Project Report: Train a Smartcab How to Drive

Implement a Basic Driving Agent

*QUESTION: Observe what you see with the agent's behavior as it takes random actions. Does the **smartcab** eventually make it to the destination? Are there any other interesting observations to note?*

With the implementation of basic driving agent, the agent is able to randomly choose between 'none', 'forward', 'left' and 'right' actions, and move randomly throughout the grid/map. However, the movement is not optimal, and approximately 80% of the time, the agent could not reach the destination on time. Furthermore, the agent is not utilizing any information gained from the rewards gain from the previous iterations. Therefore, we can see some actions repeated in every iterations even though the actions gained penalty/negative rewards.

Inform the Driving Agent

QUESTION: *What states have you identified that are appropriate for modeling the **smartcab** and environment? Why do you believe each of these states to be appropriate for this problem?*

For the modeling of the smartcab, we choose traffic lights, the oncoming, left, and next_waypoint as the states. The traffic light state will help the agent to learn how to move without any traffic violation. The oncoming, and left state will provide the status of the surrounding environment so that the agent can learn how to move safely. Right side oncoming traffic state is not considered because right side traffic information does not affect agent's performance as per US traffic rule. The next_waypoint state is a very important part of the modeling because it can provide a relative direction towards the final destination that can be utilized to learn an optimal path reach the destination in time. Finally we also did not include 'deadline' state because this information is not relevant for agent to reach its destination. Furthermore, too much state space will slow down the computation.

Implement a Q-Learning Driving Agent

QUESTION: What changes do you notice in the agent's behavior?

The Q-learning algorithm is implemented by initializing and updating a table of Q-values at every iteration. The agent now pick the best action available based on the Q-values derived from the agent's current state.

After the implementation of Q-learning, the agent have a much better understanding of the surroundings. For example; the agent now does not choose those decisions that will cause penalty.

Therefore, the agent is able to learn the traffic rules, move towards the final destination rather than moving randomly. Although the implementation of simple Q-Learning algorithm improves the agent's performance, the overall success rate is still small (approximately 40%).

Improve the Q-Learning Driving Agent

QUESTION: *Report the different values for the parameters tuned in your basic implementation of Q-Learning. For which set of parameters does the agent perform best? How well does the final driving agent perform? Does your agent get close to finding an optimal policy, i.e. reach the destination in the minimum possible time, and not incur any penalties? How would you describe an optimal policy for this problem?*

The Q-Learning algorithm is implemented in such a way that it utilizes a lambda operator to select the action with highest Q-value. However, when it observes that multiple actions have same Q-values, it chooses the first action. In this way, the algorithm could not learn from other potential routes and yields a low accuracy. In order to overcome this, we introduce a random action choice where the agent that can randomly choose from the list of actions that might results same Q-value. This way, the learning algorithm explores the all the alternative paths. As a result, the agent can learn more effectively and success rate increases significantly.

Next the model is tuned for both alpha (learning rate) and gamma (discount factor) to find the optimal values. During model tuning, we observed that gamma does not have a significant impact on the output. This happened due to the nature and objective of the problem. Agent's goal is to reach the destination in time and it does not receive any reward for reaching the destination by following the shorter path or taking less time than the other successful iterations. The result of tuning alpha is presented in table 1. The optimal value for alpha in this problem is 0.2 with 99.9% success rate for 1000 iterations. The model can be further improved by using decaying learning rate i.e. learning rate is large at the start of the problem and it will decay linearly as the model progresses.

Alpha	Success (n_trials = 1000)
0.5	99.7%
0.4	99.8%
0.3	99.7%
0.2	99.9%
0.1	99.9%

From the performance of the agent, we can observe most of its mistakes occur in the first few iterations. Success rate in first 10 iterations and last 10 iterations are approximately 90% and 100%. We also observed that the agent sometimes break traffic rules in the first few iterations and get penalty/negative rewards. As the iteration increases, the agent learns from its mistakes and starts to reach destination within time limit. Initially, the total rewards are relatively large, however, relatively small rewards are observed towards the end. This indicates agent's performance is increasing over time and it is taking less time and fewer steps to reach destination. However, I do not think it is an optimal policy for this problem because it does not include any rewards for using less time or fewer steps. So if we include rewards for using less time or fewer steps, agent will learn more quickly and more efficiently.