Issued: Wed., April 8, 2020
Due: Wed., Apr. 22, 2020 (11:59pm to Canvas)

PROJECT #5: THE MDP & INTRO TO MACHINE LEARNING

Complete all parts of this project independently. Do not use any solutions found online or written by other students. If you are struggling please go to office hours, post to Piazza, and/or send us email. Please submit a single PDF file `project5.pdf` to Canvas for the handwritten problems you submit. Please also submit a single code file `project5.tar.gz` to Canvas containing your Python or C++ code solutions.

1. **Markov Chains:** (20 points)
   Suppose a person could be in one of four economic states: billionaire (B), wealthy (W), middle-class (M), or poor (P). It is a time of economic volatility, resulting in the following single-step dynamics: A billionaire has a 40% chance of becoming only wealthy (W), a 10% chance of becoming middle-class, and a 5% chance of becoming poor. A wealthy person has a 5% chance of becoming a billionaire, a 20% chance of becoming middle class, and a 15% chance of becoming poor. A person of average (middle-class) status has a 2% chance of becoming a billionaire (due to hard work and good luck!), a 10% chance of becoming wealthy, and a 25% chance of becoming poor. A person of modest means (poor) has no chance of immediately becoming a billionaire (in this homework problem!), a 10% chance of becoming wealthy, and a 30% chance of becoming middle-class.
   a. Draw the Markov Chain directed graph corresponding to the above uncertain transition dynamics. Label all states {B, W, M, P} and specify all transition probabilities along the edges.
   b. Specify the transition probability matrix M such that $X_{i+1} = X_i M$ given a mapping $\mathbf{X} = \{X_1, X_2, X_3, X_4\} = \{B, W, M, P\}$.
   c. What is the probability a person that is initially middle-class will be wealthy after four time steps?
   d. What is the probability a person that is initially wealthy will be middle-class after ten time steps?
   e. Specify the steady state probability vector $\mathbf{X}$ for this Markov Chain if such a solution exists.

2. **Decision Tree Classification:** (20 points)

Suppose we observe a set of baseball games played between two top-ranked teams, Team A and Team B. We observe the conditions of the game (Time, Game Type, Weather) as well as each game outcome (A=Team A wins; B=Team B wins), as shown below.

TIME	GAME TYPE	WEATHER	OUTCOME
Morning	Regular Season	Rainy	A
Night	Regular Season	Rainy	A
Afternoon	Regular Season	Windy	B
Night	Playoffs	Cloudy	A
Night	Playoffs	Clear	A
Morning	Regular Season	Rainy	A
Night	Playoffs	Clear	A
Night	Preseason	Windy	B
Afternoon	Preseason	Cloudy	A
Night	Regular Season	Clear	B
Night	Playoffs	Rainy	A
Night	Playoffs	Cloudy	A
Night	Playoffs	Clear	B
Afternoon	Preseason	Snowy	A
Night	Playoffs	Cloudy	A
Night	Regular Season	Clear	B

a. Manually create a decision tree using attribute ordering Time → Game Type → Weather. Label each leaf node; terminate a branch when all examples have been classified correctly (if possible).
b. Now create a decision tree using attribute ordering Weather → Time → Game Type. Label each leaf node; terminate a branch when all examples have been classified correctly (if possible).
c. What is the most likely outcome of a Playoff game on a clear night, given your decision tree(s)?

3. **Markov Decision Process -- Coding** (30 points)

Implement Value Iteration (R&N Figure 17.4; shown below). Read the MDP specification and error threshold from file `mdpinput.txt`; write the computed policy in file `policy.txt`. In addition to returning state values (utilities U), also return the optimal policy matching actions to states. Below are sample input and output files from the "Mars Rover on a Hill" example, for which a solution is also provided below. Please assume all comment lines (beginning %) will and should appear in graded test case input and output files. Also make sure your reward function R(s,a) is included inside the argmax(a) expression since we allow our reward function to be a function of selected action as well as state.

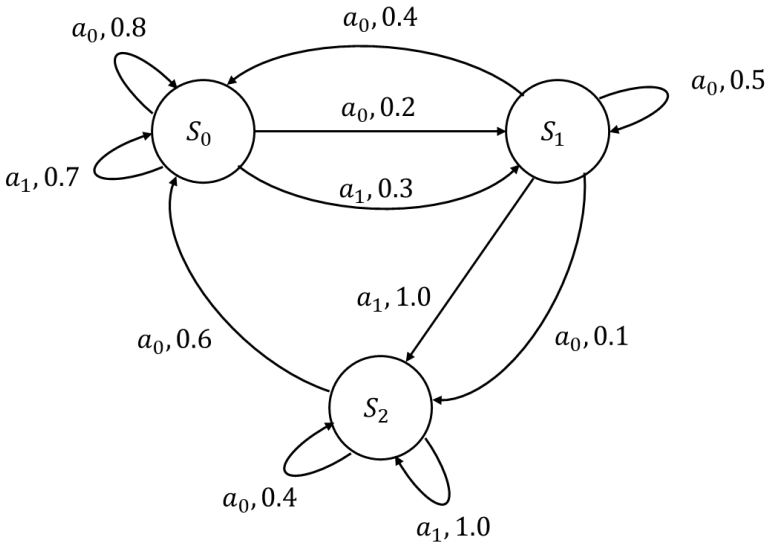
function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'$; $\delta \leftarrow 0$
    for each state s **in** S **do**
      $U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

Mars Rover Problem: A Mars rover depends on its solar panels for energy. Its goal is to collect as much energy as possible. It is close to a small hill. The rover collects the most energy when it is at the top of the hill, but it has a tendency to roll off the hill, and it takes energy to get back up the hill. Specifically, the rover can be in one of three states: on top of the hill (S0), rolling down the hill (S1), or at the bottom of the hill (S2). There are two actions that it can take in each state: drive (a0) or don't drive (a1). Driving always costs 1 unit of energy. At the top of the hill, the rover collects 3 units of energy; in the other two states, it collects 1 unit of energy. For example, if the rover is at the top of the hill and is driving (to stay on top of the hill), its reward is 3 − 1 = 2. If the rover is at the top of the hill and drives, then it is still at the top of the hill in the next period with probability 0.8, and rolling down in the next period with probability 0.2. If it is at the top of the hill and does not drive, these probabilities are 0.7 and 0.3, respectively. If it is rolling down and drives, then with probability 0.4 it is at the top of the hill in the next period, with probability 0.5 it is still rolling down in the next period, and with probability 0.1 it is at the bottom in the next period. If it is rolling down and does not drive, then with probability 1 it is at the bottom in the next period. Finally, if it is at the bottom of the hill and drives, then in the next period it is at the top of the hill with probability 0.6 and at the bottom with probability 0.4. If it does not drive, then with probability 1 it is at the bottom in the next period.

Mars Rover Problem MDP graph:



Example input file `mdpinput.txt`:

```
% States (comma-separated list)
S0, S1, S2
% Actions (comma-separated list)
a0, a1
% Transition model: One transition matrix per action.
% State, action orderings will match the above lists.
% Action 1:
0.8, 0.2, 0.0
0.4, 0.5, 0.1
0.6, 0.0, 0.4
% Action 2:
0.7, 0.3, 0.0
0.0, 0.0, 1.0
0.0, 0.0, 1.0
% Rewards (State, Action, R(s,a))
S0, a0, 2
S0, a1, 3
S1, a0, 0
S1, a1, 1
S2, a0, 0
S2, a1, 1
% Discount factor (gamma)
0.8
% Epsilon (tolerance)
0.001
```

Example output file `policy.txt`:

```
% Format:  State: Action (Value)
S0: a1 (10.64)
S1: a1 (7.01)
S2: a0 (7.51)
```

4. **Decision Tree Generation -- Coding** (30 points)

Implement the `DECISION-TREE-LEARNING(examples,attributes,parent_examples)` function shown in R&N Figure 18.5, reproduced below, within a program or script that manages a single test case and print the decision tree to a file. Read training examples from input file `examples.txt` and write your output to file `dtree.txt`. Below (next page) are sample files for the "restaurant patron" example. Comment lines beginning % will appear exactly as written. You can assume there will be no spaces or tabs in each attribute or value name.

function DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$) **returns** a tree

if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
  $A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
  $tree \leftarrow$ a new decision tree with root test A
  for each value v_k of A **do**
    $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
    $subtree \leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - A$, $examples$)
    add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
  return $tree$

Figure 18.5 The decision-tree learning algorithm. The function IMPORTANCE is described in Section 18.3.4. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

Example `examples.txt` file (from R&N Figure 18.3):

```
% Input Attributes (Format: Attribute name: value1, value2, …)
Alt: Yes, No
Bar: Yes, No
Fri: Yes, No
Hun: Yes, No
Pat: None, Some, Full
Price: $, $$, $$$
Rain: Yes, No
Res: Yes, No
Type: French, Thai, Burger, Italian
Est: 0-10, 10-30, 30-60, >60
% Decision values (comma-separated list)
Yes, No
% Example instances
% Attribute ordering will be the same as above; decision is last
Yes, No, No, Yes, Some, $$$, No, Yes, French, 0-10, Yes
Yes, No, No, Yes, Full, $, No, No, Thai, 30-60, No
No, Yes, No, No, Some, $, No, No, Burger, 0-10, Yes
Yes, No, Yes, Yes, Full, $, Yes, No, Thai, 10-30, Yes
Yes, No, Yes, No, Full, $$$, No, Yes, French, >60, No
No, Yes, No, Yes, Some, $$, Yes, Yes, Italian, 0-10, Yes
No, Yes, No, No, None, $, Yes, No, Burger, 0-10, No
No, No, No, Yes, Some, $$, Yes, Yes, Thai, 0-10, Yes
No, Yes, Yes, No, Full, $, Yes, No, Burger, >60, No
Yes, Yes, Yes, Yes, Full, $$$, No, Yes, Italian, 10-30, No
No, No, No, No, None, $, No, No, Thai, 0-10, No
Yes, Yes, Yes, Yes, Full, $, No, No, Burger, 30-60, Yes
```

An example `dtree.txt` file (consistent with R&N Figure 18.6):

```
% Format: decision? value, next node (leaf value or next decision?)
% Use question mark and comma markers as indicated below.
Pat? None, No
Pat? Some, Yes
Pat? Full, Hun?
Hun? No, No
Hun? Yes, Type?
Type? French, Yes
Type? Italian, No
Type? Thai, Fri?
Type? Burger, Yes
Fri? No, No
Fri? Yes, Yes
```