SARSA and K-means

Course: Machine Learning
Instructor: Dr. Mirela Popa

Student name: Simon Kohl Student ID: p6242698

Handing in
Upload a single report in form of a PDF. E.g. make a scan. Hand in code in form of a single zip file. Submissions by email or other types of archives are not accepted. Thank you for your understanding.

For the first part (a) include in the report a short description of your result, the best policy and your interpretation of the role of the two parameters alpha and gamma. For the second part (b) include the required explanations.

Filling in
You can use this Word file to answer your questions in a digital form. Alternatively, you can print the document, fill it in, and upload a scan. Make sure that we can read your hand-writing.

Graded: Code and Paper assignment: SARSA

Your task is to implement the SARSA algorithm for a simple single player game, in which an agent explores the environment, collects rewards and eventually arrives in the destination state, finishing the game (e.g. snake game, PacMan). Your goal is to maximize the final score (which is obtained by arriving in the shortest time to the destination state), while also exploring the environment. The grid is 4x4 and the set of valid actions are move up, down, right, left, except for the boundary walls, where only specific actions are possible. All the other values are currently initialized, but you can adjust them as you consider. A part of the code is provided for you in Canvas (tutorial6.ipynb); your task is to complete the missing steps, including the update of the value function.

The algorithm is the following:

For each s, a initialize the state $Q(s,a)$ to zero

Start from a random state s

Do forever:

- Select an action a randomly and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for Q(s,a) as follows
- $$Q(s,a) = (1-\alpha) \cdot Q(s;a) + \alpha \cdot (r + \gamma Q(s',a'))$$
- Make the transition $s \leftarrow s'$
- If s' is the destination state then stop

Include in this report your observations about the process, the obtained Q matrix and your interpretation about the role of the two parameters alpha and gamma and how do they affect the final policy.

First of all, some things ~~had to be~~ were changed:
- the Q-matrix is only 4x4 in the original boiler plate code but I changed it to be 16x4 (state space and action space sizes) to correctly compute the best actions and index them directly ~~to not~~ to get an actually usable policy in the end.

=)

Also I implemented states and actions to be used in the computations by their index in the respective lists (for example: states = [[0,1],...] valid_actions = [[...],[...],[0,1,3,0],[...]] the variables used would then be action = 2 and state = 0 in this case).
I used 200 iterations and an exploration rate of 0.7, since this seemed to work very well.

Best policy for maximizing the score (include it as a matrix/drawing)

[0 0 0 0
-1.103 -1.1103 -1.111 -1
-1.11107 -1.11107 -1.111105 -1.1109
-1.111108 -1.11105 -1.11109 -1.1105
-1 -1.11102 -1.1108 -1.11098
⋮
(see report for full 16×4 matrix)]

0 = up; 1 = down; 2 = right; 3 = left

⇒ [0 3 3 3
0 3 3 1
0 1 2 1
0 2 2 0]

In 4×4 representation, where Q[i][j] is the best action at according to the current policy)

⇒

Explanation of the role of the parameters:

- α (exploration rate) changes how ~~much~~ many random moves the agent makes while learning (ϵ-greedy approach), which leads to the ~~agent~~ agent visiting more states ~~than~~ than when strictly following the policy, therefore giving it a higher ~~than~~ chance ~~to~~ ~~does~~ to discover/explore better solutions, but also introducing more noise to how the agent behaves (less predictible behaviour)
- γ (discount factor) changes how much experiences /rewards that lie further back in time matter. (~~favor~~ usually favours more recent rewards ($\gamma \in [0,1]$)

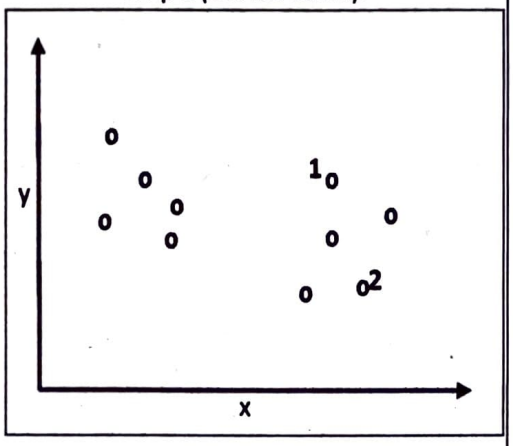
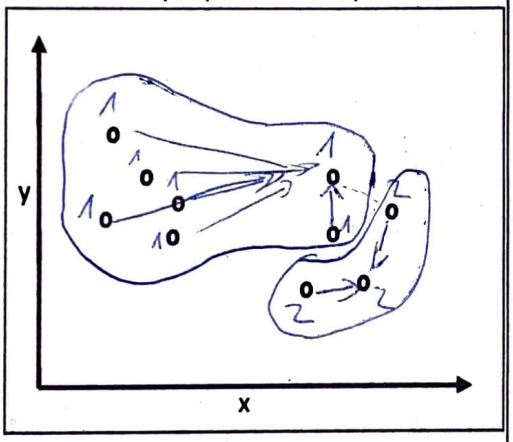
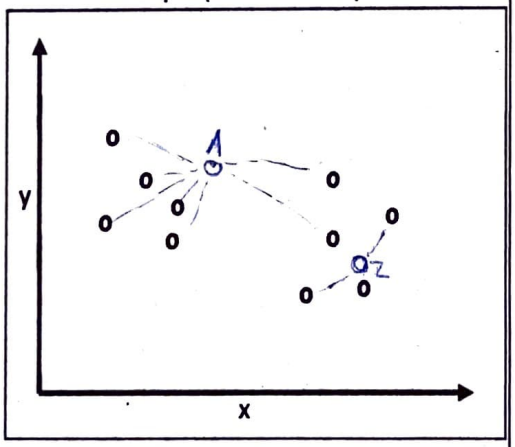
↳ Changing α: possibly suboptimal solutions/ less diversification in solutions

↳ ~~Changing~~ Changing γ: paying more/less "attention" towards non-recent rewards (lower/higher) leads to higher penalization of any ~~reward~~ action, so all rewards would get smaller ⇒ no effect on policy, since all rewards scale the same in this case (every state has the same rewards.

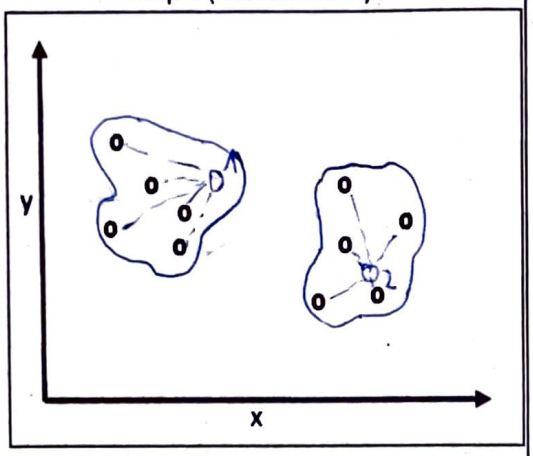
Note: I know it can also be done in a 4×4 matrix, but wouldn't this lead to many unnecessary operations, while, with a 16×4 Q-matrix you could simply look-up the best move?

Graded: Paper assignment: K-Means

Given the following data set, show (with drawings) and explain (with your own words) the different steps of a k-means algorithm when k=2. Show and explain individual steps of the algorithm – not just full iterations.
(Explanation of symbols: o = data points; 1 = marker for first centroid, 2 = marker second centroid)

Step 1 (not iteration!):	Explanation:
	*Initialization: Centroids get assigned to random locations. Here two random points from the data set are picked as initial seeds.*
Step 2 (not iteration!):	**Explanation:**
	assign each data point to the closest centroid (& I drew arrows, but the data ~~points are~~ points are not actually being moved)
Step 3 (not iteration!):	**Explanation:**
	Calculate means of each cluster to ~~find the~~ get the ~~opt~~ new centroids.

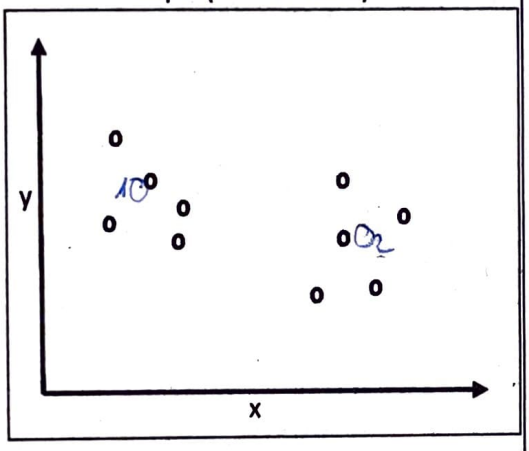
Step 4 (not iteration!):



Explanation:

Assign each data point to the cluster corresponding to the closest centroid.

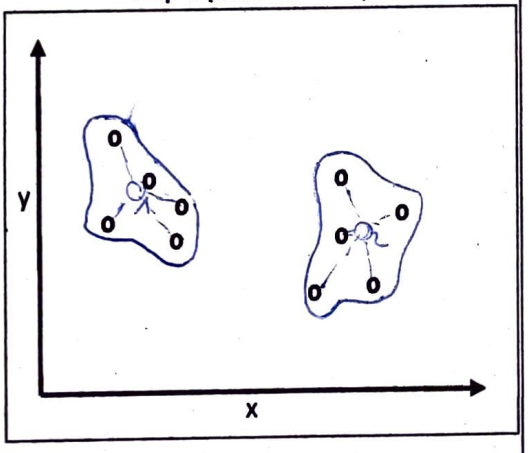
Step 5 (not iteration!):



Explanation:

Calculate new centroids.

Step 6 (not iteration!):



Explanation:

Assign each data point to the cluster corresponding to the closest centroid.