**Machine Learning using Python**

**Project Report – [PROVIDE BETTER TREATMENT USING REINFORCEMENT LEARNING]**

|  |  |
| --- | --- |
| **College Name :** | **Government College of Engineering ,Bodinayakanur** |
| **Team Name :** | **FABRICATORS OF ACCURATE DECISIONS** |

|  |  |
| --- | --- |
| **Team Members:** | **1.SARAVANAN S J**  **2.THAMARAI SELVAN P**  **3.BHARATH KUMAR K**  **4.MUTHURAJ K** |

**.**

**1.Abstract**

This program implements reinforcement learning algorithms, Q-Learning and SARSA, to optimize diabetes treatment decisions. It models a simple environment with two states (Controlled and Uncontrolled) and four actions (Insulin, Oral Medication, Diet Adjustment, Exercise). The Q-table, representing the expected cumulative rewards for state-action pairs, is updated through episodes based on the chosen algorithm. The reward structure reflects the effectiveness of treatments. The program trains the agent to learn optimal treatment strategies and suggests actions based on the learned Q-values. Users can select the algorithm, view the Q-table, and get treatment recommendations.

**2. Model**

**REINFORCEMENT LEARNING:**

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent takes actions, observes the results, and adjusts its behavior to maximize a cumulative reward.

RL is particularly well-suited for problems where a sequence of decisions leads to long-term outcomes, such as playing a game or controlling a robot.

**Key Concepts in Reinforcement Learning**

1. **Agent**:
   * The learner or decision-maker that interacts with the environment. The agent's goal is to learn a policy (a strategy) that maximizes cumulative rewards over time.
2. **Environment**:
   * Everything outside the agent. The environment presents situations (states) to the agent and responds to the agent’s actions with new states and rewards.
3. **State (s)**:
   * A representation of the current situation or configuration of the environment. The state provides all the information the agent needs to make a decision.
4. **Action (a)**:
   * A decision made by the agent, which affects the state of the environment. In each state, the agent selects an action from a set of possible actions.
5. **Reward (R)**:
   * A scalar value given by the environment in response to the agent's action. The reward signals the immediate benefit of an action. The agent's objective is to maximize the total (cumulative) reward.
6. **Policy (π)**:
   * A policy is a mapping from states to actions. It defines the agent's behavior: given a state, what action should the agent take? Policies can be deterministic (always taking a specific action in a state) or stochastic (taking actions with certain probabilities).
7. **Value Function (V)**:
   * The value function estimates how good it is for the agent to be in a certain state, considering the future rewards. The value function is a prediction of future rewards, assuming the agent follows a particular policy.
8. **Q-Value (Q-function)**:
   * The Q-value (or action-value function) estimates the value of taking a particular action in a particular state, considering future rewards. It helps the agent decide the best action to take in a given state.
9. **Exploration vs. Exploitation**:
   * Exploration involves trying new actions to discover their effects, while exploitation involves choosing actions that are already known to yield high rewards. Balancing exploration and exploitation is a key challenge in RL.
10. **Discount Factor (γ)**:
    * A factor between 0 and 1 that determines the importance of future rewards. A higher discount factor means the agent values future rewards more, while a lower discount factor makes the agent more focused on immediate rewards.

There are several types of reinforcement learning,in that there are two main types they are,

1.Model free reinforcement learning

2.Model based reinforcement learning

**MODEL FREE REINFORCEMENT LEARNING**:

Model-free reinforcement learning (RL) refers to a class of RL algorithms where the agent learns to make decisions without an explicit model of the environment. In other words, the agent does not need to know the transition probabilities between states or the reward function in advance; it learns directly from the interaction with the environment.In this there are two types of algorithms,they are,

**1.Q Learning**

Q-Learning is a popular model-free reinforcement learning algorithm used to learn the value of taking certain actions in particular states in order to maximize cumulative rewards over time. It is an off-policy algorithm, meaning it learns the optimal policy independently of the agent's current actions.

**2.SARSA**

SARSA (State-Action-Reward-State-Action) is a model-free, on-policy reinforcement learning algorithm used to find the optimal policy for decision-making problems. Unlike Q-Learning, which is an off-policy method, SARSA updates its value estimates based on the actions actually taken by the agent according to its current policy.

**3. Algorithm Implementation**

Algorithm Implementation Overview

1. Initialization:
   * Q-Table: Initialized with zeros. The Q-table is a 2D array where rows correspond to states and columns correspond to actions. The values represent the expected cumulative rewards for each state-action pair.
   * Hyperparameters:
     + learning\_rate (α\alphaα): Controls how much new information overrides old information.
     + discount\_factor (γ\gammaγ): Represents the importance of future rewards.
     + episodes: Number of training episodes.
     + epsilon: Exploration rate for balancing exploration and exploitation.
2. Reward Structure:
   * A dictionary defines the reward values for each state-action pair. For instance, giving Insulin in a Controlled state yields a reward of 10, while giving Oral Medication in an Uncontrolled state yields a reward of -2.

Transition Function:

* + Simulates the environment’s response to actions and defines how the state changes based on the current state and action taken.

1. Action Selection:
   * Uses an epsilon-greedy policy to choose actions. With probability ϵ\epsilonϵ, a random action is chosen (exploration), and with probability 1−ϵ1 - \epsilon1−ϵ, the action with the highest Q-value is chosen (exploitation).

**SARSA Algorithm Implementation**

**SARSA (State-Action-Reward-State-Action)** is an on-policy algorithm that updates the Q-values based on the action taken in the next state, according to the current policy.

1. **Initialize Q-Table**:
   * Reset the Q-table for each algorithm run.
2. **Episode Loop**:
   * For each episode:
     + **Start State**: Randomly choose an initial state.
     + **Choose Initial Action**: Use the epsilon-greedy policy to select the first action.
3. **Interaction with Environment**:
   * While the episode is ongoing:
     + **Transition**: Determine the next state and reward based on the current state and action using the transition function.
     + **Choose Next Action**: Select the next action in the next state using the epsilon-greedy policy.
     + **Update Q-Value**: *Q*(*s*,*a*)←*Q*(*s*,*a*)+*α*[*r*+*γQ*(*s*’,*a*’)–*Q*(*s*,*a*)]
       - Here, Q(s′,a′)Q(s', a')Q(s′,a′) is the Q-value of the next state-action pair, where a′a'a′ is the next action chosen in the next state.
4. **Transition to Next State**:
   * Move to the next state and update the action to the next action.
   * **Stopping Condition**: To prevent infinite loops, limit the number of steps or use a specific stopping condition.

**Q-Learning Algorithm Implementation**

**Q-Learning** is an off-policy algorithm that updates Q-values based on the maximum possible future reward, regardless of the agent's actions.

1. **Initialize Q-Table**:
   * Reset the Q-table for each algorithm run.
2. **Episode Loop**:
   * For each episode:
     + **Start State**: Randomly choose an initial state.
     + **Choose Initial Action**: Use the epsilon-greedy policy to select the first action.
3. **Interaction with Environment**:
   * While the episode is ongoing:
     + **Transition**: Determine the next state and reward based on the current state and action using the transition function.
     + **Update Q-Value**: *Q*(*s*,*a*)←*Q*(*s*,*a*)+*α*[*r*+*γ*max*a*’​*Q*(*s*’,*a*’)–*Q*(*s*,*a*)]
       - Here, maxa′Q(s′,a′)\max\_{a'} Q(s', a')maxa′​Q(s′,a′) represents the maximum Q-value for the next state s′s's′, representing the best possible future reward.
4. **Transition to Next State**:
   * Move to the next state.
   * **Stopping Condition**: Similar to SARSA, to prevent infinite loops, limit the number of steps or use a specific stopping condition.

**4. Prediction Comparison Report**

The objective of this program is to apply reinforcement learning algorithms (Q-Learning and SARSA) to optimize diabetes treatment decisions. The algorithms aim to determine the best actions (treatment options) for each state (health status of the patient) to maximize cumulative rewards.

**Predictions**

1. **Q-Learning Predictions**:
   * **Expected Convergence**: Q-Learning is expected to converge faster to the optimal policy due to its off-policy nature, which allows it to explore and learn the best actions more aggressively.
   * **Policy Outcome**: The learned policy should favor actions with the highest Q-values, leading to the selection of treatments that provide the highest cumulative rewards over time. For instance, the Q-Learning algorithm might consistently suggest Insulin as the best action for Controlled states due to high rewards.
2. **SARSA Predictions**:
   * **Expected Convergence**: SARSA is expected to converge more slowly compared to Q-Learning, as it updates Q-values based on the actions actually taken, which can be more conservative.
   * **Policy Outcome**: The learned policy will reflect the actions chosen during training, which may be less aggressive. This means SARSA might suggest treatments that are consistent with the actiontaken during training, even if they are not always the optimal choices.

**Comparison**

1. **Convergence Speed**:
   * **Q-Learning**: Typically converges faster because it updates Q-values based on the best possible future rewards. This aggressive approach can quickly lead to the discovery of optimal actions.
   * **SARSA**: Converges more slowly due to its conservative nature of updating Q-values based on the current policy and the actions actually taken.
2. **Policy Quality**:
   * **Q-Learning**: Often results in a policy that maximizes long-term rewards by focusing on the best possible future outcomes. It is likely to identify the most effective treatments more quickly.
   * **SARSA**: Produces a policy that is more cautious and reflects the actual actions taken. This can lead to more stable but potentially suboptimal policies if the current policy is not ideal.
3. **Exploration vs. Exploitation**:
   * **Q-Learning**: More aggressive exploration due to its focus on the maximum future reward, which can sometimes lead to better performance in environments with clear optimal actions.
   * **SARSA**: More cautious exploration as it depends on the current policy, which may result in a more gradual improvement.
4. **Stability**:
   * **Q-Learning**: Can be less stable because it uses the maximum future reward for updates, which might lead to oscillations in the Q-values if not managed carefully.
   * **SARSA**: Generally more stable as it updates Q-values based on actual experiences, leading to smoother policy adjustments.
5. **Final Prediction**

The program predicts that Q-learning will prioritize maximizing long-term rewards and may suggest more aggressive treatments such as **Insulin** for **Controlled** states. However, based on the output, Q-learning's recommendations in this case favored **Exercise**, which may indicate that the specific rewards and state transitions used in this program led to less aggressive treatments. Q-learning tends to explore the action space aggressively, leading to faster optimization. In contrast, SARSA, being an on-policy algorithm, learns based on the actions taken during training, resulting in a more stable and consistent policy. While SARSA is expected to be slower to converge, it ensures reliability and stability in treatment recommendations.

1. **Conclusion**

Q-Learning and SARSA offer distinct approaches for optimizing diabetes treatment. **Q-Learning** maximizes long-term rewards by aggressively exploring actions, often leading to faster optimization. In your program, it favored **Exercise** over more aggressive treatments like **Insulin** for "Controlled" states, based on the reward structure. **SARSA**, learning from actual actions during training, provides a more stable, experience-driven policy. Although generally more conservative, it suggested **Insulin** for "Controlled" patients, indicating that SARSA can also recommend aggressive treatments. Thus, Q-Learning is ideal for quick optimization, while SARSA ensures stability and experience-based decision-making in treatment planning.