Clustering Summary

- Group input training data into clusters where training data are likely to be grouped into the same cluster if they are similar to one another.
- Clustering algorithms: k-means, Expectation-Maximization Algorithm, Self-Organization Maps (topological ordering).
- How many clusters?: Information theoretical problem.
Reinforcement Learning

- In many learning situations you do not have a teacher to help you learn, but may have hints or an evaluation function to evaluate actions.
- Unlike unsupervised learning, some information is available.
- Minsky: learning with a critic or evaluation function.
- Psychology studies of animals: learn from rewards and penalties (action followed by reward (penalty) tends to produce actions that are strengthened (weakened)).
Reinforcement Learning Examples

- Game playing: Chess, Backgammon
- Adaptive controller: Chemical processing plant
- Natural learning: Gazelle calf learns to run
- Robot tasks: Robot picks up objects versus recharging batteries
- Learning to drive
Elements of Reinforcement Learning

- Policy: way learning algorithm behaves (mapping from state to action)
- Reward function: Mapping of state action pair to reward or cost
- Value function: long term reward, total weighted or unweighted reward in present and future
- Model: mimic behavior of environment
History of Reinforcement Learning

- Optimal control: Markov Decision Processes, Dynamic programming, Bellman’s equation
- Trial and error learning: psychology, cognitive science, Artificial Intelligence (Klopf: adaptive behavior to control environment moving towards desirable ends, different from supervised learning)
- Temporal difference learning: credit assignment problem
Consider n-armed bandit problem: at every instant must choose one of n actions with the goal of maximizing rewards.

Expected reward for action $a$, $Q^*(a)$ and estimated value of $t$th play $Q_t(a)$. Set

$$Q_t(a) = \frac{\sum r_i}{k_a}$$

where $k_a$ is number of times action $a$ taken.

- Exploration versus exploitation
- How to choose action $a$
Policies for n-armed bandit problem

- Greedy policy: $Q_t(a^*) = \max_a Q_t(a)$. No exploration, initially does well, but poor long term performance.
- $\varepsilon$-greedy policy: same as greedy policy, but with prob. $\varepsilon$ randomly choose policy. Some exploration, does better than greedy policy.
- Softmax Action selection: weight actions probabilistically with temperature parameter (Gibbs distribution).
- Reinforcement methods: keep track of payoffs (reinforcement as opposed to action-value method).
- Pursuit methods: use both action-value estimates and action preferences.
Summary of example

- Exploitation versus Exploration
  - Algorithms presented
  - What is the proper balance?

- Learning schemes
  - Supervised learning: instructed what to do
  - Evaluative learning: try different actions and observe rewards (allows more control of environment)

- Non associative learning: trial and error learning not associated with situation or state of the problem (only one state)
Reinforcement Learning Model

- Exploration versus exploitation
- Learning can be slow
Finite Markov Decision Processes

- **Parameters**
  - **State:** $X(n) = x(n)$, $N$ states
  - **Action:** $A(n) = a_{ik}$ (action from state $i$ performing action $k$)
  - **Transition probability:** $p_{ij}(a) = P(X(n+1) = j | X(n) = i, A(n) = a)$
  - **Cost:** $r(i,a,j)$ and discount factor $\gamma$, with $0 \leq \gamma < 1$
  - **Policy:** $\pi = \{u(0), u(1), \ldots\}$, policy mapping states into actions (stationary and nonstationary)
Value Functions

- Cost or value function (infinite horizon, discounted)
  \[ J^\pi(i) = E \left( \sum \gamma^n r(x(n), u(x(n)), x(n+1)) \right| x(0)=i) \]
  averaged over Markov chain \( x(1), x(2), \ldots \)

- Action-value function
  \[ Q^\pi(i, a) = E \left( \sum \gamma^n r(x(n), u(x(n)), x(n+1)) \right| x(0)=i, a(0)=a) \]
  averaged over Markov chain \( x(1), x(2), \ldots \)

Find policy \( \pi \) that minimizes \( J^\pi(i) \) for all initial states \( i \).
Recursive expression for value function

\[ J_\pi(i) = E \left( \sum \gamma^n r(x(n),u(x(n)),x(n+1)) | x(0)=i \right) \]
\[ = E(r(i,u(i),j) + \gamma \sum \gamma^n r(x(n+1),u(x(n+1)),x(n+2)) | x(1)=j) \]
\[ = \sum_a \sum_j \pi(i,a) p_{ij}(a) (r(i,a,j) + \gamma J_\pi(j) ) \]

Bellman equation for \( J_\pi \) allows for calculation of value function for policy \( \pi \).
Equation can be solved iteratively or directly.
Optimal Value Function

Want to find optimal policy to maximize value function

\[ J^*(i) = \max_{\pi} J^\pi(i) \]

Can express optimal value function in terms of action-value function as

\[ J^*(i) = \max_{u(i)} Q^*(i,u(i)) \]

where \[ Q^*(i,u(i)) = \max_{\pi} Q^\pi(i,u(i)) \]

Then can find a recursive expression for \( J^*(i) \) by expanding RHS of equation similar to method found in previous slide for value function.
MDP Solution and Bellman Equation

Use dynamic programming, can formulate cost function in terms of Bellman’s Optimality equation

\[ J^*(i) = \max_u E_{x(i)} [r(i,u(i),x(i)) + \gamma J^*(x(i))] \]

Current cost: \( c(i,u(j)) = E_{x(i)} [r(i,u(i),j)] = \sum_{j=1,N} p_{ij} r(i,u(i),j) \)

Rewrite Bellman’s equation

\[ J^*(i) = \max_u [c(i,u(i)) + \gamma \sum_{j=1,N} p_{ij} J^*(j)] \]

System of \( N \) equations with (equation/ state) and minimization