Machine Learning – Implementing RL Algorithms

Introduction:

We are going to explore the domain of Reinforcement Learning (RL). We will apply different methods on two 'interesting' MDPs and analyze findings as per the following 3 parts:

Strategy:

Part A: Discussion on the 2 MDPs chosen and why they are interesting to study

Part B: Solving MDP1 – Frozen Lake and Analysis using VI, PI and Q-Learning

Part C: Solving MDP2 – Forest Management and Analysis using VI, PI and Q-Learning

Part D: Additional Comparison between MDP1 & MDP 2 – IHULU TIME COMPARISON/Cost comparison log o3

Strategy: Epsilon & Alpha Greedy Strategy (Move this to end)

We need a 'greedy' strategy in order to balance exploitation vs exploration of the environment by our agent. This trade-off is managed by the parameter exploration rate denoted by epsilon, e. Think of it as the probability that our agent will explore rather than exploit the world. We start with e = 1 for agent to explore and as the agent learns more about the environment, at each new episode, epsilon decays so likelihood of exploration decreases and exploitation increases and thus agent becomes greedy for exploitation.

Choosing action: We generate random number, 0<r<1 if r>e, action chosen via exploitation to find highest q- value for its current state from q-table and if r<e, chosen via exploration i.e. random choice is made.

Understanding Learning Rate: 0<alpha<1 – used as a tool to determine how much information we keep about the previously computed q-value for the state-action pair in contrast to the new q-value for the same state-action pair at a later time step.

Part A) Interesting Markov Decision Processes (MDPs):

The two MDPs are as follows:

	Frozen Lake *Open AI Gym*	**Forest** *Hiive MDPTOOLBOX*
#States	64 (8x8)	640 (10 times more states)
World	Grid	Non-Grid
Decision Type	Frozen Surface or Hole	Cut forest or Wait
Problem Type	Stochastic Maze	Stochastic (Probabilistic) transition to another state

This table describes the nature of two MDPs being studied!

From RL standpoint, what makes them **interesting** is the fact that they represent 'small' number of states in a grid world VS 'large' no. of states in a non-grid world as well as few other factors listed in the table on the left!

These are interesting also because they have so many real-world applications. E.g. Autonomous Vehicles exploring an unknown terrain. Consider we send a rover to the surface of mars or any other planet; RL or Deep Learning RL will be the core of its search algorithm. Instead of relying on back and forth communication with earth, an Agent will have to learn from the environment and make decisions. Now, MDP environments can be of multiple types but we will be focusing on the following:

Frozen Lake: A grid environment of a lake with holes, taken from Open AI Gym. Agent traverses a lake divided into 4x4 = 16 tiles. Each tile is either a frozen surface or a hole. Agent begins from the start tile 'S' and aims to reach the goal tile 'G'. Agent earns rewards for avoiding obstacles i.e. holes and finding a path to the non-hole tile. However,

frozen also indicates slippery tile and hence agent may not always go in the intended direction. There are only 4 possible directions for the agent to move – Up, Down, Right and Left.

Forest Management: A non-grid environment with a growing forest, taken from hive mdptoolbox. Agent has a CUT or WAIT decision to make at every step. Depending on this decision, the transition states change between 0 and 1. When agent decides to CUT forest , it gets reward of +1 and forest transitions to state-0 whereas when agent decides to WAIT, forest passes to next state i.e. state-1 with probability 'q' and goes back to previous state i.e. state-0 with probability (1-q) because of forest fires. Agent makes a choice weather to risk going to final state and get higher reward or to receive +1 reward by cutting this forest at any given state. And in final state, if agent CUT forest, it earns reward r1 and goes back to state-0 in contrast to WAIT, where it can earn reward r2 and remains in current state with probability 'q' but goes to state-0 with probability (1-q).

`mdp_example_forest` *Generates a MDP for a simple forest management problem*

Description

Generates a simple MDP example of forest management problem

Usage

```
mdp_example_forest(S, r1, r2, p)
```

Arguments

`S`	(optional) number of states. S is an integer greater than 0. By default, S is set to 3.
`r1`	(optional) reward when forest is in the oldest state and action Wait is performed. r1 is a real greater than 0. By default, r1 is set to 4.
`r2`	(optional) reward when forest is in the oldest state and action Cut is performed r2 is a real greater than 0. By default, r2 is set to 2.
`p`	(optional) probability of wildfire occurence. p is a real in]0, 1[. By default, p is set to 0.1.

Details *

mdp_example_forest generates a transition probability (SxSxA) array P and a reward (SxA) matrix R that model the following problem. A forest is managed by two actions: Wait and Cut. An action is decided each year with first the objective to maintain an old forest for wildlife and second to make money selling cut wood. Each year there is a probability p that a fire burns the forest.

* *https://miat.inrae.fr/MDPtoolbox/QuickStart.pdf*

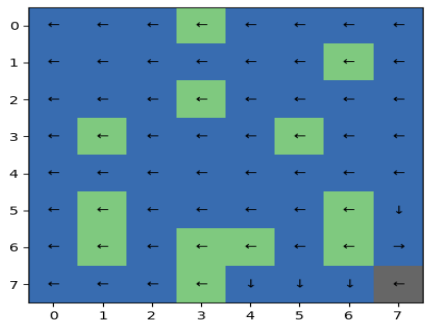
Part B) Solving MDP1: Frozen Lake 8x8

current state with probability 'q' but goes to state-0 with probability (1-q).

epsilon (float) – Stopping criterion. The maximum change in the value function at each iteration is compared against epsilon. Once the change falls below this value, then the value function is considered to have converged to the optimal value function.

https://pymdptoolbox.readthedocs.io/en/latest/api/mdp.html#mdptoolbox.mdp.PolicyIteration

This is a classic and simple grid-world wherein we have reward as +1 for the goal(final) state and 0 in case of obstacles(hole). Agent has to start from top-leftmost grid-cell and find its way to the goal state by avoiding falling into holes. The following is a glimpse of my 8x8 frozen lake with Blue color – Frozen surface, Green color – Holes and Black color – Goal state:



Agent can take Up, Down, Right or Left actions. However, surface is slippery and the grid is stochastic. Environment changes with probabilities. E.g. Agent decides to take Up action, it moves in the Up direction with a probability of 1/3 and it moves Right & Left with 1/3 probability.

Total States = 8x8 = 64

Apart from video lectures, I followed the book – Reinforcement Learning by Richard S. Sutton

and Andrew G. Barto for my code preparation and implementation of Frozen Lake environment using python.

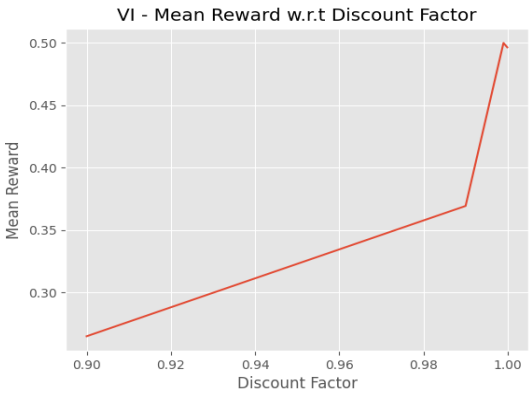
B1: Value Iteration

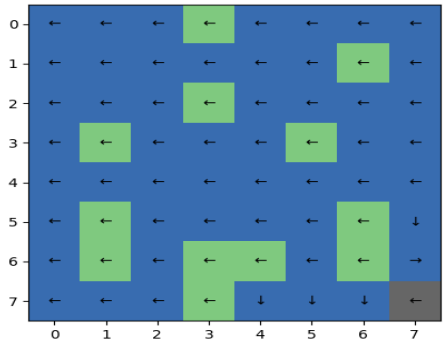
I did VI with varying epsilon values [0.1, 0.001, 0.0000001, 0.0000000001] which affects the convergence iterations of the process as well as varying the discount factor [0.9, 0.99, 0.999, 0.9999] which affects the final values at the end of each episode. I calculated mean reward at the end of 1000 iterations in each case.

B1.1 Analysis with Changing Discount Factor keeping epsilon values constant

Looking at the graph below, at higher discount factor values, we get improved rewards leading to better policies.

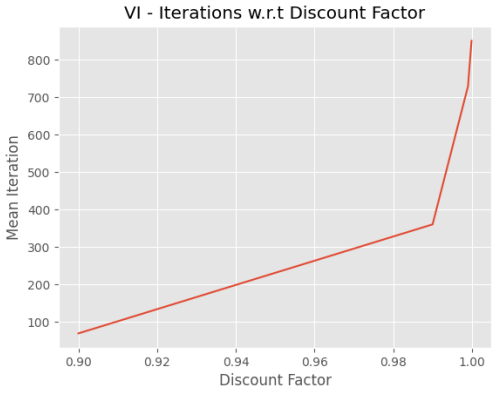
As discount factor becomes greater than 0.98, mean rewards shoot up and maximizes at 0.999. This indicates that in environments with lower discount rates and with states that are far away from the goal state, it is difficult for the algorithm to assign optimal values for those states. At 0.999, VI performs the best in terms of assigning optimal values and creating good policy.

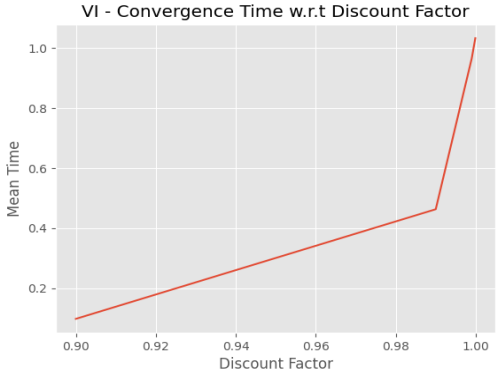




The grid on the right shows the "optimum policy" at discount factor of 0.999 (and epsilon 1e-7). As we can see, the optimum policy always tries to get away from holes and manages to arrive the goal state ~50% of the times.

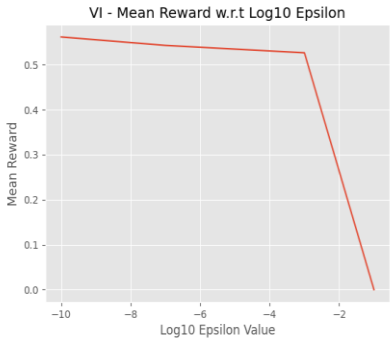
The graphs below compares the iterations to converge and average time taken to the optimum values w.r.t discount factor. We can see that increasing the discount factor (over 0.98), increases the number of iterations and time taken to converge. However, highest time taken is still under 1 second with iterations maxing out at 720 for discount factor 0.999. Considering all 3 graphs in this section, I'd conclude that for 8x8 environment with 64 states, VI converges best with discount factor of 0.999!

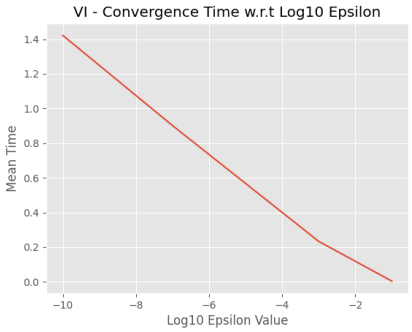


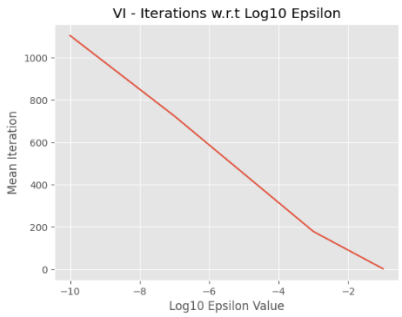


B1.2 Analysis with Changing Epsilon values keeping discount factor constant

Looking at the graphs below, as we increase epsilon, mean reward reduces. At lower values of epsilon, VI is able to perform better in terms of gaining rewards. Reward maximizes at epsilon of 1e-7. This reward is greater than 0.5, which we got by varying, discount factor in previous section. This indicates that changing epsilon has a slightly greater impact on mean reward gained as compared to changing discount factor.







When we look at the graphs for iterations to converge and the duration or time taken w.r.t epsilon values, we see that increasing epsilon value, reduces the time taken and the number of iterations to converge to optimum values.

I'd like to conclude from the analysis in section B1.1 & B1.2, that the Value Iteration Algorithm favors an environment with lower epsilon value and a higher discount factor.

B1: Value Iteration

I did VI with varying epsilon values [0.1, 0.001, 0.0000001, 0.0000000001] which affects the convergence iterations of the process as well as varying the discount factor [0.9, 0.99, 0.999, 0.9999] which affects the final values at the end of each episode. I calculated mean reward at the end of 1000 iterations in each case.

B1.1 Analysis with Changing Discount Factor keeping epsilon values constant

Part C) Solving MDP2: Forest Management

This is a non-grid world wherein we do not have just one reward for the goal(final) state, instead, we are dealing with multiple rewards (r1,r2) that agent can get by deciding whether to cut or wait in the final state. This MDP is solved using the discount factor (gamma) parameter as the decision factor to cut or not to cut forest.

The discount factor essentially determines how much the RL agent cares about rewards in the distant future relative to those in the immediate future. If γ=0, the agent will be completely myopic and only learn about actions that produce an immediate reward. If γ=1, the agent will evaluate each of its actions based on the sum total of all of its future rewards. I have set it to a constant value of 0.999 for the purpose of my study because changing discount changes the optimal policy, which could cause confusion. Also, greater number of states require a greater discount in order to converge faster in finite amount of time.

Unlike Frozen lake, here, I am using this discount factor as environment variable as well as for the rewards.

For rewards, since we are dealing with a high number of states, hence, I used r1 = 100 for cutting forest in final stage and r2 =10 for waiting and keeping the forest in that state.

C1. Value Iteration: I did VI with constant discount factor but varying epsilon values [0.1, 0.001, 0.0001, 0.0000001, 0.0000000001] and store the results in my custom data frame as follows:

640 states:

My VI dataframe w.r.t Changing Epsilon

	Epsilon	Log10 Eps	Policy	Iterations	Time	Mean Reward	Value Function
0	1.000000e-01	-1.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	130	0.183698	2.385200	(57.52189743626019, 58.04846213077683, 58.0484...
1	1.000000e-03	-3.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	174	0.232355	2.170125	(75.43406299781395, 75.96062820808476, 75.9606...
2	1.000000e-04	-4.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	195	0.280079	2.498389	(83.70902497063832, 84.23559018625782, 84.2355...
3	1.000000e-07	-7.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	260	0.362331	2.594689	(108.2473323339396, 108.77389754904095, 108.77...
4	1.000000e-09	-9.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	304	0.400819	2.581574	(123.97490027872962, 124.5014654938315, 124.50...

64 states:

My VI dataframe w.r.t Changing Epsilon

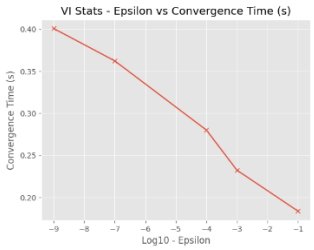
	Epsilon	Log10 Eps	Policy	Iterations	Time	Mean Reward	Value Function
0	1.000000e-01	-1.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	130	0.024004	17.813644	(57.52189743626019, 58.04846213077683, 58.0484...
1	1.000000e-03	-3.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	174	0.024282	15.033072	(75.43406299781395, 75.96062820808476, 75.9606...
2	1.000000e-04	-4.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	195	0.024000	14.046721	(83.70902497063832, 84.23559018625782, 84.2355...
3	1.000000e-07	-7.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	260	0.040941	17.333811	(108.2473323339396, 108.77389754904095, 108.77...
4	1.000000e-09	-9.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	304	0.050033	17.143698	(123.97490027872962, 124.5014654938315, 124.50...

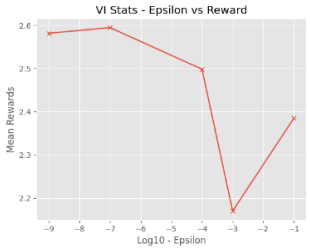
On closer inspection, I noticed that the optimum policies are same despite different values of epsilon. This hints the fact that we obtain one optimum policy for a constant discount factor which is as expected because the discount factor is the environment variable that should affect the policy and not epsilon for VI.

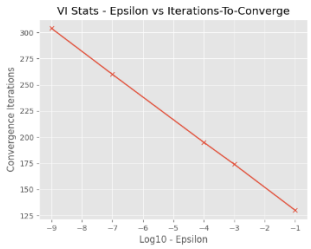
Looking at the criteria of 'maximum reward at minimum iterations and least amount of time' in the graphs below, we see that epsilon = 0.1, is the best one for obtaining the optimum policy – which converged at 130 iterations.

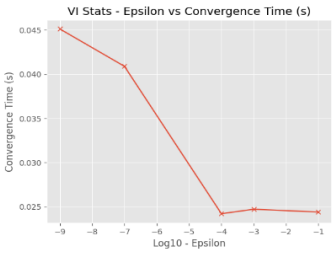
Smaller epsilon values taking more iterations to converge and hence, we also see a longer time.

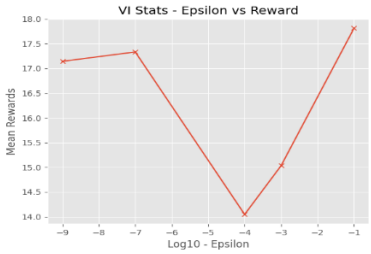
Reward to Time ratio – 13.2, 9.43, 8.89, 7.19, 6.45; Iterations – Highlight Best Ratio

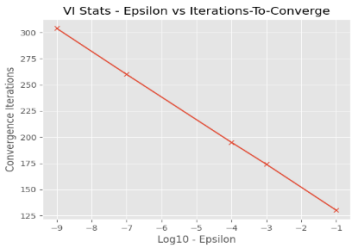












C2. Policy Iteration: I did PI with constant discount factor but varying epsilon values [0.1, 0.001, 0.0001, 0.0000001, 0.0000000001] and store the results in my custom dataframe as follows:

Reward to Time ratio – 0.92, 0.63, 0.56, 0.44, 0.33; Iterations – Highlight Best Ratio

640 States:

My PI dataframe w.r.t Changing Epsilon

	Epsilon	Log10 Eps		Policy Iterations	Time	Mean Reward	Value Function
0	1.000000e-01	-1.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	51	2.450036	2.252048	(473.342763392478, 473.8693286075798, 473.8693...
1	1.000000e-03	-3.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	51	4.002533	2.521786	(473.43385555356514, 473.9604207686668, 473.96...
2	1.000000e-04	-4.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	51	4.423113	2.507937	(473.434691643851, 473.96125685895294, 473.961...
3	1.000000e-07	-7.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	51	5.598171	2.466444	(473.4347848037066, 473.9613500188084, 473.961...
4	1.000000e-09	-9.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...	51	7.593336	2.532935	(473.43478489739937, 473.96135011250124, 473.9...

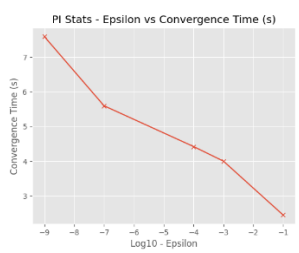
64 States:

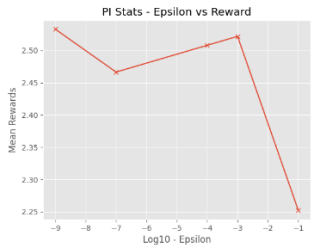
My PI dataframe w.r.t Changing Epsilon

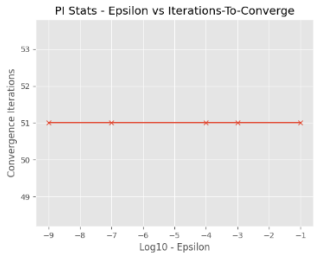
	Epsilon	Log10 Eps		Policy Iterations	Time	Mean Reward	Value Functior
0	1.000000e-01	-1.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	51	0.223173	14.163934	(473.342763392478, 473.8693286075798, 473.8693...
1	1.000000e-03	-3.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	51	0.362184	13.625128	(473.43385555356514, 473.9604207686668, 473.96...
2	1.000000e-04	-4.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	51	0.340792	16.130521	(473.434691643851, 473.96125685895294, 473.961...
3	1.000000e-07	-7.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	51	0.549590	18.852554	(473.4347848037066, 473.9613500188084, 473.961...
4	1.000000e-09	-9.0	(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, ...	51	0.705558	15.347731	(473.43478489739937, 473.96135011250124, 473.9...

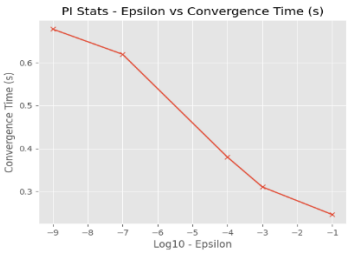
On closer inspection, I noticed that optimum policies as well as number of iterations are same with constant discount factor despite different values of epsilon. Again, this is expected as by design of PI method, we pick one policy and improve that over multiple iterations and in our case, the model converges at 51 iterations.

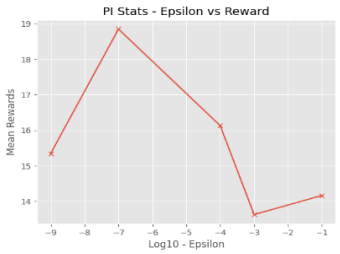
We can pick the best epsilon value for optimum policy to be any of the 5 values I tested, however, keeping it same as in VI, I choose eps = 0.1 (best reward to convergence time ratio) for my analysis later in this report. This is further supported by the below graphs.

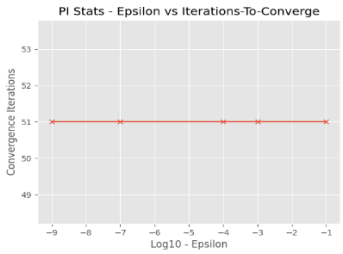












C3. Q-Learning:

This is a Model-Free algorithm as opposed to model-based VI/PI. This is a non- grid world MDP that requires a different set up. For purpose of my study, I explored changing epsilon, epsilon decay rates, learning rate decay,

iterations (4) parameters, keeping discount factor & learning rate constant. I checked the mean rewards, time to converge and so obtained the following dataframe:

Important point to note is that the Optimum Policy for VI = PI but different from all 24 Optimum policies (extracted via different combinations of hyperparameters)

C4. Comparative Analysis: Policy Iteration Vs Value Iteration Vs Q-Learning

Policy (same or not), Iterations to converge, Max Reward, epsilon, discount for both

Considering criteria of maximum reward in least iterations(or time), lets pick epsilon value as 0.1 and compare. We observe that, PI -> took 51 itrations to converge in 2.45 secondes & VI-> took 130 iterations to converge in 0.025 sec.

Iterations to converge for PI is less than that in case of VI AND time taken by PI to converge was way slower than that taken by VI. This is because PI by design takes one policy and aims to improve that over multiple iterations which takes time. VI on the other hand changes the policy itself, iteratively, till it converges to the best policy.

For epsilon = 0.1, the Mean Rewards earned in PI (2.38) was slightly more than the mean rewards in VI (2.25) but way more than mean rewards earned in QL (1.54).

For QL, best policies obtained are all different as opposed to those obtained by VI/PI. Infact, none of the 24 optimum policies match with the policy we obtained from PI/VI(which was same). Another observation is that the time taken by Q-learning is maximum which makes sense because …..

Graph with both plots

C5. Did change of states affect this outcome?

I reduced the number of states by a factor of 1/10 i.e. from 640 to 64 and I observed that:

PI, VI –On reducing states, the time taken to converge to optimum policy reduced i.e. it was faster as expected because of relatively low number of states, the Mean rewards earned increased but the number of iterations to converge remained constant i.e. 51 iterations

QL – At smaller state, same set of hyperparameter combinations results into less mean rewards and faster time, which is expected as the state space is drastically reduced. Also, larger state spaces will require more training compared to smaller ones which further supports my observations.

Conclusion and future plans:

I explored so much in Unsupervised Learning and still it feels there is so much to learn. It was interesting to study different clustering techniques, analyze plots to determine optimum clusters and draw comparisons between different clustering algorithms. Same happened in case of Dimensionality Reduction algorithms. It was interesting to see the impact of reduced dimensionality on our data model. In general, I found that the neural network algorithms with reduced dimensions or when used with clustering were faster. This analysis was based on ExoPlanet dataset which is a binary classification problem. In future, I would like to perform similar analysis on different nature of problems.

References:

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html

https://www.geeksforgeeks.org/ml-v-measure-for-evaluating-clustering-performance/

https://scikit-learn.org/stable/modules/random_projection.html

https://bioturing.medium.com/how-to-read-pca-biplots-and-scree-plots-186246aae063