CS 182 FALL 2022, PROBLEM SET 4

Due: November 28, 2022 11:59pm

This problem set covers Lectures 15 through 19. The topics include Markov decision processes, reinforcement learning, decision trees, linear classification, and neural networks.

1. *Markov Decision Processes.* (25 points) In class we claimed that the utility estimates $U_t(s)$ under value iteration converge to $U(s)$ for all $s \in S$, where U is the utility of the optimal policy. Here we will prove this.

(1) (15 points) For any estimate of the utility function $\hat{U} : S \to \mathbb{R}$, we define the *Bellman backup operator* B which takes $\hat{U}$ as input and returns a new utility estimate $B\hat{U} : S \to \mathbb{R}$, defined for each $s \in S$ as

$$B\hat{U}(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s' \in S} P(s' \mid s, a)\hat{U}(s').$$

Prove that for any two utility estimates U', U'' (i.e., two arbitrary functions from S to $\mathbb{R}$),

$$\max_{s \in S} |BU'(s) - BU''(s)| \leq \gamma \max_{s \in S} |U'(s) - U''(s)|.$$

Hint: You might find the following inequality useful:

$$|\max_x f(x) - \max_x g(x)| \leq \max_x |f(x) - g(x)|$$

(2) (10 points) Let $U_0, U_1, U_2, \ldots$ be the utility estimates under value iteration, starting from an arbitrary U_0. Using the previous part, prove that for all $\epsilon > 0$ there exists $T \in \mathbb{N}$ such that for all $t \geq T$, $\max_{s \in S} |U(s) - U_t(s)| \leq \epsilon$.

Hint: $U_{t+1} = BU_t$ and $U = BU$. The former equality is the definition of value iteration; the latter equality is not completely trivial but you may use it without proof.

2. *Reinforcement Learning.* (20 Points) In this problem, you will be implementing various planning and reinforcement learning algorithms on OpenAI's Frozen Lake Environment. You will need the packages `gym==0.21.0`, `IPython==7.29.0`, and `matplotlib==3.4.3`.

In this environment, an agent controls the movement of a character in a 4x4 grid world. Some tiles of the grid are walkable (S, for start, F, for frozen, G, for goal), and others lead to the agent falling into the water (H, for hole). The agent is rewarded +1 for reaching the goal state and 0 otherwise.

We will work with a few variations of the Frozen Lake environment. In the first version, the parameter `is_slippery` is set to False, so every action leads to a deterministic tile. When `is_slippery` is set to True, the movement direction of the agent is uncertain. In particular, if an agent chooses to go in one direction, there is a 1/3 probability the agent goes in the intended direction and a 1/3 probability that the agent goes in each of the directions that are perpendicular to the intended direction. If an agent is on the edge of the map and attempts to move off the map, it simply stays in place.

(1) (2 points) Model this problem as a Markov Decision Process (MDP), formally specify the states (including terminal states), actions, and transition and reward functions.

(2) (5 points) Implement value iteration in `pset4a.py` by filling out the method `value_iteration` within the class `DynamicProgramming`. You may find `updated_action_values` to be a useful helper function when writing this.

(3) (5 points) Report the mean and variance of the rewards over 1000 episodes of the final policy using the parameters `gamma` $= 0.9$, `epsilon` $= 0.001$. For an agent using the policy found by value iteration, plot a histogram (include this in your PDF write-up) of the number of steps it takes an agent to reach the goal over those 1000 episodes using the final policy. If the agent falls into a hole in the ice and never reaches the goal state, let that be recorded as a zero. Does this agent always reach the goal? Why or why not? Use the map to inform your explanation.

(4) (5 points) Implement active, model free reinforcement learning in the form of Q-learning in `pset4a.py` by filling out the functions `choose_action` and `q_learning` within the class `QLearning`. Use $\alpha(k_{sa}) = \min(0.1, 10k_{sa}^{-0.8})$[1].

(5) (3 points) Plot the mean returns over 100 episodes of the Q-learning agent that acts solely based on max-Q values after every 1000 episodes (this should be done by using the `compute_episode_rewards` function). Use the parameters `gamma` $=$

[1]The technical conditions in order to theoretically guarantee convergence is that $\sum_{k_{sa}=1}^{\infty} \alpha(k_{sa}) = \infty$ and $\sum_{k_{sa}=1}^{\infty} \alpha(k_{sa})^2 < \infty$, and while you are welcome to change this so long as you converge to the correct value, this rate was chosen by staff as one that seems to work well in practice for this environment.

0.9, `epsilon` = 0.01. How does your Q-learning agent compare to the value-iteration agent following the policy derived from part 3?

3. *Decision Trees.* (15 points)

(1) (9 points) Consider the following dataset comprised of three binary input attributes A, B, and C, and one binary output. Use the algorithm Learn-DT to learn a decision tree for this data. Show the computations made to determine the attribute to split at each node and draw the resulting decision tree.

Example	Attribute A	Attribute B	Attribute C	Output
x_1	1	0	0	0
x_2	1	0	1	0
x_3	0	1	0	0
x_4	1	1	1	1
x_5	1	1	0	1

(2) (6 points) A decision *graph* is a generalization of a decision tree that allows nodes (i.e., attributes used for splits) to have multiple parents, rather than just a single parent. The resulting graph must still be acyclic. Now consider the *XOR* (or *parity*) function of *three* binary input attributes, which produces the value 1 if and only if an odd number of the three input attributes has value 1.

(a) (3 points) Draw a minimal-sized decision *tree* for the three-input XOR function.

(b) (3 points) Draw a minimal-sized decision *graph* for the three-input XOR function.

4. *Linear Classification.* (20 Points)

(1) *Logistic Regression.* (10 points) Use the `LogisticRegression` module of the `sklearn` Python library to implement a logistic regression model on the Iris dataset. Starter code is provided in `pset4a.py`. The Iris data set contains data about iris flowers. Each data point in the data set represent a separate flower examined. The X data describes the flower in terms of petal length and width. Our response vector y is a boolean vector of 0s and 1s indicating the true species classifications of these flowers into 2 distinct categories. Our goal is to build a model to classify iris flowers into these 2 species categories based on their petal dimensions alone. Our model will be both descriptive and predictive.

(a) (4 points) Plot this dataset with blue dots for $y = 0$ and orange dots for $y = 1$. Then run a logistic regression using the features $x_1 =$ `petal_length` and $x_2 =$ `petal_width` and plot the resulting decision boundary. Be sure to label your plot axes, include a legend and a title. Include this plot in your PDF write-up.
Note: Use the keyword argument `penalty='none'` when instantiating from the `LogisticRegression` class to avoid adding regularization.

(b) (3 points) What are the estimated logistic model coefficients and intercept?

(c) (3 points) What is the in-sample accuracy of your model at distinguishing between these 2 species? What is the baseline accuracy that you'd achieve by simply choosing the majority class every time? Does this logistic regression model provide a substantial improvement to that baseline?

(2) *Perceptron.* (10 points)

(a) (7 points) Implement the Perceptron algorithm with initial weights $\vec{w} = (1, 1, 0)$ using the toy data set contained in the starter code of `pset4a.py`. Please write down how many passes through the data it took for your algorithm to converge.

(b) (3 points) Does your algorithm converge when you add the point $((x_1, x_2), y) = ((2.5, 0), 1)$ to you dataset? Why or why not?
Note: Note, for both parts of this question regarding the perceptron algorithm, you should imagine a situation where the algorithm goes through the data set again and again (whereas in class we assumed only a single pass through the data). In part B of 4.2, we are asking you to consider convergence with respect to a finite data set by feeding it again and again into the perceptron algorithm in subsequent passes.

5. *Neural Networks.* (15 points)

(1) *Comprehension.* (5 points) Consider the following input "image":

1	2	1
0	2	1
1	2	1

Suppose we run this input through a forward pass of a simple convolutional neural network. The first layer is a convolution with kernel

1	-1
1	-1

with stride 1 and 0 padding of size 1 on all four sides. The second layer is an average pooling with 2×2 filters and stride 2. What is the output of these two layers?

(2) *Programming.* (10 points) In this problem, you will get practice training a neural network model on the MNIST dataset using the TensorFlow deep learning library. MNIST is a very famous dataset in neural networks and computer vision for demonstrating the power of deep learning on the task of image recognition. The dataset is composed of 10s of thousands of 28x28 hand-written digits 0 to 9 and our objective is to build an image classifier that is able to distinguish between them with very high accuracy. In `pset4b.py`, implement the `train` and `predict` methods of a object-oriented approach to building a network. The autograder will verify that you obtain at least 0.95 accuracy on the test dataset.

6. Collaboration, Calibration and References (5 points).

(1) With whom did you work on this problem set? What (if any) references and/or resources did you use beyond the course lecture slides and textbook?

(2) (5 points) Approximately how long did it take you to complete this problem set? Please complete this brief survey worth 5 points, graded for completion.