STATICAL LEARNING

WENYU JIAO, YANG MENG, BORUI FENG
EN.530.641
DUE SATURDAY, DEC. 17, 11:59PM

1. MACHINE LEARNING

1.1. The dataset we choose is called adult data set which we got from this link: https://archive. ics.uci.edu/ml/datasets/Adult. It's a binary classification problem and uses several features to predict people's income. The features are [age', 'workclass', 'fnlwgt', 'education', 'education_num','marital_status', 'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss', 'hours_per_week', 'native_country']. And the target cloumn is 'wage_class' (income >50K or ≤50K)
We have performed data pre-processing including one-hot encoding categorical columns and standardization using StandardScaler in sklearn.
The original data set has splited train and test set for us. However, to meet the requirement of this problem, we combined all the dataset we have and conducted train test split with test size = 0.1 as required.

1.2. The three models we have chosen to use are :

1 Random Forest
2 Logistic Regression
3 Support Vector Machine

1.3. We performed model selection as required using GridSearchCV with CV = 4. The selected hyperparameters for each model are listed below:

1 Random Forest:

```
params = {'max_depth': list(np.arange(3,10)) + [None],
          'min_samples_leaf': [1,2,3],
          'n_estimators': [10,100]}
```

2 Logistic Regression:

```
params = {'C':[0.001, 0.01, 0.1, 1, 10, 100]}
```

3 Support Vector Machine:

```
params = {'kernel':['poly','rbf']
          ,'C': [100,10,1,50]}
```

The best models are listed below:

1 Random Forest with max_depth = None, min_samples_leaf = 3, n_estimators = 100
2 Logistic Regression with C = 0.1
3 Support Vector Machine with kernel = rbf, C = 1

1.4. The model accuracy for the three models are listed below:

```
Random Forest
0.8716479017400205
Logistic Regression
0.8567041965199591
SVM
0.8638689866939611
```

As a result, Random Forest with max_depth = None, min_samples_leaf = 3, n_estimators = 100 is the best model among the three.

2. Dynamic programming

2.1. For the dynamic programming part, we first establish the mathematical equations that describe the model. These equations output angular acceleration and acceleration. Using the obtained angular acceleration and acceleration, we calculate velocity, angular velocity, displacement, and angle. After defining the mathematical model, we assign values to it. For the convenience of calculation, we discretize the model data. The thresholds for discretization are theta: -12, -6, -1, 0, 1, 6, 12; x: -2.4, -0.8, 0.8, 2.4; theta dot: negative infinity, -50, 50, positive infinity; x dot: negative infinity, -0.5, 0.5, positive infinity.

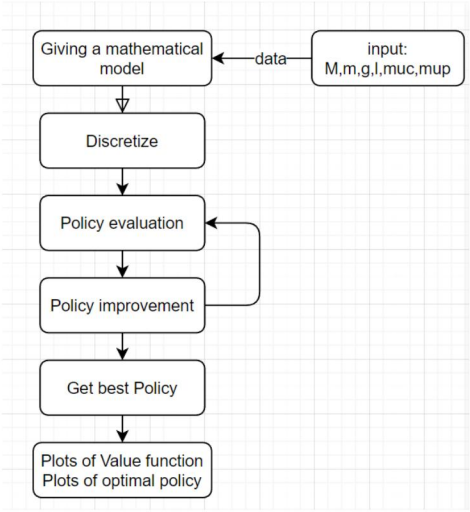
We set boundaries for the problem. If theta is greater than 12 degrees or less than -12 degrees, or if x is greater than 2.4m or less than -2.4m, the step reward for this step is 0. If theta is less than 12 degrees or greater than -12 degrees, and x is less than 2.4m or greater than -2.4m, the reward for this step is 1.

Next, we perform policy evaluation. Policy evaluation uses the algorithm of synchronous iterative joint dynamic programming: starting from any state value function, we update the state value function synchronously and iterative according to a given strategy, the Bellman expectation equation, state transition probability, and reward until it converges to the final state value function under this strategy. The threshold we set is 0.00001.

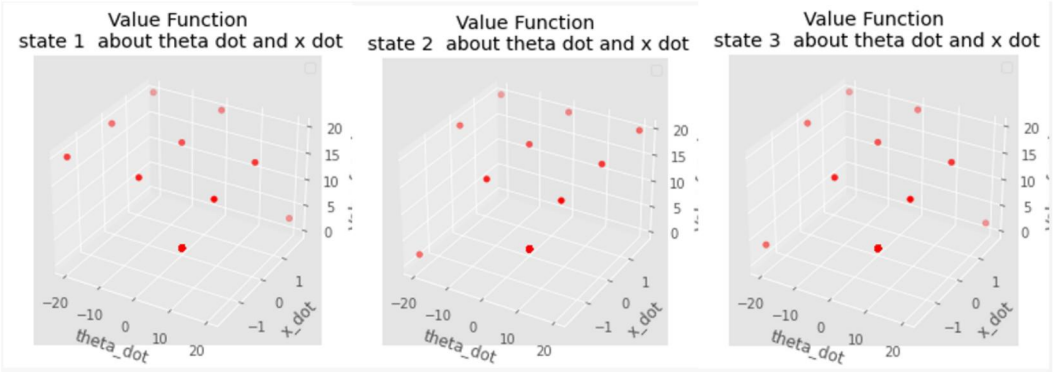
The next step is policy enhancement. The goal of calculating the value function v_π under a certain policy is to find a better $policy_\pi$. Assuming that the value function v_π has been determined for any $policy_\pi$, we evaluate whether for some states it would be beneficial to change the strategy and choose an action $\neq\pi$(s). One way to do this is to choose an action and then follow the existing $policy_\pi$. If q_π(s, a)>v_π(s), that is, choosing action a once in state s and then following strategy π is better than following strategy π all the way, then the new strategy is generally a better strategy.

The last step is to perform policy iteration. The purpose of policy iteration is to make the strategy converge to the optimal solution by iteratively calculating the value function. Essentially, it uses the current strategy to generate new samples, then uses the new samples to better estimate the value of the strategy, and then uses the value of the strategy to update the strategy, repeating this process until convergence.

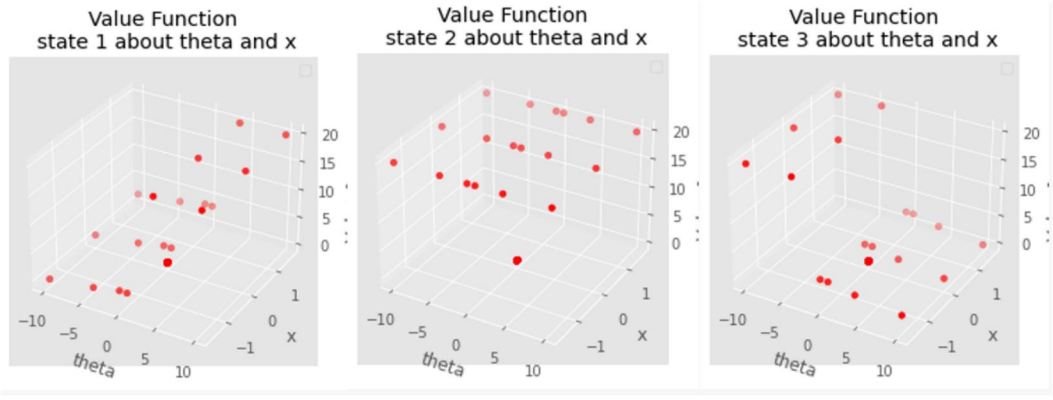
2.2. The flow chart is as follows



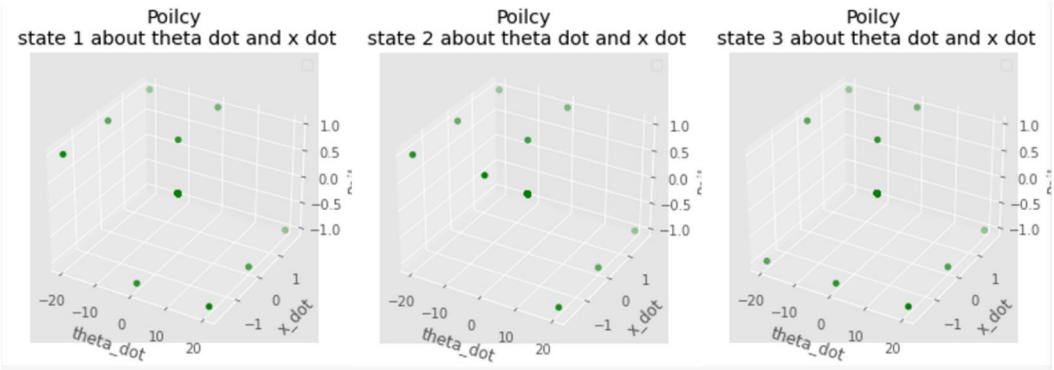
2.3. Plots of the value function for given θ and x



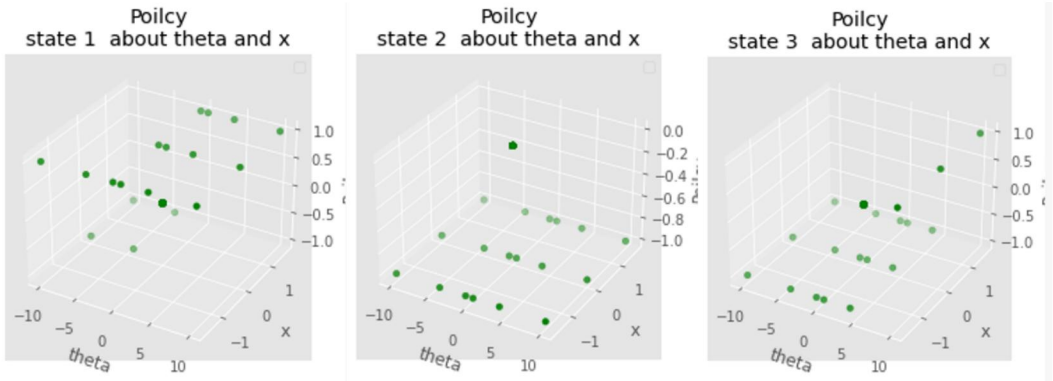
2.4. Plots of the value function for given $\dot{\theta}$ and x



2.5. Plots of the optimal policy for given θ and x



2.6. Plots of the optimal policy for given $\dot{\theta}$ and x



3. Q-LEARNING

The method here is to update the policy with each trial. Recall that a policy is a function

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$$

with

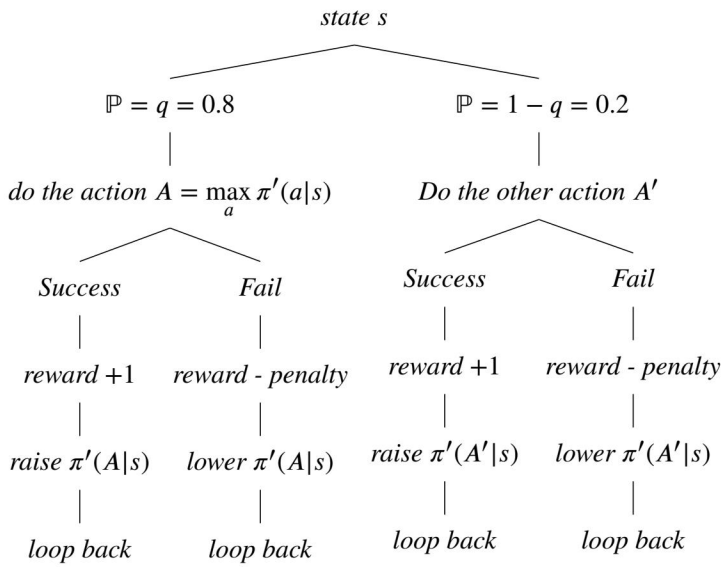
$$\pi(a|s) = \#\text{probability of choosing action a given state s}$$

and the way we represent it here is to define the function π' such that

$$\pi'(a|s) \in \mathbb{R}$$

and with probability $q = 0.8$ (manually chosen) will the system choose the action with the highest π' value, and with probability $(1 - q)$ a random action from the others.

Since we are dealing with a situation where $\mathcal{A} = \{1, -1\}$ which corresponds to moving left and right, our policy-action function is written in this manner:



In this way we let the machine update its policy with its current behavior. Now some of the parameters we can modify are

- $q =$ # probability that the system will execute a seemingly better action. (exploration vs exploitation)
- $\alpha =$ the rate at which the policy is updated, i.e. for each end of trial, how much reward/penalty is assigned to $\pi'(a|s)$.
- The penalty $f(t)$ where t is the total step till failure.
- $\gamma =$discount factor.

After some trial, we note that the best interval for these parameters are:

- $0.8 \leq q \leq 1$;
- $0.09 \leq \alpha \leq 0.2$;
- $0.9 \leq \gamma \leq 1$.

and the penalty function is in the end decided to be

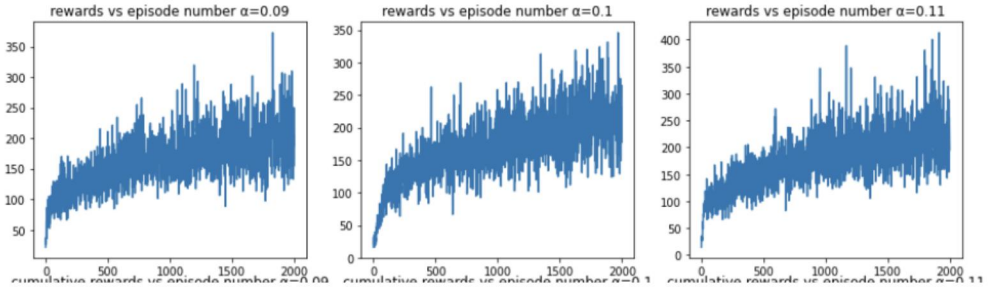
$$f(t) = \begin{cases} -8000 & t < 30 \\ -5000 & 30 \leq t < 60 \\ -3000 & 60 \leq t < 100 \\ -1080 & 100 \leq t < 200 \\ -200 & 200 \leq t < 300 \end{cases}$$

for which we also tried

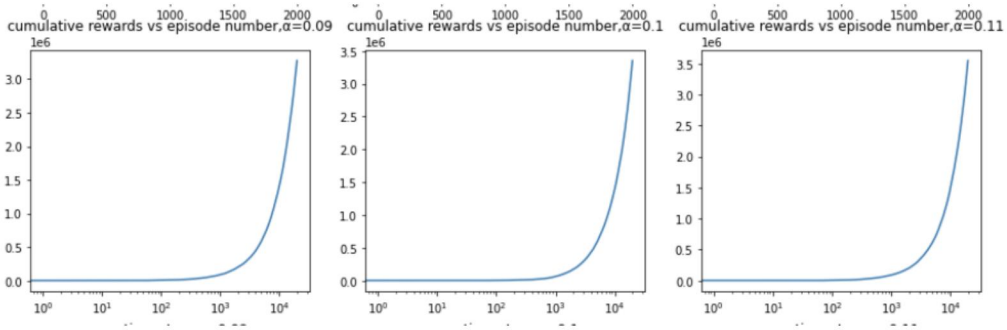
$$f(t) = -ce^{-\frac{1}{k}t} - b$$

for various choice of c, k, and b, but turns out that a "sharper" penalty is helpful for finding better policies.

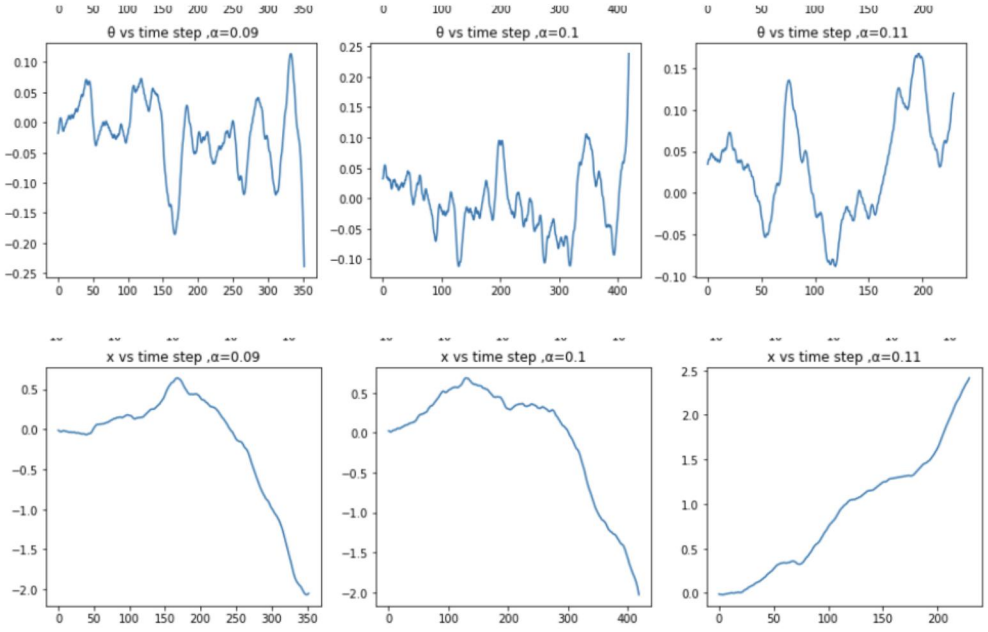
The results are: (for $\alpha = [0.09, 0.1, 0.11]$)



and the cumulative reward is



For more information on the real-life parameters (θ and x), they are:



Workload Distribution: Borui Feng (Machin Learning), Yang Meng (Dynamic Programming), Wenyu Jiao (Temporal difference with Q-learning)