SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

50.007 Machine Learning, Summer 2023
Homework 2

Due 25 June 2023, 11:59 pm

1. Clustering [15 points]

In clustering, Euclidean distance is not the only way to measure the distance between two points/vectors. l_p norms is a family of distance measures that are parameterized by $p \geq 1$. The l_p norm of a vector is:

$$\|x\|_p = \left(\sum_j |x_j|^p\right)^{\frac{1}{p}}.$$

Euclidean distance is the l_2 norm of the vector difference between two points, i.e.,

$$\|x - y\|_2 = \left(\sum_j |x_j - y_j|^2\right)^{\frac{1}{2}}.$$

The Manhattan distance is the l_1 norm of the vector difference between two points, i.e.,

$$\|x - y\|_1 = \sum_j |x_j - y_j|.$$

The l_∞ distance is the maximum absolute element in the vector difference between two points, i.e.,

$$\|x - y\|_\infty = \max_j |x_j - y_j|.$$

Consider a set of points $X = (0.6, 0.8), (0.8, 0.6), (-0.8, 0.6)$. Compute the value of z that minimizes $\sum_{x \in X} d(x, z)$ when $d(x, z)$ is defined as follows respectively:

(a) the Euclidean distance between x and z,

(b) the squared Euclidean distance between x and z,

(c) the Manhattan distance between x and z.

2. k-Means [30 points]

Consider the image "hw2_img.jpg". The image has 187 rows and 250 columns which can be represented as a 3-d array with one axis for the height, width and colour channels. We can rearrange this 3-d array into a 2-d array of shape (46750, 3), where each row represents one pixel in the image and each column represents one of the three colour channels. In this assignment, we will explore clustering methods, applying them in particular to the problem of dividing the pixels of the image into a small number of similar clusters. Consider the k-means clustering algorithm, as described in class. In particular, consider a version in which the inputs to the algorithm are:

- The set of data to be clustered. (i.e., the vectors $x^{(1)}, x^{(2)}, x^{(3)}, ...$)
- The desired number of clusters, k.
- Initial centroids for the k clusters.

Then the algorithm proceeds by alternating: (1) assigning each instance to the class with the nearest centroid, and (2) recomputing the centroids of each class—until the assignments and centroids stop changing. Please use squared Euclidean distance (Lecture 5, Eq. 2) as the metric for clustering.

There are many implementations of K-means publicly available. However, please implement K-Means on your own. Then, use your implementation to cluster the data in the file mentioned above ("hw2_img.jpg"), using k = 1 to k = 8. You may choose your own initialization points for each case. (Hint: Try to make the different centroids to be far from each other when performing initialization.)

You may use any image IO library to load and convert the image into an integer array. The Pillow package has good compatibility with NumPy so you may choose to use that.

Turn in your code, as well as a report on each of the following for each case:

(a) How many clusters there are in the end. (A cluster can "disappear" in one iteration of the algorithm if no vectors are closest to its centroid.)

(b) The final centroids of each cluster.

(c) The number of pixels associated to each cluster.

(d) Visualize your result by replacing each pixel with the centroid to which it is closest, and displaying the resulting image.

Plot the cost of clustering as a function of k. Specifically, for k = 1 to k = 8. Use the elbow heuristics for selecting the best k value.

3. Markov Decision Process [32 points]

Consider the Markov decision process (MDP) associated with a simplified version of the robot that we showed during class that plays the guessing game with us. It has 4 states $\{$*Uncertain*, *Certain*, *Lose*, *Win*$\}$, with *Win* being the final state. The following table summarizes the actions that the robot can take at each state, and the transition probabilities from one state to another after taking a certain action. For example, when the robot is at the *Certain* state (it is certain about the answer), there are two possible actions to take: ask (A) and guess (G). If it takes the action G, then there is a probability 0.7 that the robot will arrive at the *Win* state (and wins the game).

s	a	s'	$T(s,a,s')$
Uncertain	A	*Uncertain*	0.9
Uncertain	A	*Certain*	0.1
Uncertain	G	*Lose*	0.9
Uncertain	G	*Win*	0.1
Certain	A	*Certain*	1.0
Certain	G	*Lose*	0.3
Certain	G	*Win*	0.7
Lose	A	*Uncertain*	0.8
Lose	A	*Certain*	0.2
Lose	G	*Lose*	0.8
Lose	G	*Win*	0.2
Win	A	*Win*	1.0
Win	G	*Win*	1.0

The reward function $R(s,a,s') = R(s')$ is defined as:

s'	*Uncertain*	*Certain*	*Lose*	*Win*
$R(s')$	0.0	1.0	-2.0	2.0

The discount factor is $\gamma = 0.5$.

(a) Suppose we initialize $Q_0^*(s,a) = 0$ for all $s \in S$ and $a \in \{\text{A,G}\}$. Evaluate the Q-values $Q_1^*(s,a)$ after exactly one iteration of the Q-Value Iteration Algorithm (assume we perform synchronized updates. In other words, we always use Q_0 values when we calculate Q_1 values). Write your answers in the table below.

	$s =$*Uncertain*	$s =$*Certain*	$s =$*Lose*	$s =$*Win*
A				
G				

(b) What is the policy that we would derive from $Q_1^*(s,a)$? Answer by filling in the action that should be taken at each state in the table below (in case of draw, the action G is preferred).

	$s =$*Uncertain*	$s =$*Certain*	$s =$*Lose*	$s =$*Win*

(c) What are the values $V_1^*(s)$ corresponding to $Q_1^*(s,a)$?

	$s =$*Uncertain*	$s =$*Certain*	$s =$*Lose*	$s =$*Win*

(d) Evaluate the Q-values $Q_2^*(s,a)$ after exactly two iterations of the Q-Value Iteration Algorithm. Write your answers in the table below.

	$s =$*Uncertain*	$s =$*Certain*	$s =$*Lose*	$s =$*Win*
A				
G				

(e) What is the policy that we would derive from $Q_2^*(s,a)$? Answer by filling in the action that should be taken at each state in the table below (in case of draw, the action G is preferred).

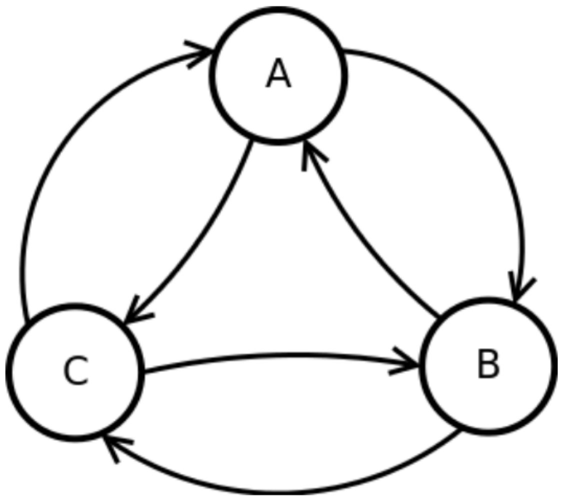
	$s =$*Uncertain*	$s =$*Certain*	$s =$*Lose*	$s =$*Win*

(f) What are the values $V_2^*(s)$ corresponding to $Q_2^*(s,a)$?

	$s =$*Uncertain*	$s =$*Certain*	$s =$*Lose*	$s =$*Win*

4. Policy Iteration [8 points]

Consider the following transition diagram, transition function and reward function for an MDP.



Discount Factor, $\gamma = 0.5$

s	a	s'	T(s,a,s')	R(s,a,s')
A	Clockwise	B	0.8	0.0
A	Clockwise	C	0.2	2.0
A	Counterclockwise	B	0.4	1.0
A	Counterclockwise	C	0.6	0.0
B	Clockwise	C	1.0	-1.0
B	Counterclockwise	A	0.6	-2.0
B	Counterclockwise	C	0.4	1.0
C	Clockwise	A	1.0	-2.0
C	Counterclockwise	A	0.2	0.0
C	Counterclockwise	B	0.8	-1.0

Suppose we are doing policy evaluation, by following the policy given by the left-hand side table below. Our current estimates (at the end of some iteration of policy evaluation) of the value of states when following the current policy is given in the right-hand side table.

A	B	C
Counterclockwise	Counterclockwise	Counterclockwise

$V_k^\pi(A)$	$V_k^\pi(B)$	$V_k^\pi(C)$
0.000	-0.840	-1.080

(a) What is $V_{k+1}^\pi(A)$?

Suppose that policy evaluation converges to the following value function, V_∞^π.

$V_\infty^\pi(A)$	$V_\infty^\pi(B)$	$V_\infty^\pi(C)$
-0.203	-1.114	-1.266

Now let's execute policy improvement.

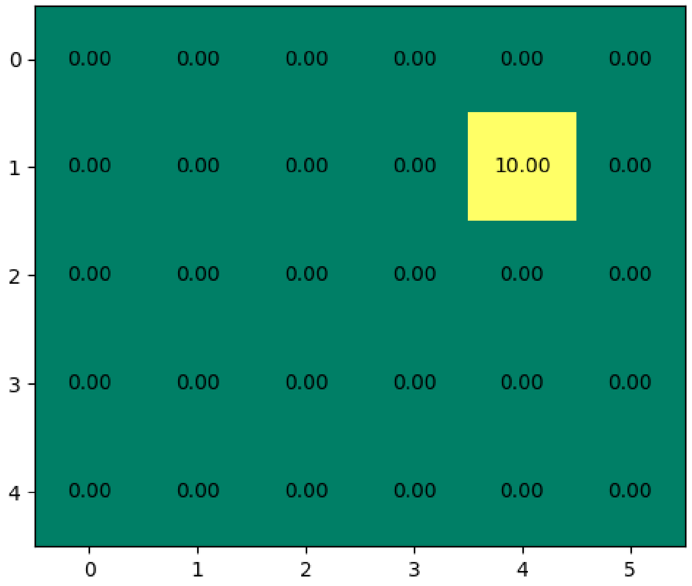
(b) What is $Q_\infty^\pi(A, clockwise)$?

(c) What is $Q_\infty^\pi(A, counterclockwise)$?

(d) What is the updated action for state A?

5. Q-Value Iteration [15 points]

Consider a 5x5 grid world. There is a golden state at row 1 column 4 with a reward of +10, all other states have a reward of 0. The start state is randomly initialized. The possible actions from each state are $\{up, down, right, left, stay\}$. The transition probability between all states is deterministic i.e. $T(s'|s,a) = 1$ for all states.



Implement a Q-Value Iteration algorithm for the infinite horizon case on this gridworld i.e. there is no terminal state, by completing the q_value_iteration() function in the gridworld_mdp.py file. Submit your clearly commented implementation along with plot of $Q(s,a)$ of the learnt solution. The gridworld_mdp.py script will automatically generate the plot for the values learnt at each state for each iteration along with the final learnt Q-values. Submit only the final Q-values plot generated by the script.