CS7646 ML4T
Machine Learning
for Trading

# Project 7: Q-Learning Robot Documentation

## QLearner.py

***class*** `QLearner.QLearner`(***num_states=100, num_actions=4, alpha=0.2, gamma=0.9, rar=0.5, radr=0.99, dyna=0, verbose=False***)

This is a Q learner object.

**Parameters**

- **num_states** (*int*) – The number of states to consider.
- **num_actions** (*int*) – The number of actions available..
- **alpha** (*float*) – The learning rate used in the update rule. Should range between 0.0 and 1.0 with 0.2 as a typical value.
- **gamma** (*float*) – The discount rate used in the update rule. Should range between 0.0 and 1.0 with 0.9 as a typical value.
- **rar** (*float*) – Random action rate: the probability of selecting a random action at each step. Should range between 0.0 (no random actions) to 1.0 (always random action) with 0.5 as a typical value.
- **radr** (*float*) – Random action decay rate, after each update, rar = rar * radr. Ranges between 0.0 (immediate decay to 0) and 1.0 (no decay). Typically 0.99.
- **dyna** (*int*) – The number of dyna updates for each regular update. When Dyna is used, 200 is a typical value.
- **verbose** (*bool*) – If “verbose” is True, your code can print out information for debugging.

`query`(***s_prime, r***)

Update the Q table and return an action

**Parameters**

- **s_prime** (*int*) – The new state
- **r** (*float*) – The immediate reward

**Returns**

The selected action

**Return type**

int

`querysetstate(s)`

Update the state without updating the Q-table

**Parameters**

**s** (*int*) – The new state

**Returns**

The selected action

**Return type**

int

## testqlearner.py

`discretize(pos)`

convert the location to a single integer

**Parameters**

**pos** (*int, int*) – the position to discretize

**Returns**

the discretized position

**Return type**

int

`getgoalpos(data)`

find where the goal is in the map

**Parameters**

**data** (*array*) – 2D array that stores the map

**Returns**

the position of the goal

**Return type**

tuple(int, int)

`getrobotpos(data)`

Finds where the robot is in the map

**Parameters**

**data** (*array*) – 2D array that stores the map

**Returns**

the position of the robot

**Return type**
int, int

`movebot`(***data***, ***oldpos***, ***a***)

move the robot and report reward

**Parameters**

- **data** (*array*) – 2D array that stores the map
- **oldpos** (*int, int*) – old position of the robot
- **a** (*int*) – the action to take

**Returns**
the new position of the robot and the reward

**Return type**
tuple(int, int), int

`printmap`(***data***)

Prints out the map

**Parameters**
**data** (*array*) – 2D array that stores the map

`test`(***map***, ***epochs***, ***learner***, ***verbose***)

function to test the code

**Parameters**

- **map** (*array*) – 2D array that stores the map
- **epochs** (*int*) – each epoch involves one trip to the goal
- **learner** (*QLearner*) – the qlearner object
- **verbose** (*bool*) – If "verbose" is True, your code can print out information for debugging.
  If verbose = False your code should not generate ANY output. When we test your code, verbose will be False.

**Returns**
the total reward

**Return type**
np.float64