Report 31/03 : Reinforcement learning for Cache-Friendly Recommendations

Jade Bonnet and Clément Bernard

Modelisation of our problem

We simplify our problem by taking N = 1, which means that we recommend any of the files in the catalogue.

Agent : Algorithm of recommendation
Action : All the possible files (to recommend).
Environment : Markovian User
State : Song which is currently listened
Reward : 1) If we make constraints on the action,
2) No constraints on the actions, and then reward on :

- Recommended and cached :
- Recommended and not cached
- Not recommended and cached
- Not recommended and not cached

Q-table : KxK table. It gives for a couple (state,action) the quality of an action taken from that state.

Coding

Class Environment :
We created a class which simulates the environment of our problem.
It initializes the different states and actions as well as the reward attribution. For now, the actions implemented are those of markovian user.
The reward attribution chosen is that of Good/Bad reward according to the type of file chosen (recommended, cached, both, neither).
A system with reward and constraints is yet to be implemented.

Q-learning algorithm :
We used a matrix to code the Q-table. Our Q-table has value Q[state, action].
We have two ways to update our model : either by selecting the actions based on the max future reward (**exploiting**) or by acting randomly (**exploring**). It allows to exploit some cases that the agent wouldn't discover by exploiting.

Then, we implemented the algorithm by making a loop “while”. The condition is that the Q-table doesn’t evolve anymore. Here is our function :

```
# Q learning algorithm
while ( (i<max_iter) or (np.sum(np.abs(before_q-q_table)) > epsilon_c)    ):
    # We copy the first step of the table to compare at the end
    before_q = copy.copy(q_table)
    # Initialize the state
    state = env.reset()
    # Initialization of the variables

    epochs, reward = 0, 0
    # The boolean which says whether the process is done or not
    done = False
    while not done :
        if random.uniform(0, 1) < epsilon:
            # Explore action space
            action = env.step_action()
        else :
            # Exploit learned values
            action = np.argmax(q_table[state])
        # Go to the next state
        next_state, reward, done = env.step(action)
        if (next_state is None) :
            # We stop when the user leaves the process
            break
        old_value = q_table[state, action]
        next_max = np.max(q_table[next_state])
        # Q learning algorithm
        new_value = (1 - alpha) * old_value + alpha * (reward + gamma * next_max)
        q_table[state,action] = new_valu
        # Go to the next state
        state = next_state
        epochs +=1

    i+=1
```

Q-learning algorithm coded in Python

Results

Our algorithm iterates well to update the q-table but the result isn’t relevant yet. Here is a result of our Q table (for 1000 actions and 1000 states).

	action_0	action_1	action_10	action_100	action_101	action_102	action_103	action_104	action_105	action_106	...	action_990	action_991	action_992
state_0	4.556046	-5.394444	-4.395445	-5.439044	-4.044395	-5.443945	-4.455394	-5.394544	-4.554444	-4.444394	...	-4.544439	-5.404445	-5.43944
state_1	-5.395544	5.459560	-5.444444	-4.545404	-5.453944	-4.539404	-4.539444	-4.444544	-4.444444	-4.394444	...	-4.454540	-4.444454	-5.43944
state_2	-5.395544	-5.394444	-5.444444	-4.545404	-5.453944	-4.539404	-4.544554	-4.444544	-4.444444	-4.394444	...	-5.444444	-4.444444	-5.43944
state_3	-5.395544	-5.394444	-5.444444	-4.545404	-5.453944	-4.539404	-4.444544	-4.444544	-4.444444	-4.394444	...	-5.394039	-4.444444	-5.43944
state_4	-5.395544	-5.394444	-5.444444	-4.545404	-5.453944	-4.539404	-4.439454	-4.444544	-4.444444	-4.394444	...	-4.445444	-4.444444	-5.43944
state_5	-5.395544	-5.394444	-5.444444	-4.545404	-5.453944	-4.539404	-4.439454	-4.444544	-4.444444	-4.394444	...	-4.445444	-4.444444	-5.43944
state_6	-5.395544	-5.394444	-5.444444	-4.545404	-5.453944	-4.539404	-4.439454	-4.444544	-4.444444	-4.394444	...	-4.445444	-4.444444	-5.43944
state_7	-5.395544	-5.394444	-5.444444	-4.545404	-5.453944	-4.539404	-4.439454	-4.444544	-4.444444	-4.394444	...	-4.445444	-4.444444	-5.43944
state_8	-5.395544	-5.394444	-5.444444	-4.545404	-5.453944	-4.539404	-4.439454	-4.444544	-4.444444	-4.394444	...	-4.445444	-4.444444	-5.43944

Q-table after 20000 iterations

What to do next :

- Improve our algorithm to have more relevant results for the Q-table
- Implement the same algorithm with multiple users

Question :

- We had troubles to see the difference, in the coding part, between the states and the actions. Indeed, we coded both as an array of increasing integers. We don’t really see the differences that will appear when we will implement our problem more in details.