Learning 3:
Bayesian Networks
Reinforcement Learning
Genetic Algorithms

based on material from
Ray Mooney, Daphne
Koller, Kevin Murphy

Project

- Teams: 2-3
- Two components:
  - Agent/World Program
  - Learning Agent

Agent/World Program

- Implements a function
  State x Action -> Reward
  State x Action -> State'
- Representation:
  - State – 8 bits
    may think of this as 8 binary features,
    4 features with 4 possible values, etc.
  - Action – 4 possible actions
  - Reward – integer range from -10…10

Example: Boring_Agent

- State:
  mood: happy, sad, mad, bored
  physical: hungry, sleepy
  personality: optimist, pessimist
- Action:
  smile, hit, tell-joke, tickle
- State x Action -> Reward:
  s x a -> 0
- State x Action -> State
  s x a -> bored = T, all others same as s

Example: Agent_with_Personality

- State:
  mood: happy, sad, mad, bored
  physical: hungry, sleepy
  personality: optimist, pessimist
- Action:
  smile, hit, tell-joke, tickle
- State x Action -> Reward:
  mood = ? x physical <> sleepy, smile -> 5
  mood = ? x physical <> sleepy, tell-joke -> 10 w/ prob. .8,
  -10 w/ prob. .2
  etc.
- State x Action -> State
  s x a -> mood = happy if reward is positive
  s x a -> mood = mad if action is hit
  etc.

Example: Robot Navigation

- State:
  location
- Action:
  forward, back, left, right
- State x Action -> Reward:
  define rewards of states in your grid
- State x Action -> State
  defined by movements

Learning Agent

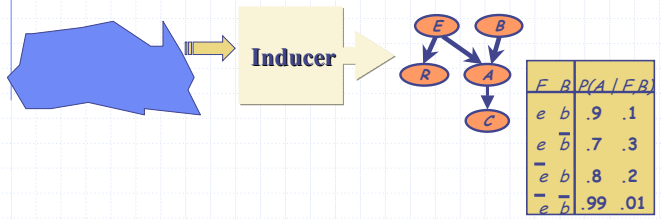
- Calls Agent Program to get a training set
- Learns a function
- Calls Agent Program to get an evaluation set
- Computes Optimal set of actions
- Calls Agent Program to evaluate the set of actions

Schedule

- Thursday, Dec. 4
  In class, hand in a 1 page description of your agent, and some notes on your learning approach
- Friday, Dec. 5
  Electronically submit your agent/world program
- Thursday, Dec. 12
  Submit your learning agent

Learning, cont.

Learning Bayesian networks



we won't cover in this class… ☹

Naïve Bayes

- aka Idiot Bayes
- particularly simple BN
- makes overly strong independence assumptions
- but works surprisingly well in practice…

Bayesian Diagnosis

- suppose we want to make a diagnosis D and there are n possible mutually exclusive diagnosis $d_1, \ldots, d_n$
- suppose there are m boolean symptoms, $E_1, \ldots, E_m$

$$P(d_i \mid e_1,\ldots,e_m) = \frac{P(d_i)P(e_1,\ldots,e_m \mid d_i)}{P(e_1,\ldots,e_m)}$$

how do we make diagnosis?

we need: $P(d_i)$ $P(e_1,\ldots,e_m \mid d_i)$

Naïve Bayes Assumption

- Assume each piece of evidence (symptom) is independent give the diagnosis
- then

$$P(e_1,\ldots,e_m \mid d_i) = \prod_{k=1}^{m} P(e_k \mid d_i)$$

what is the structure of the corresponding BN?

Naïve Bayes Example

- possible diagnosis: Allergy, Cold and OK
- possible symptoms: Sneeze, Cough and Fever

	Well	Cold	Allergy
P(d)	0.9	0.05	0.05
P(sneeze|d)	0.1	0.9	0.9
P(cough|d)	0.1	0.8	0.7
P(fever|d)	0.01	0.7	0.4

my symptoms are: sneeze & cough, what is the diagnosis?

Learning the Probabilities

- aka parameter estimation
- we need
  - $P(d_i)$ – prior
  - $P(e_k|d_i)$ – conditional probability
- use training data to estimate

Maximum Likelihood Estimate (MLE)

- use frequencies in training set to estimate:

$$p(d_i) = \frac{n_i}{N}$$

$$p(e_k \mid d_i) = \frac{n_{ik}}{n_i}$$

where n_x is shorthand for the counts of events in training set

Example:

what is: P(Allergy)?
P(Sneeze| Allergy)?
P(Cough| Allergy)?

D	Sneeze	Cough	Fever
Allergy	yes	no	no
Well	yes	no	no
Allergy	yes	no	yes
Allergy	yes	no	no
Cold	yes	yes	yes
Allergy	yes	no	no
Well	no	no	no
Well	no	no	no
Allergy	no	no	no
Allergy	yes	no	no

Laplace Estimate (smoothing)

- use smoothing to eliminate zeros:

$$p(d_i) = \frac{n_i + 1}{N + n}$$

many other smoothing schemes…

$$p(e_k \mid d_i) = \frac{n_{ik} + 1}{n_i + 2}$$

where n is number of possible values for d and e is assumed to have 2 possible values

Comments

- Generally works well despite blanket assumption of independence
- Experiments show competitive with decision trees on some well known test sets (UCI)
- handles noisy data

Learning more complex Bayesian networks

- Two subproblems:
- learning structure: combinatorial search over space of networks
- learning parameters values: easy if all of the variables are observed in the training set; harder if there are 'hidden variables'

Clustering

- Aka 'unsupervised' learning
- Find natural partitions of the data

Reinforcement Learning

- supervised learning is simplest and best-studied type of learning
- another type of learning tasks is learning behaviors when we don't have a teacher to tell us how
- the agent has a task to perform; it takes some actions in the world; at some later point gets feedback telling it how well it did on performing task
- the agent performs the same task over and over again
- it gets carrots for good behavior and sticks for bad behavior
- called **reinforcement learning** because the agent gets positive reinforcement for tasks done well and negative reinforcement for tasks done poorly

Reinforcement Learning

- The problem of getting an agent to act in the world so as to maximize its rewards.
- Consider teaching a dog a new trick: you cannot tell it what to do, but you can reward/punish it if it does the right/wrong thing. It has to figure out what it did that made it get the reward/punishment, which is known as the **credit assignment** problem.
- We can use a similar method to train computers to do many tasks, such as playing backgammon or chess, scheduling jobs, and controlling robot limbs.

Reinforcement Learning

- for blackjack
- for robot motion
- for controller

Formalization

- we have a state space S
- we have a set of actions a1, …, ak
- we want to learn which action to take at every state in the space
- At the end of a trial, we get some reward, positive or negative
- want the agent to learn how to behave in the environ
actions

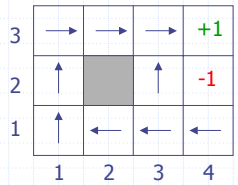
example: Alvinn
state: configuration of the car
learn a steering action for each state

Reactive Agent Algorithm

Repeat:

- s ← sensed state (Accessible or observable state)
- If s is terminal then exit
- a ← choose action (given s)
- Perform a

Policy (Reactive/Closed-Loop Strategy)



- A policy Π is a complete mapping from states to actions

Reactive Agent Algorithm

Repeat:

- s ← sensed state
- If s is terminal then exit
- a ← Π(s)
- Perform a

Approaches

- learn policy directly– function mapping from states to actions
- learn utility values for states, the value function

Value Function

- An agent knows what state it is in and it has a number of actions it can perform in each state.
- Initially it doesn't know the value of any of the states.
- If the outcome of performing an action at a state is deterministic then the agent can update the utility value U() of a state whenever it makes a transition from one state to another (by taking what it believes to be the best possible action and thus maximizing): U(oldstate) = reward + U(newstate)
- The agent learns the utility values of states as it works its way through the state space.

Exploration

- The agent may occasionally choose to explore suboptimal moves in the hopes of finding better outcomes. Only by visiting all the states frequently enough can we guarantee learning the true values of all the states.
- A discount factor is often introduced to prevent utility values from diverging and to promote the use of shorter (more efficient) sequences of actions to attain rewards. The update equation using a discount factor gamma is:
- U(oldstate) = reward + gamma * U(newstate)
- Normally gamma is set between 0 and 1.

Q-Learning

- augments value iteration by maintaining a utility value Q(s,a) for every action at every state.
- utility of a state U(s) or Q(s) is simply the maximum Q value over all the possible actions at that state.

Q-Learning

- foreach state s
  foreach action a Q(s,a)=0
  s=currentstate
  do forever
  a = select an action
  do action a
  r = reward from doing a
  t = resulting state from doing a
  Q(s,a) += alpha * (r + gamma * (Q(t)-Q(s,a))
  s = t
- Notice that a learning coefficient, alpha, has been introduced into the update equation. Normally alpha is set to a small positive constant less than 1.

Selecting an Action

- simply choose action with highest expected utility?
- problem: action has two effects
  - gains reward on current sequence
  - informa stuck in a rut used in learning for future sequences
- trade-off immediate good for long-term well-being

jumping off a cliff just because you've never done it before…

Exploration policy

- wacky approach: act randomly in hopes of eventually exploring entire environment
- greedy approach: act to maximize utility using current estimate
- need to find some balance: act more wacky when agent has little idea of environment and more greedy when the model is close to correct
- example: one-armed bandits…

RL Summary

- active area of research
- both in OR and AI
- several more sophisticated algorithms that we have not discussed
- applicable to game-playing, robot controllers, others

Genetic Algorithms

- use evolution analogy to search for 'successful' individuals
- individuals may be policy (like RL), a computer program (in this case called genetic programming), a decision tree, a neural net, etc.
- success is measured in terms of **fitness** function.
- start with a pool of individuals and use selection and reproduction to evolve the pool

Basic Algorithm

1. **[Start]** Generate random population of *n* individuals (suitable solutions for the problem)
2. **[Fitness]** Evaluate the fitness *f(x)* of each individual
3. **[New population]** Create a new population by repeating following steps until the new population is complete
4. **[Selection]** Select two parents from a population according to their fitness (the better fitness, the bigger chance to be selected)
5. **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
6. **[Mutation]** With a mutation probability mutate new offspring at each locus (position in chromosome).
7. **[Accepting]** Place new offspring in a new population
8. **[Loop]** Go to step **2**

GA Issues

- what is fitness function?
- how is an individual represented?
- how are individuals selected?
- how do individuals reproduce?

GA summary

- easy to apply to a wide range of problems
- results good on some problems and not so hot on others

> "neural networks are the second best way of doing just about anything… and genetic algorithms are the third"