CSCI 4150 Introduction to Artificial Intelligence, Fall 1999
Assignment 8 details

1 Bayes rule, Bayesian belief networks, and Bayesian classifiers

(a) In Athens (so the problem goes) 90% of the taxis are green; the rest are blue. An accident involving a taxicab occurs; assume that the accident rate for green taxis is the same as for blue taxis. An eyewitness says that the taxi was blue. Extensive testing has shown that under the lighting conditions at the accident scene, discrimination between green and blue is 75%, i.e. if the taxi was blue, then there is a 75% chance that an eyewitness says it was blue and a 25% chance that an eyewitness says it was green (and vice versa if the taxi was green.) What is the probability that the taxi was blue given the eyewitness' statement?

(b) This problem illustrates the principle behind the Bayes optimal classifier.

Suppose that there are just 3 hypotheses, h_1, h_2, and h_3, for a classification problem. The probabilities that these hypotheses are true given the training data are:

$$\begin{aligned} P(h_1|D) &= 0.2 \\ P(h_2|D) &= 0.3 \\ P(h_3|D) &= 0.5 \end{aligned}$$

We want to classify a new example. This example has a negative or positive classification given a hypothesis with the following probabilities:

$$\begin{array}{ll} P(\ominus|h_1) = 0.1 & P(\oplus|h_1) = 0.9 \\ P(\ominus|h_2) = 0.4 & P(\oplus|h_2) = 0.6 \\ P(\ominus|h_3) = 0.7 & P(\oplus|h_3) = 0.3 \end{array}$$

What is the most likely classification of this new example? Show your work.

(c) This problem illustrates the Bayes naive classifier.

Suppose we have only three attributes for the restaurant example in our text (in Chapter 18): Patrons, Hungry, and Friday. These have the following conditional probabilities (given the value of the goal attribute, WillWait).

P(Patrons=none | Wait=yes)	=	0	P(Patrons=none | Wait=no)	=	1
P(Patrons=some | Wait=yes)	=	1	P(Patrons=some | Wait=no)	=	0
P(Patrons=full | Wait=yes)	=	1/3	P(Patrons=full | Wait=no)	=	2/3
P(Hungry=yes | Wait=yes)	=	5/7	P(Hungry=yes | Wait=no)	=	2/7
P(Hungry=no | Wait=yes)	=	1/5	P(Hungry=no | Wait=no)	=	4/5
P(Friday=yes | Wait=yes)	=	2/5	P(Friday=yes | Wait=no)	=	3/5
P(Friday=no | Wait=yes)	=	4/7	P(Friday=no | Wait=no)	=	3/7

From the training data, we also know that P(Wait=yes) = 0.5, and P(Wait=no) = 0.5.

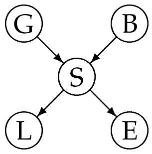
Now we want to classify a new example whose attribute values are: Patrons=full, Hungry=yes, and Friday=no. What is the most likely classification? Show your work.

(d) This is an OPTIONAL problem covering inference in Bayesian belief networks. It requires and illustrates the concept of conditional independence used in many forms of Bayesian inference, including the Bayesian naive classifier.

Bob isn't very good about maintaining his car. Sometimes he forgets to get gas, so his car won't start in the morning when he needs to go to work. Sometimes, he leaves his lights or the radio on overnight, so that in the morning, the battery is dead and his car won't start. However, when one of these things happens, Bob may be able to borrow some gas or get a jump start from a neighbor.

If Bob's car doesn't start, then he will probably be late for work, and his parking place will be empty. Of course there's always the chance that he'll get a lift from someone else or borrow his friend's car.

We can represent the relationship between these events with the following Bayesian belief network:



Where G means "out of gas", B means "battery dead", S means "car won't start", L means "late for work", and E means "parking spot empty". Assume the following conditional probabilities:

G	B	P(S)
T	T	0.9
T	F	0.7
F	T	0.8
F	F	0.1

S	P(L)
T	0.7
F	0.2

S	P(E)
T	0.8
F	0.1

The probability that Bob's car is out of gas is P(G) = 0.2, and the probability that the batter is dead is P(B) = 0.1.

(a) What is the value of P(L | B)? Show your work.

(b) What is the value of P(G | E)? Show your work.

2 Q-learning

[The introduction to this problem is given in the original Assignment 8 handout.]

Grid worlds

The worlds used for this problem are similar to the example in Chapter 20. I have defined two worlds for you to test your program on:

7	8	9	10
4	X	5	6
0	1	2	3

- Name: `threefour`
- Terminal states:
  - 10, reward = 1
  - 6, reward = -1
- Reward for nonterminal states: -0.04
- Transition probabilities (relative to action)
  - straight = 0.7
  - left = 0.15
  - right = 0.15

10	11	12	13
8	X	9	X
4	5	6	7
0	1	2	3

- Name: `fourfour`
- Terminal states:
  - 9, reward = 1
  - 5, reward = -1
  - 7, reward = -1
- Reward for nonterminal states: -0.01
- Transition probabilities (relative to action)
  - straight = 0.7
  - left = 0.15
  - right = 0.15

Each epoch begins in cell 0. Your `q-learner` function will be called repeatedly and must return an action (one of `north`, `south`, `east`, and `west`). This action will be simulated, and there will be a new state (i.e. cell). Your `q-learner` function will be called repeatedly until a terminal state is reached.

Running learning trials

Once you have written your `q-learner` function, you can use it to learn Q values for a "grid world" using the following functions:

- `(run-learner world epochs)`

  This function will set up a world (initialize the simulator and then create tables for N(a,i) and Q(a,i)) and then run the given number of epochs. This function will increment the values in the N(a,i) table and call your `q-learner` function to update the Q values and to choose an action. At the end, this function will print the Q table and print the policy determined by your Q table.

  The amount of debugging information printed during this and other function below is controlled by the variable `print-debug-level` which can be an integer from 0 to 4 (inclusive). See the source code for details.

- `(run-epochs epochs)`

  This command can be used *after* you have run the `run-learner` command to run additional epochs without resetting the N and Q tables.

By the way, the Q and N tables are global variables (even though they will be passed to your function). This is useful if you want to examine them after running a few epochs.

Action-state tables

I have implemented an abstract data type for you to use as an action state table. The `run-learner` function takes care of setting up the tables and initializing them. Your `q-learner` function will only need to get and set values in the table, but all the commands are described here.

These tables are objects. In reality, they are functions masquerading as data (or perhaps vice versa!) In order to use them, you make a function call and give them arguments. I use the `q` table in the specifications below.

- `(Q 'get <action> <state>)`

  Returns the value of the table for the given action and state. The action is one of: `north`, `south`, `east`, and `west`. The state is an integer which refers to the cell number, i.e. from 0 to $n-1$ in an n cell world.

- `(Q 'set <action> <state> <value>)`

  Sets the value of the table for the given action and state to the given value.

- `(Q 'print)`

  Prints the table.

You should not need the following commands, but I'll list them here for completeness:

- `(N 'inc <action> <state>)`

  Adds 1 to the table entry for the given action and state. Used in the `run-learner` function to update the frequency table N

- `(Q 'reset)`

  Sets all the values in the table to 0.

- `(Q 'no-states)`

  Returns the number of states (i.e. the number of cells in the grid-world).

- `(make-action-table no-states)`

  Creates a new action/state table of the given size with all elements initialized to 0.0.

Notes on the `q-learner` function

You are writing the function:

```
(q-learner state prev-state prev-action reward q n)
```

This function should perform the following algorithm:

- if there is a previous state, update the appropriate Q value (using the temporal differencing update rule for Q-learning)
- pick and return an action

This is simpler than the algorithm given in Figure 20.12 in our text because you do not need to keep static variables (all the information you need is passed to your function through arguments) and because the `run-learner` function takes care of updating the N table.

A few notes on writing this function:

- Note that `state` is the current state, and `reward` is the reward for the previous state.
- At the beginning of each epoch, there will be no previous state. For this call, the value of `prev-state` will be `'()`.
- You should use only floating point numbers! Mixing integers and floating point numbers doesn't always work. For example:

  ```
  (equal? 1 1.)
  ;Value: ()

  (= 1 1.)
  ;Value: #t
  ```

- I suggest starting with the $f(u, n)$ function that the text suggests (R^+ for $n < N_c$, u otherwise). For the α function, you can start with $\alpha(n) = 1/(1 + n)$. See the text for more discussion on these functions, and see Figure 20.13 for some suggested starting values.
- You will have to run at least 50 epochs to get some barely reasonable Q values. In order to get more stable values, you'll have to run at least 100, or perhaps even 1000 epochs for the values to stabilize. You can look at the N table to get an idea of what your Q-learner is doing. Note that once the exploration function no longer alters the Q values, your Q-learner may never even try to reach certain parts of the grid-world.

  Try different R^+ values to see their effect on the policy. In the `fourfour` world, can you get the policy to approach the +1 reward from the back (between the two obstacles) in order to avoid the -1 terminal states?