Scott Burns
Machine Learning Engineer - Project 4
Train a Smartcab to Drive

Introduction

This report outlines findings when implementing a simple reinforcement learning agent - a "smartcab" which navigates a grid of streets and other vehicles, attempting to reach a destination several blocks away.

Implement a Basic Driving Agent

*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?*

Prior to any updates of the agent.py file, the agent (red cab in the pygame window) doesn't move. Regardless of the time limit, the cab does not make it to the target destination.

Inform the Driving Agent

*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?*

After trying several different combinations for inputs to describe state for the learning agent, I decided to include the following elements:

- Traffic light value
- Car present ('oncoming')
- Previous smartcab action
- Pre-previous smartcab action (2 actions previous)

I chose to use the first two elements (traffic light value and 'oncoming') as they are critical for avoiding traffic rule violations, and reinforce agent decisions that avoid these penalties. The remaining two elements (previous action and pre-previous action) I chose for their value in potentially informing more useful navigation around the grid (e.g. potentially avoiding circular behavior), which might help the smartcab to reach its destination more rapidly and earn the completion reward.

Initially I included a longer list of variables (including time, number of steps in specific direction and others), but discovered that the cab found it difficult to learn from a much wider set of states in the q-table, as it was less likely to encounter a new state that could be informed by similar previous state, with multiple actions and their corresponding rewards.

Scott Burns
Machine Learning Engineer - Project 4
Train a Smartcab to Drive

*How many states in total exist for the **smartcab** in this environment? Does this number seem reasonable given that the goal of Q-Learning is to learn and make informed decisions about each state? Why or why not?*

There are 2028 potential states for the smartcab in an environment defined by my chosen inputs. This follows as the product of two valid traffic light states and four states each for: 'oncoming', 'left', 'right', 'last_action' and 'before_last_action', or 2x4^5.

The inputs I've chosen seems to strike a reasonable balance between being sufficiently detailed to offer insight on the best potential action at a given state, while avoiding excessive specificity, that could hinder a cab from generalizing from previous experience.

Implement a Q-Learning Driving Agent

What changes do you notice in the agent's behavior when compared to the basic driving agent when random actions were always taken? Why is this behavior occurring?

After implementing Q-Learning (prior to including 'last_action' and 'before_last_action'), I noticed that my agent seemed to improve slightly in avoiding penalties for traffic violations, but that it would also often get stuck traveling in repetitive concentrated travel. For example, the cab would often circle in one corner of the traffic grid. Following longer trial sets of repeated trial simulations, the agent started to more often reach the destination.

This behavior occurs because the agent recognizes some patterns in the rewards earned given actions for a particular state. As a result, when presented with states where there is a risk of incurring penalties for traffic violations, the agent may 'choose' an action that under similar state features may have avoided a penalty previously. For some of these test runs, I may have also set the exploration rate too low, which could have caused the cab to continue selecting low reward actions from similar states without trying new, random choices to learn their rewards.

Improve the Q-Learning Driving Agent

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?

To optimize my Q-Learning algorithm I added 4 key features:

1. Strict randomization in the case that the best known q-table rewards are below a constant threshold

2. Discounting of rewards achieved at higher values of a 'time' index - using a 'discount factor' (gamma) raised to a power equal to the time unit and multiplied times the reward of an action
3. Addition of action randomization according to a constant 'exploration rate' like the epsilon proposed in later Reinforcement Learning Core Lessons
4. Inclusion of 'last action' and 'before last action' inputs for the q-table

With these updates, for trial sets (or simulations) of around 50 trials, my cab was able to fairly consistently achieve average trial scores of over 5.

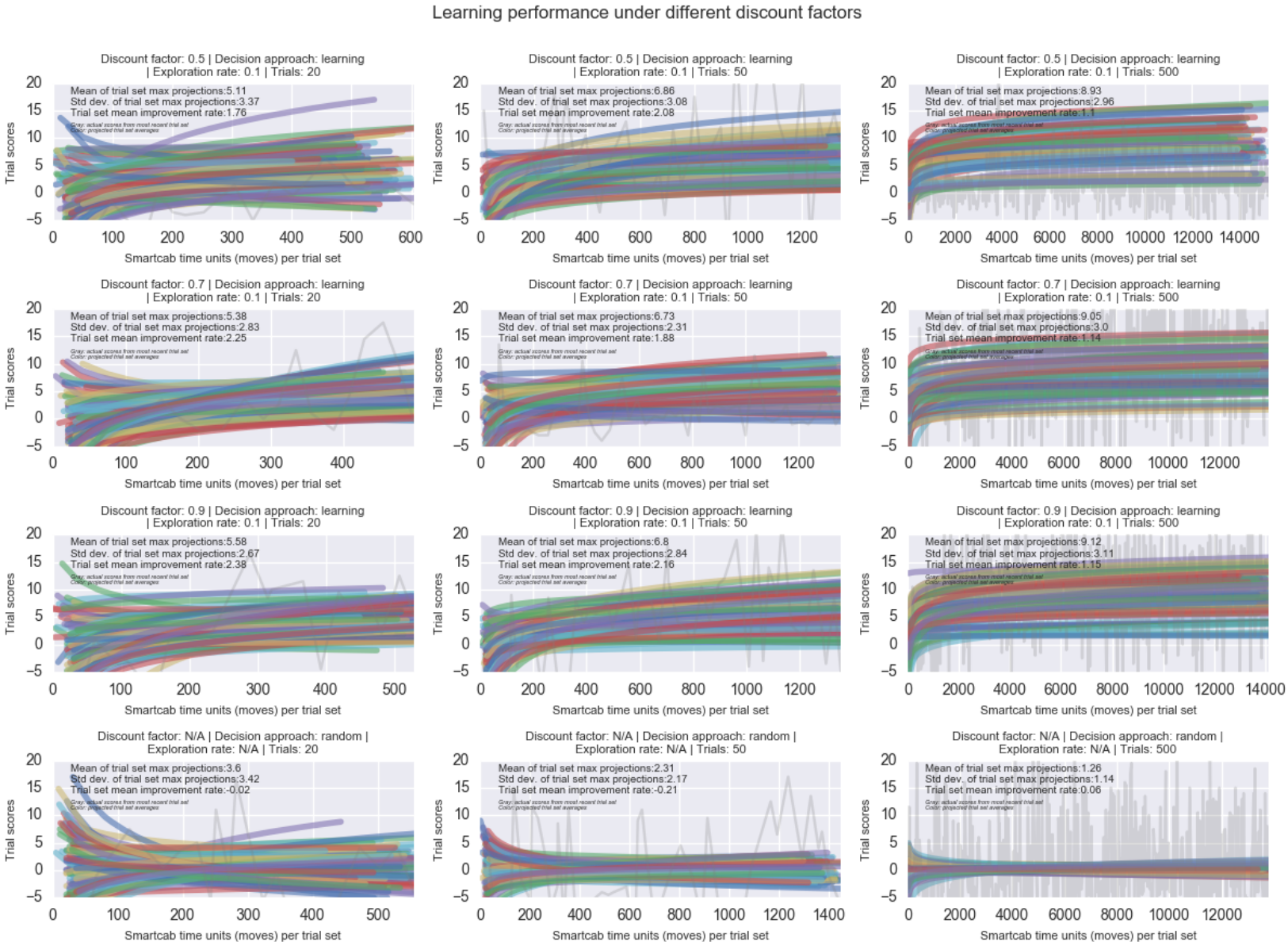
To definitively understand how variations to parameters affected model performance, I added a 'tester' (smartcab/tester.py) file to generate multiple plots of level-log regression curves over time (each agent move). These curves were fit to each smartcab's scores paths over multiple trials, through which the agent retains, and learns from a common q-table. Each plot of trial sets showed performance for specific parameter values, faceted in a grid.

For each set of parameters I generated three summary metrics to highlight how they affected smartcab navigation performance. These included the

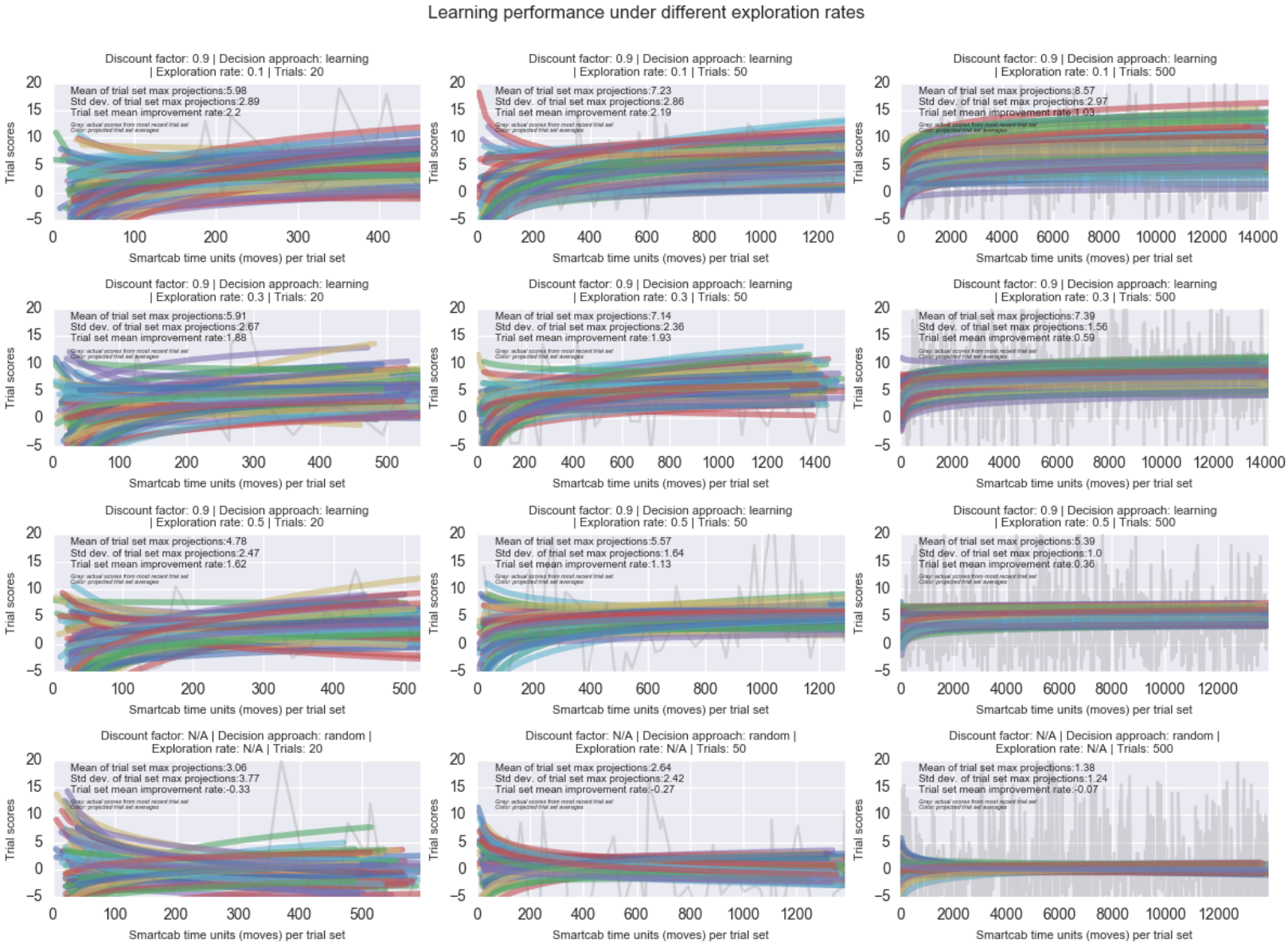
- **Trial set mean improvement rate** – the coefficient of the fitted regression curve - *score = improvement rate * log(time) + constant* – an indicator of how rapidly the smart cab was 'learning' how to navigate
- **Mean of trial set max projections** – the average maximum scores per trial reached over the fitted curves for multiple trial sets,
- **Standard deviation of trial set max projections** – the standard deviation of maximum scores per trial reached over the fitted curves for multiple trial sets

Two faceted charts below show the impact of discount factor and exploration rate on cab navigation performance. The impacts are shown for three values of trials per trial set – 20, 50 and 500 – over 100 total trial sets in each plot.

Scott Burns
Machine Learning Engineer - Project 4
Train a Smartcab to Drive



Scott Burns
Machine Learning Engineer - Project 4
Train a Smartcab to Drive



Scott Burns
Machine Learning Engineer - Project 4
Train a Smartcab to Drive

Based on the charts, I see evidence that

1. A higher discount factor (less aggressive discounting over time) of 0.9 may outperform lower discount factors of 0.7 and 0.5 in terms of maximum projected trial score value and improvement rate
2. An agent with a lower exploration rate, of around 0.1 seems to substantially outperform agents with higher exploration rates of 0.3 and 0.5
3. Learning algorithms dramatically outperform smartcabs choosing actions at random (as we would hope), and this outperformance becomes more obvious over trial sets with more trials, as the agent has more time to learn
4. There seems to be a slight trade-off of mean performance for performance variance with increasing discount factors
5. Increasing discount factor parameters in testing raises max projection standard deviation, while reducing max projection means

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?

In the sample trial sets shown in the charts above, average trial scores for discount factors around 0.9 and exploration rates around 0.1 over trial sets of 100 typically reach above 8.5. While this is not an optimal value, the figure does suggest that the cab is fairly capable of avoiding most traffic violations and frequently reaching the goal, given that the total payoff to a successful arrival is 10 points. Maximum achieved values max values over all trial sets reach higher, some around 15 points per trial.

An optimal policy would describe the correct action for every state possible given the smartcab's position on the grid relative to the destination, the traffic light state at the waypoint and the state of any other vehicles on the grid. I would expect an optimal policy to yield a score value of 25 points or more for a trial.

References

http://www.pysnap.com/reinforcement-learning-in-python/
http://stackoverflow.com/questions/8023306/get-key-by-value-in-dictionary
http://stackoverflow.com/questions/29885408/how-plot-ployfit-n-log-n-with-python
http://stackoverflow.com/questions/8248467/matplotlib-tight-layout-doesnt-take-into-account-figure-suptitle
http://matplotlib.org/users/text_props.html
http://stackoverflow.com/questions/14770735/changing-figure-size-with-subplots
https://classroom.udacity.com/nanodegrees/nd009/parts/0091345409/modules/540405889375460