Ayush Kadakia (ank102)
CS 440: Introduction to Artificial Intelligence Assignment 3

Sheepdog Bot 1 Questions:

Overview of Sheepdog Bot 1:

Sheepdog 1 objective is to guide the sheep into the center of the grid to be captured in the pen. It utilizes value iteration to optimize the actions of the sheepdog by considering its own distance, and the distance of the sheep to the pen. It begins by initializing state spaces and actions the sheepdog bot can take. Through the use of value iteration, the Bellman Equation determines a value for each state based on maximum expected value of neighbors & rewards. Eventually the value will converge to reveal optimality through the use of value iteration. A reward function is further implemented, which incentivizes the sheep to be guided towards the center. All of this comes together to guide an optimal policy for the sheepdog bot 1 to follow.

1. What states are easy/immediate to determine T*(state) for?

   Certain states would be easier to determine T*(state) for. This includes such as when the sheep is already at the goal, in the pen. This is because this would require 0 moves, as the sheep is already in the goal state. Moreover, in instances where both the sheep and sheepdog bot are close to the pen, this would make it easier to determine T*(state), as the sheepdog bot now is able to have a more concise strategy to herd the sheep, with more limited moves. It certainly helps make it easier to determine T*(state) when the sheepdog is already in the sheep's field of view, as this takes away the step of having to reach the sheep; now it is a matter of trying to guide the sheep to the pen through influencing its movement path. Additionally, if the sheepdog is able to find the the sheep in a corner, this would limit the sheep's movement options, thus making the path to guide the sheep more concise; though not easy, this would make it just a bit simpler to determine T*(state) for as well.

2. For any other given state, express a formula for T*(state) in terms of a) The actions the bot can make b) How the sheep responds, and c) What state results. What are special cases you ought to consider?

   When devising a formula for T*(state), it is important to consider the three criteria mentioned in the problem. The actions of the bot, being able to move into any of its 8 neighboring cells and herding the sheep, the sheep being able to move up, down, left and right and its desire to charge the sheepdog, influence the game strategy and how we decide which state results. The current T*(state) = 1 + expected value of T(next). In this instance, 1 represents the current time. To build on top of this, we want to add T(next), or the moves that are to come. In terms of the actions of the bots and how the sheep responds, we can equate the expected value of T(next) to be ∑[(possible bot moves)*(probability of move)]; making this T*(state) = 1 + ∑[(possible bot moves)*(probability of moves)]. It is important to note that these moves are dependent on the actions of the sheep, and how the sheep itself responds to the moves of the bot. Resultantly, we can equate T*(state) to the sum of current & expected value number of moves. Special cases include when the sheep is already in the pen, in which T*(state)=0. Additionally, if the sheep cannot be captured for a certain state, then T*(state) = ∞.

3. If you could determine T*(state) for each state, how would you determine the optimal action for the sheepdog bot to take?

   With this determination, we would constantly pick moves at each state which would have the minimal T*(state). By constantly choosing the move for each state which would minimize the number of moves an optimal bot would need to take to catch the sheep, a course of action with the least number of moves would be determined. This would thus make this the optimal action.

4. Compute T*(state) for every state. If the sheep starts in the upper left corner, and the robot starts in the lower right corner, what's the expected number of moves needed to capture the sheep?

   Given the sheep starts in the upper left corner, and the robot starts in the lower right corner, we would need to perform value iteration on the Bellman Equation to calculate T*(state) for all states. Once the number has converged we would be able to find the optimal course of action. Using the Bellman Equation, we can express this as the number of moves needed to capture the sheep = T*(state) = 1 +(sum of probabilities of possible actions)*expected resulting state moves, where the expected resulting state moves = T*(state) from the next position = T*(next state). This can cohesively be written as T*(state) = 1 + (sum of probabilities of possible actions)* T*(next state). T*(state) can be recalculated and used for each iteration through value iteration until the values converge, which is when we can determine the expected number of moves to capture the sheep, based on the robot's initial position.

5. If the robot can start at any location, and the sheep will be introduced at a random empty spot - where should the robot start? Why? What is the expected number of moves to catch the sheep, when started in this location?

Given that the sheep will be introduced at a random empty spot, the robot should start in the center of the grid, near the pen. I argue that this would be the best location for the robot to spawn for a variety of reasons. First, this is a safe option to avoid having the robot to traverse across the entire grid if the robot and sheep spawn on complete opposite ends of the grids; worst case scenario, the robot will have to traverse about 10 spots to reach the field of view of the sheep. More so, being close to the pen in the center allows for the robot to make concise moves if the sheep is spawned near the center, or walks its way towards the pen. Moreover, being located in the center would allow the robot to traverse the grid in all 4 directions, as opposed to being in a corner and forced to move a certain way. For these reasons, I would choose the robot to start in the center near the pen. The expected number of moves to catch the sheep when started in this location would once again be dependent on the location of the sheep and its movements. This can be predicted through the use of the aforementioned Bellman Equation, in which value iteration can be used to constantly update T*(state) until the value converges, and we can make an educated prediction of the expected number of moves. Once again, the equation which can be used in this scenario would be T*(state) = 1 + (sum of probabilities of possible actions) * T*(next state).

6. Based on T*(state) and the optimal actions, simulate the sheepdog bot / sheepdog interactions. How does the simulated data compare to the compute expected value?

   The simulated data tends to be more leaning towards both sides of the extremes as compared to the computed expected value. For instance, in certain cases the sheepdog robot is not even able to catch the sheep in even one million turns. On the other hand, there are other instances where the sheep is caught in a couple thousand turns. This large discrepancy is something that was anticipated, however not reflected during the computed expected value. Through the computed expected value, this number is closer to the lower end of the extreme, as opposed to the million turns it sometimes took.

Sheepdog Bot 2 Questions:

1. How should you represent states as input to the model? What kind of model do you want to consider? What input features are relevant? What kind of error or loss are you using to assess your model? What training algorithm are you using? How can you compare the size of the fit model to the size of the fully computed T *?

The states should be represented by tuples that contain the information about the current state of the game. The kind of model we are considering is the Q-function approximator which will estimate the Q- values values for all the different state-action pairs.
The relevant input features are the position of the sheep and the sheepdog. Another relevant feature is the distance between them. All these features will help the model learn optimal actions.
The loss or error used to assess this model is the "Mean Squared Error" (MSE) which computes the values from the regression line between predicted Q values and the target Q values obtained from the Bellman equation during each Q-learning update.
The training algorithm used by Sheepdog2 was based on the descent-based optimization technique which will allow us to minimize the MSE.
We can compare the size of the fit model with the computed T* by measuring the size. In this context, we refer to the size as the number of values stored. The fit model, which approximates the Q-values, will have fewer values than the fully computed T* because it generalizes across states. T*. On the other hand, it represents the optimal expected number of moves for each individual state-action pair, resulting in a larger number of values to compute and store. The Q-function approximation is an efficient way to represent the Q-values for large state spaces and it will reduce memory requirements compared to storing T* directly.

2. How can you assess the quality of your model fit? Is overfitting an issue? If so, how can you avoid it?
   To assess the quality of the model fit we can split data into a training set and validation set. In the training, we can monitor the performance of the model in the validation set. This can provide us with an indication of how well the model is generalizing to unseen data. We can also assess metrics as accuracy, convergence speed, and evaluating the ability to make good decisions in various scenarios.
   Overfitting can be an issue for the model of the Sheepbot2. For our model the overfitting can occur when using the approximation. The overfitting will occur because our model will learn the noise from training data instead of generalizing to new undiscovered scenarios.
   We can avoid overfitting by using validation sets, regularization, exploring probabilities, or using early stopping.

3. Simulate Sheepdog Bot 2, starting with the sheep in the upper left corner, and the robot in the lower right corner. Does Sheepdog Bot 2 reliably catch the sheep? How does it compare in its performance to Sheepdog Bot 1?

Our sheepdog2 will start with an initially untrained Q-table and will use an epsilon-greedy exploration strategy. It might initially take time to explore and learn effective actions. It will depend on the number of episodes for training and the choice of hyperparameters, Sheepdog2 may require some episodes to learn a successful strategy. Sheepdog1 will be using value iteration, and should converge to an optimal policy if we give it enough time to compute fully accurate state-action values. Whereas Sheepdog2, will be using Q-learning with function approximation. It can converge to a good policy as well, but its convergence rate can vary based on the training schedule and hyperparameters.
Sheepdog2's strength lies in its ability to generalize its learning across different states. This is particularly advantageous in cases where the bot encounters new or undiscovered situations during game.
We can compare the performance from the beginning of the game. In the initial stages of training, Sheepdog2 may not perform as well as Sheepdog Bot 1 due to the exploration phase and the approximation of Q-values. As training progresses and Sheepdog2 updates its Q-table, it should improve its performance and start catching the sheep more effectively. The rate at which Sheepdog2 catches the sheep and its overall performance will depend on the effectiveness of its learning process.
To evaluate Sheepdog2's reliability in catching the sheep, we can measure its success rate (percentage of games won), and the average number of turns needed to catch the sheep. We can compare these metrics with the performance of Sheepdog1 under the same starting conditions.

Sheepdog Bot 3 Questions:

Overview of Sheepdog Bot 3:

The goal of Sheepdog 3 is to make a move based upon the actions of Sheepdog 1; in other words the goal is to generate data and build rules on how Sheepdog 1 moves from a variety of states. Based upon the most common move for a given state of Sheepdog 1, Sheepdog 3 will move accordingly. Input states would need to be converted into a simple format to be usable for training. As mentioned in the instructions of Sheepdog 3, it would be quite conspicuous to have this data in the form (input state, output action). Keeping this to two aspects would make it easier for processing. With this, we can utilize a decision tree classifier as the learning model; this can be a more simple take on the problem. After training the decision tree, we can integrate this into the Sheepdog 3 class. Within the move method, the decision tree can pick which move to take based upon what

Sheepdog 1 would do in a given state. We can use many different sets of data to train the tree, while holding others back to test the effectiveness of the model.

1. What is your input space? What is your output space? What is your model space? How are you assessing the error, and how are you reducing that error - i.e., what is your training algorithm?

   The input space would be the different input states that would be possible. This includes all of the different configurations of the sheep and sheepdog bot on the grid. As the data is to be held in the form (input state, output action), this would be the first part. The second part of this data would be the output space, which corresponds to the output action. The output space would in total consist of the sheepdog bot moving to either of the 8 adjacent spots, or staying in place. The specific output action would be dependent on the given input state. Therefore, we need to pick a model to satisfy this relationship, which would be a decision tree classifier in this situation. The model space as a whole would be all possible algorithms which can be utilized to satisfy this relationship. Based on research, some of these more complex methods that can be used instead of the more simple decision tree include neural networks, long short-term memory networks, & Gaussian processes; as mentioned, for our purpose we will stick with the more simple decision tree. The training algorithm would involve sorting through vast amounts of data between input states and the corresponding output that Sheepdog 1 would pick. The goal would be to find the decision rules which can predict the proper move. Error would be assessed through the use of a validation set, in which certain data points not used in the training will be exposed to the model. Based on the accuracy of the output of the model when exposed to these new input states, we can assess the effectiveness and accuracy of the model. We can reduce error through introducing a larger data set for inputs to learn from, as well as adjusting the model based on the validation set results. At the same time, we need to be mindful of how much data we introduce, in order to avoid overfitting.

2. How can you assess the quality of your model fit? Is overfitting an issue? If so, how can you avoid it?

   We can assess the quality of the model through a variety of ways. First, we can check if the model works properly with regards to the training data it is given. If so, this is a good first step to see that we are on the right track. Next, we would want to see if this model works well for the validation set as well. This is to ensure that the model will work for the large number of possible states in this problem, not just those that the model is trained on. One study from NYU Stern School of Business further suggests that plotting learning curves is a useful tactic for visualizing the effectiveness of a model. If the model only works accurately for training data and not the validation set, this indicates that overfitting is indeed an issue. In order to avoid overfitting, we can utilize a larger data set to account for more scenarios (training & validation sets), regularization, and monitor the performance of the validation set; once this starts to plateau we know overfitting has happened.

3. Simulate Sheepdog Bot 3, starting the sheep in the upper left corner, and the robot in the lower right corner. Does Sheepdog Bot 3 reliably catch the sheep? How does it compare in its performance to Sheepdog Bot 1?

   There are a few different scenarios which may occur in this instance. To begin, if the model reliably catches the sheep in the proper fashion, we know that a successful model has been built. On the other hand, if Sheepdog 3 is only able to catch the sheep in slower times aka more moves than Sheepdog 1, this is indicative of the model not being able to handle the complexities of the data too well. Inability to catch the sheep is indicative of a poor model being built. Overfitting can also be seen in this stage. This would make the model work well for training set examples, but fail when tasked with new and unseen input states. The ultimate goal would be for this model to surpass the performance of Sheepdog 1, through the model being able to generalize data at a superior rate. This would be more accomplishable with more sophisticated models other than the decision trees, such as some of those mentioned prior in question 1.