Assignment 4

CSE 343/543: Machine Learning **Due: 11:59 PM, Nov 17, 2017**

Note: Please complete programming as well as theory component

Submission: For question on reinforcement learning, you just have to submit 2 files: valueIterationAgents.py and analysis.py. For the question on K-Means clustering, you will submit all the files of your code.

Programming Questions

Reinforcement Learning

Note: The code for this question only runs on python 2.7

Attribution Information: The code for the grid world and pacman projects have been developed at UC Berkeley (http://ai.berkeley.edu/). We are using the above code for educational purposes only without claiming any kind of ownership for any part of the 3 questions on reinforcement learning.

For this part of the assignment, you will implement a value iteration agent which will learn to solve a known MDP.

You will edit the following files in the project folder:

1. valueIterationAgents.py
2. analysis.py

You will go through the classes and functions of the following files to know more about the assignment but do not edit them.

1. mdp.py
2. learningAgents.py
3. util.py
4. gridworld.py

You can ignore any other file not mentioned in the above two lists. Do not make changes to any files apart from the ones in which you are required to add code.

Files to submit: valueIterationAgents.py, analysis.py

Evaluation: Your code will be auto-graded for technical correctness. **Please *do not* change the names of any provided functions or classes within the code**, or you will wreak havoc on the autograder.

MDPs

To get started, run Gridworld in manual control mode, which uses the arrow keys:

```
python gridworld.py -m
```

You will see the two-exit layout from class. The blue dot is the agent. Note that when you press *up*, the agent only actually moves north 80% of the time. Such is the life of a Gridworld agent!

You can control many aspects of the simulation. A full list of options is available by running:

```
python gridworld.py -h
```

The default agent moves randomly

```
python gridworld.py -g MazeGrid
```

You should see the random agent bounce around the grid until it happens upon an exit. Not the finest hour for an AI agent.

Note: The Gridworld MDP is such that you first must enter a pre-terminal state (the double boxes shown in the GUI) and then take the special 'exit' action before the episode actually ends (in the true terminal state called TERMINAL_STATE, which is not shown in the GUI). If you run an episode manually, your total return may be less than you expected, due to the discount rate (-d to change; 0.9 by default).

Look at the console output that accompanies the graphical output (or use -t for all text). You will be told about each transition the agent experiences (to turn this off, use -q).

Question 1)

Write a value iteration agent in ValueIterationAgent, which has been partially specified for you in valueIterationAgents.py. Your value iteration agent is an offline planner, not a reinforcement learning agent, and so the relevant training option is the number of iterations of value iteration it should run (option -i) in its initial planning phase. ValueIterationAgent takes an MDP on construction and runs value iteration for the specified number of iterations before the constructor returns.

Value iteration computes k-step estimates of the optimal values, V_k. In addition to running value iteration, implement the following methods for ValueIterationAgent using V_k.

- computeActionFromValues(state) computes the best action according to the value function given by self.values.
- computeQValueFromValues(state, action) returns the Q-value of the (state, action) pair given by the value function given by self.values.

These quantities are all displayed in the GUI: values are numbers in squares, Q-values are numbers in square quarters, and policies are arrows out from each square.

Note: A policy synthesized from values of depth k (which reflect the next k rewards) will actually reflect the next k+1 rewards (i.e. you return $\pi_{k+1}\pi_{k+1}$). Similarly, the Q-values will also reflect one more reward than the values (i.e. you return Q_{k+1}).

You should return the synthesized policy π_{k+1}.

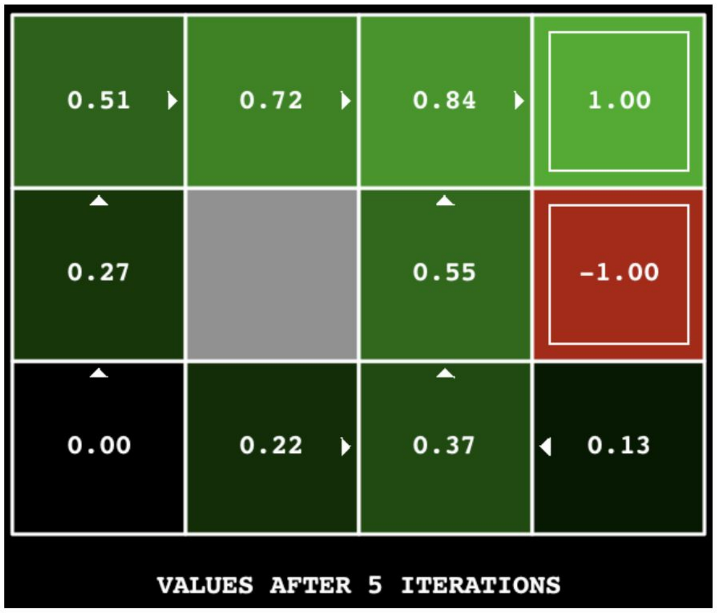
Hint: Use the util.Counter class in util.py, which is a dictionary with a default value of zero. Methods such as totalCount should simplify your code. However, be careful with argMax: the actual argmax you want may be a key not in the counter!

Note: Make sure to handle the case when a state has no available actions in an MDP (think about what this means for future rewards).

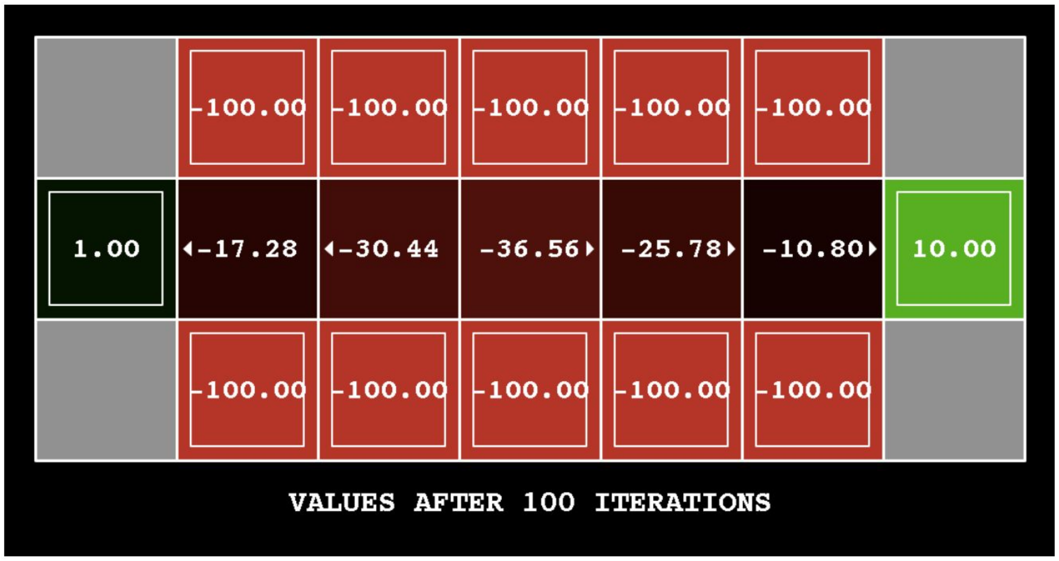
To test your implementation:

On the default BookGrid, running value iteration for 5 iterations should give you this output:

```
python gridworld.py -a value -i 5
```



Question 2)



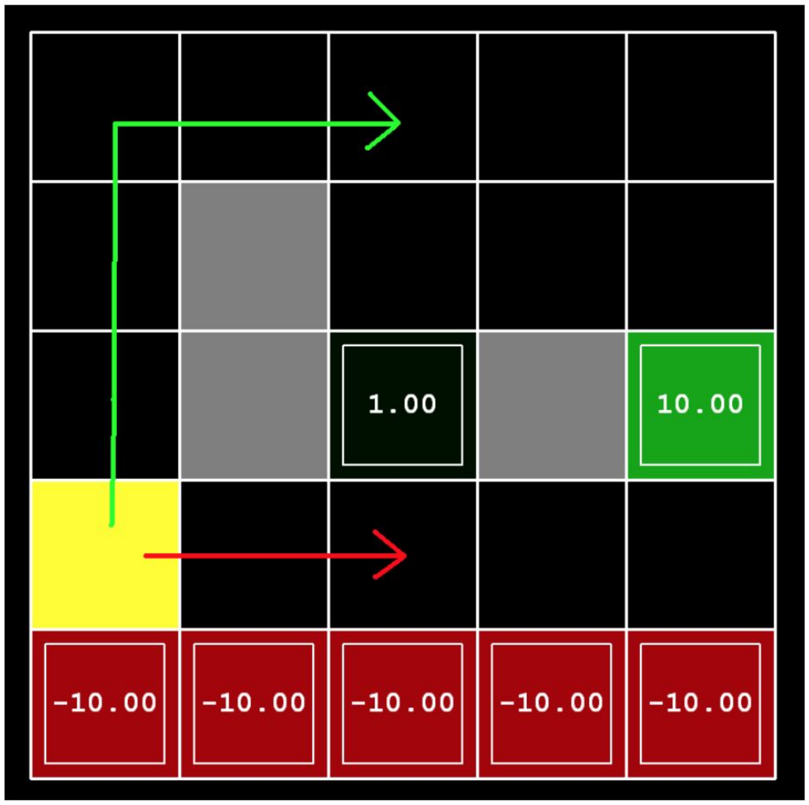
BridgeGrid is a grid world map with the a low-reward terminal state and a high-reward terminal state separated by a narrow "bridge", on either side of which is a chasm of high negative reward. The agent starts near the low-reward state. With the default discount of 0.9 and the default noise of 0.2, the optimal policy does not cross the bridge. Change only ONE of the discount and noise parameters so that the optimal policy causes the agent to attempt to cross the bridge. Put your answer in question2() of analysis.py. (Noise refers to how often an agent ends up in an unintended successor state when they perform an action.) The default corresponds to:

```
python gridworld.py -a value -i 100 -g BridgeGrid --discount 0.9 --noise 0.2
```

Grading: We will check that you only changed one of the given parameters, and that with this change, a correct value iteration agent should cross the bridge.

Question 3)

Consider the DiscountGrid layout, shown below. This grid has two terminal states with positive payoff (in the middle row), a close exit with payoff +1 and a distant exit with payoff +10. The bottom row of the grid consists of terminal states with negative payoff (shown in red); each state in this "cliff" region has payoff -10. The starting state is the yellow square. We distinguish between two types of paths: (1) paths that "risk the cliff" and travel near the bottom row of the grid; these paths are shorter but risk earning a large negative payoff, and are represented by the red arrow in the figure below. (2) paths that "avoid the cliff" and travel along the top edge of the grid. These paths are longer but are less likely to incur huge negative payoffs. These paths are represented by the green arrow in the figure below.



In this question, you will choose settings of the discount, noise, and living reward parameters for this MDP to produce optimal policies of several different types. Your setting of the parameter values for each part should have the property that, if your agent followed its optimal policy without being subject to any noise, it would exhibit the given behavior. If a particular behavior is not achieved for any setting of the parameters, assert that the policy is impossible by returning the string 'NOT POSSIBLE'.

Here are the optimal policy types you should attempt to produce:

1. Prefer the close exit (+1), risking the cliff (-10)
2. Prefer the close exit (+1), but avoiding the cliff (-10)
3. Prefer the distant exit (+10), risking the cliff (-10)
4. Prefer the distant exit (+10), avoiding the cliff (-10)
5. Avoid both exits and the cliff (so an episode should never terminate)

question3a() through question3e() should each return a 3-item tuple of (discount, noise, living reward) in analysis.py.

Note: You can check your policies in the GUI. For example, using a correct answer to 3(a), the arrow in (0,1) should point east, the arrow in (1,1) should also point east, and the arrow in (2,1) should point north.

Note: On some machines you may not see an arrow. In this case, press a button on the keyboard to switch to qValue display, and mentally calculate the policy by taking the arg max of the available qValues for each state.

Grading: We will check that the desired policy is returned in each case.

K-Means

Download the following datasets from the UCI Machine Learning Repository:

1. Seeds
2. Vertebral Column
3. Image Segmentation
4. Iris

Each download will contain a folder with (possibly) many files. The relevant files are as follows:

1. seeds_dataset.txt: the last column is the label, all other columns are numeric features
2. column_3C.dat: the last column is the label, all other columns are numeric features
3. segmentation.data: the first column is the label, all other columns are numeric features
4. iris.data: the last column is the label, all other columns are numeric features

You should explore the datasets a little bit, read their description, and perform necessary cleaning/preprocessing if required.

Your main task is to write an implementation of the k-means algorithm and use it to cluster the above datasets. Your algorithm should be in the form of a function that should be generic enough to take an unlabelled dataset (possibly as a matrix) as input and return labels (possibly as a vector) for each datapoint as output.

Using the above implementation of k-means, generate the following plots for each dataset (while running k-means, pass the true (known) value of k to your function):

1. 2D scatter-plot after applying t-SNE, color coded with the ground truth labels (you don't need k-means for this. You should do this before you write your k-means implementation to visualize and understand the data).
2. 2D scatter-plot after applying t-SNE, color coded with the predicted labels
3. Objective Function vs Iteration Number

Include these plots in your report with your analysis and inferences. This is the qualitative/visual evaluation of your k-means implementation.

Following this, you need to perform a quantitative evaluation. Using your predictions, calculate the following for each dataset and populate the table given below. You will have to run k-means multiple times with different values of k. Each entry should be averaged over 5 independent runs of k-means.

1. Adjusted Rand Index (ARI)
2. Normalized Mutual Information (NMI)
3. Adjusted Mutual Information (AMI)

Data	K = 2			K = True Value			K = 12		
	ARI	NMI	AMI	ARI	NMI	AMI	ARI	NMI	AMI
Iris									
Segmentation									
Seeds									
Vertebral									

You should read up about these metrics. An implementation is available from sklearn. Include this table along with your analysis inferences in your report.

Specifically, your report should contain at least the following (in addition to the required graphs and plots):

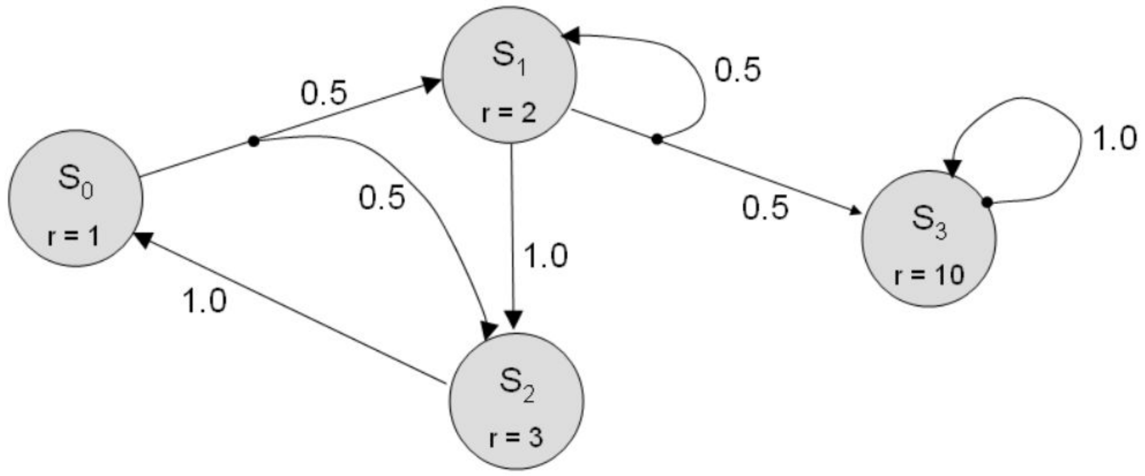
- How good was the clustering for each dataset based on qualitative/visual metrics (tsne, graphs) (this description should go with the tsne plots)
- How good was the clustering for each dataset based on quantitative metrics (ARI, NMI, AMI) (this description should go with the table above)
- Consistency between qualitative and quantitative analysis (at the end)

For your k-means implementation, you can use only pure python and numpy. For everything else, you can use sklearn in addition to python and numpy.

You are advised to make your code modular and easy to read. Separate the stages of data preparation, clustering, plot generation and numeric evaluation from each other and use a driver script to call these APIs and perform the required tasks. This will make it easier for you to debug and for us to evaluate.

Theory Questions

1. The figure below shows an MDP. The numbers next to the arrows are probabilities of outcomes and 'r' denotes the reward at each step. State S_1 has 2 actions, while all the other states have only 1 action.



   a. Write down the numerical values $V(S_n)$ of each state for the first 3 iterations of Value Iteration. Initial values are:

      $V(S_0) = 0$; $V(S_1) = 0$; $V(S_2) = 0$; $V(S_3) = 0$

   b. What is the optimal policy at state S_1? Optimal policy is defined action indicated by the maximum Q value at any of all possible actions at any state.
   c. Write true or false against the following statements. Explain your reasons in each case.
      i. Any MDP with N states converges after N value iterations for $\gamma = 0.5$
      ii. Any MDP converges after the first value iteration for $\gamma = 1$
      iii. Any MDP converges after the first value iteration for $\gamma = 0$
      iv. An acyclic MDP with N states converges after N value iterations for any $0 < \gamma < 1$
      v. An MDP, without absorbing goal states, with N states and no stochastic actions converges after N value iterations for any $0 < \gamma < 1$

2. Suppose you want to transmit an N x N image where each pixel is an {R, G, B} value stored with 8 bits of precision. How many bits would it take to transmit this image? Now suppose you first run k-means on the image data (number of clusters = K) and use the results to compress the image and then transmit it. How many bits do you need now? What is the compression ratio achieved? (Compression Ratio = Uncompressed Size/Compressed Size).