COMP 4780 Assignment 1 – Reinforcement Learning with Feature Extraction

20 Marks Total

Handed out: January 31, 2022

Due: February 18th, 2022 – by 11:59 pm

It's recommended that you use an up to date install of Python. Anaconda is a nice distribution which includes useful packages.

1. **[16 Marks] Develop a sophisticated feature extractor**

Pacman wants to learn from prior experience how to play. The only problem is that the number of possible states for normal games of pacman is much too big to learn from experience.

For example:

python pacman.py -p PacmanQAgent -x 2000 -n 2010 -l smallGrid

Gives Pacman 2000 sample runs to learn, and then plays 10 rounds on a tiny grid. If you reduce the number of training rounds by too much, Pacman doesn't know enough about the game state. For a larger board, things are hopeless. The only recourse, is to reduce the game to a simpler representation. This is done by defining features for Pacman to perform Q-Learning on rather than the raw game state.

Your task for this question, is to build a feature extractor with the aim of maximizing Pacman's score. A basic feature extractor (that isn't all that bad) is provided for you as an example. Create your own feature extractor by modifying the myExtractor class in the theFeatureExtractors.py file.

Commands:

(To run the existing SimpleExtractor)

python pacman.py -p ApproximateQAgent -a extractor=SimpleExtractor -x 50 -n 60 -l mediumGrid

(To run the feature extractor that you'll write for this problem)

python pacman.py -p ApproximateQAgent -a extractor=myExtractor -x 50 -n 60 -l mediumGrid

python pacman.py -p ApproximateQAgent -a extractor=myExtractor -x 50 -n 60 -l mediumClassic

Points for this question will be awarded based on how sophisticated your solution is (8), and on how well Pacman performs (8). Your goal is to maximize Pacman's score. The solution to this problem should be "general purpose" in the sense that it is well adapted to boards typical of the standard classic Pacman board. (e.g. mediumClassic is a good benchmark). Your solution will be tested using either 100 or 200 rounds of training (state your choice in your README). This training should not exceed 5 minutes in total for the mediumClassic layout. (Your laptop is likely slower than the test machine – but probably not more than a factor of 0.5). If you develop a more sophisticated solution but have to comment out some features to meet this constraint, mention this in your lastname_firstname.txt file.

2. **[4 Marks] Make a layout**

Design a Pacman layout with the name **mylayout.lay**. Your goal should be to design a board with the following criteria:

a. The agent you developed in part 1 performs (reasonably) well

b. You expect a less sophisticated feature extractor to experience problems (e.g. SimpleExtractor as one example)

Examine the provided layouts for format, and for ideas on your own board. Points for this question will be based on how well a and b are satisfied. If you have any comments on your proposed layout, you may optionally include these in your written description.

Other useful information regarding files:

Important Python files:

theFeatureExtractors.py – This is the file that you'll be editing to include your own feature extractor. This maps (state,action) pairs to features used for approximate Q-learning (in qlearningAgents.py).

Useful to look at:

The "layouts" directory – for examples of the layout format required for part 2.

valueIterationAgents.py - A value iteration agent for solving known MDPs (T and R are known) for general understanding of MDPs

qlearningAgents.py - Q-learning agents for Gridworld, Crawler and Pacman – examples of general learning using Q-learning

mdp.py – Methods on general MDPs.

learningAgents.py - Defines the base classes ValueEstimationAgent and QLearningAgent, extended in

valueIterationAgents.py and qlearningAgents.py.

util.py – General utilities that may be useful.

Not especially important:

environment.py - Abstract class for general reinforcement learning environments. Used by gridworld.py.

graphicsGridworldDisplay.py - Gridworld graphical display.

graphicsUtils.py - Graphics utilities.

textGridworldDisplay.py - Plug-in for the Gridworld text interface.

graphicsCrawlerDisplay.py - GUI for the crawler robot.

To hand in:

You can hand in all files, or only your featureExtractors.py file, mylayout.lay and a lastname_firstname.txt file.

For the text file you should include the following:

1. For part 1 you should explain what features you chose and your reasoning for choosing those features. You should also state what score you expect to achieve for mediumGrid and mediumClassic so that there aren't any surprises when the code is evaluated.

2. You should discuss briefly your layout. How did you decide on the layout, what makes it special, and what makes it so that *your* agent might perform well while a simpler agent might fail. Compare the performance of your agent on your layout to the SimpleExtractor in terms of average score.