Bayes nets
    - representation, parameter and structure learning
- K-means
- EM
- Mixtures of Gaussians
- Dimensionality reduction, PCA
- MDPs
- Reinforcement learning



BIG PICTURE

- Improving the performance at some task though experience!!! ☺
  - before you start any learning task, remember the fundamental questions:

What is the learning problem?

From what experience?

What model?

What loss function are you optimizing?

With what optimization algorithm?

Which learning algorithm?

With what guarantees?

How will you evaluate it?

You have done a lot!!!

- And (hopefully) learned a lot!!!
  - Implemented
    - LASSO
    - LR
    - Perceptron
    - Clustering
    - …
  - Answered hard questions and proved many interesting results
  - Completed (I am sure) an amazing ML project
  - And did excellently on the final!
- Now you are ready for one of the most sought-after careers in industry today!!! ☺

Thank You for the Hard Work!!!