Machine, Data and Learning

Assignment 2- Part 3

Team 66

Linear Programming Problem

Anishka Sachdeva (2018101112)

Satyam Viksit Pansari (2018101088)

Problem Statement :

This assignment is linked to Assignment-2 Part-A python program has to be built for the same scenario. This time, you have to pose the problem as a Linear Programming Problem(LPP) and find the optimal policy by solving the LP. The same scenario is presented here, but there are subtle differences. Students are advised to go over the scenario again.

Libraries Used :

1. Cvxpy

   CVXPY is a Python-embedded modeling language for convex optimization problems. It allows you to express your problem in a natural way that follows the math, rather than in the restrictive standard form required by solvers.

2. Os

The **OS** module in **python** provides functions for interacting with the operating system. **OS**, comes under **Python's** standard utility modules. This module provides a portable way of using operating system dependent functionality.

3. Numpy

   NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Penalty = -20 for our code.

Questions asked in the pdf :

1. **Procedure of making the matrix :**

   Steps to follow:

   - Using the constraint formula from lecture 6 slides:

$$\sum_a x_{ja} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j,$$

$$x_{ia} \geq 0$$

   x_{ia} : Expected number of times action a is taken in state i

   R_{ia} : Reward for taking action a in state i

   p^a_{ij} : Probability of reaching state j when action a is taken in state i

   α_{j} : Initial probability of being in state j

   - By doing simple manipulations:

     δ_{ij}=1 (for i = j)

     δ_{ij}=o otherwise

$$\sum_a x_{ja} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j,$$

$$x_{ia} \geq 0$$

$$= \sum_i \sum_a x_{ia} \delta_{ij} - \sum_i \sum_a x_{ia} p_{ij}^a = \alpha_j$$

$$= \sum_i x_{ia} \sum_a (\delta_{ij} - p_{ij}^a) = \alpha_j$$

- Now this formula resembles the product of two matrices A and X where

  $A_{ik} = \delta_{ij} - paij$ (k = index for state j and action a) and

  X_{ia} is the probability of taking action 'a' in state i.

  Hence, A is the matrix formed.

2. **Procedure for finding the policy**
   - Following the formulation discussed in lecture 6 slides we have to maximize

$$\max \sum_i \sum_a x_{ia} r_{ia}$$

- This becomes the objective function for our Linear programming problem.
- Now this again looks like the dot product of two vectors R and X with the number of elements in
- each vector is the number of valid state-action tuples.

- Now on solving (using cvxpy), we get this vector X.
- Using this vector X, we can determine the policy for a state(s) by choosing that action which has the maximum value of Xsa out of state action pairs (where state = s).
- This chooses action with highest probability as policy for a statei.e. The highest weight in the summation above .

3. **Can there be multiple policies ?**
    - Yes, there can be multiple policies. This is possible when two or more actions have the same probabilities in a state.
    - When such a situation arises, the order in which the actions are listed in our code will be taken into account.
    - In our code, the order of actions is defined as : ACTION_LIST = ["NOOP", "SHOOT", "DODGE", "RECHARGE"].
    - So for example, if in a state, the probabilities of actions Dodge and Recharge are the same, then the action Dodge will be taken over Recharge and hence the policy will be Dodge. If the order had been the other way round, then the policy would have been Recharge.
    - The above example explains how a state can have multiple policies.

      Changes that we can make to generate other policies:

        1. Change the order of actions(so in case of same probability me will get a different policy)
        2. If we encounter equal probability for 2 or more actions for a state-to decide which action to choose as policy for that state ,we can rerun the LP problem by changing the penalty(we can only reduce the penalty of an actions whose probability were equal to ensure the selection of an action among them) of states a little which will result in unequal probability which were earlier equal and now choose the action for the state through these probabilities. Note,for other states nothing changes.

Brief code structure:

Initialize_actions method creates the reward array and action array(which keeps a track of valid actions in each state)

Probability method for a given state ‘s’ and action ‘a’ gives all possible states we can go to from ‘s’ via action ‘a’ along with probability.

Using these we construct the A matrix.

Then we construct alpha manually.

Then we use reward , A matrix and alpha to get objective and X (using cvxpy).

Make_policy method makes policy list using action array and X.

Finally we create the required dictionary and fulfill the required submission format.