Johanna Posey
1/13/2019

Capstone Project:

Walking Tour

Introduction

Problem

Whether you are on vacation in a new city or exploring your own, it can be challenging to plan out your day's activities. There are so many places to go and so little time. Not to mention, driving and parking can be stressful, time consuming and expensive. Wouldn't it be nice if you could plug your preferences into an algorithm and have a custom list of places to stop? Even better, what if your stops circled back to where you started? That way you could leave your hotel or park your car in a convenient location and when you are done for the day, just walk a short distance back.

Solution

This project utilizes reinforcement learning to create a "walking tour" of an area based on user preferences. The user inputs their starting location, number of places they would like to visit, distance they are willing to travel between stops and venue categories they are interested in. The algorithm takes that information and uses Foursquare data to produce an ordered list of venues for the user. Each stop is within the distance the user is willing to travel from the previous stop and the last stop on the "tour" is within that distance of their starting location, creating a loop.

Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning separate from both supervised and unsupervised learning. In RL, the algorithm interacts with the data and receive feedback through which it learns the optimal way to achieve its goal. RL is commonly used to train computers to play a game, such as chess. As the computer plays, it learns more about the game and what moves result in high rewards in each situation. If we tried to solve this problem without RL, we would have to define every possible series of moves that could result for each configuration and determine if each would lead to winning the game. To see why this is an issue, let's consider just the first turn. There are 20 choices for white's first move and 20 choices for black's first move. Now, each player has only taken one move, but we already have 400 (20*20) possible configurations. The number of possible configurations will grow very quickly making the problem extraordinarily complex.

Johanna Posey
1/13/2019

To explain reinforcement learning, let's start by looking at some of its components.

- Environment: the world or domain of the problem
- Agent: the "actor" or "learner"
- State: where the agent is in the environment
- Action: what the agent does in its state- taking an action results in a new state
- Reward: the feedback the agent receives after taking an action
- Q-value: reward and expected future rewards of taking an action in a state

Take a simple example of a mouse navigating a maze, see Figure 1. The agent is the mouse, the environment is the maze, and the action is moving through the maze. The mouse will eventually find cheese, the exit or a cat. If the mouse finds the exit, it will be rewarded for reaching its goal and it will learn that that path is a good path. If the mouse finds cheese, it will get a lesser reward and continue navigating the maze. If the mouse comes across a cat, the mouse will get a negative reward and learn to avoid that path. Every time the mouse runs through the maze it learns more about the maze until it is trained to take the optimal path.

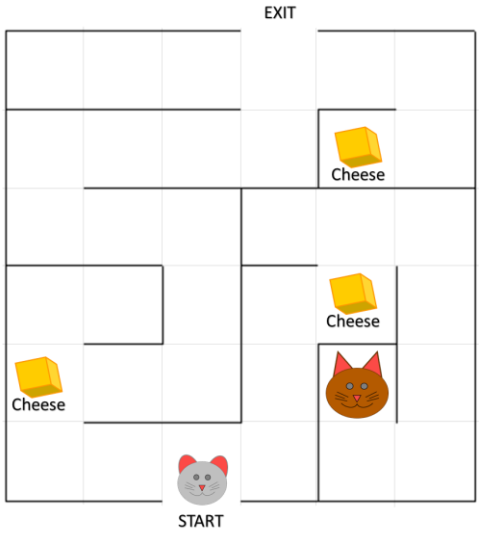

Figure 1: Mouse maze where each square in the grid is a state with a set of actions and rewards associated with that state

Continuing our mouse example, the Q-value is the reward for taking an action in its current state and discounted future rewards the mouse might receive in the new state. The rewards are discounted the further into the future they will occur since we value immediate rewards over anticipated rewards; a bird in the hand is worth two in the bush. Take our mouse at the start of the maze as in Figure 1. If the mouse moves to the square to the left, it can receive cheese in just two actions. If it goes to the right, it must take

three more actions before arriving at cheese. Both lead to cheese and the reward for each is equal, but the value of moving left is higher because it leads to cheese sooner.

In our application, the environment is the search area with all the venues, the agent is the algorithm, the state is the current location and all of the venues on the loop at that given step. The initial state will be the starting location with no venue history. The action is adding a venue to the loop, and the reward for each action is the quality of the loop it produces, which is based on a variety of factors. See the method section for a through explanation of the factors influencing the quality of the loop.

Q-Learning

There are many types of reinforcement learning with various learning mechanisms. The RL technique we will use for our algorithm is q-learning. Equation 1 outlines how learning takes place in q-learning. $Q(s_n,\ a_n)$ is the q-value for the action a_n taken in the state s_n. The state of the agent after taking the action is s_{n+1}. In the state s_{n+1}, the agent may take one of several actions, a, from the set of all possible actions in that state, $a \in A_{s_{n+1}}$. For each action there exists $Q(s_{n+1},\ a_{n+1})$. Q-learning takes r_n, the reward for taking action a_n in state s_n, and the maximum $Q(s_{n+1}, a)$ across all $a \in A_{s_{n+1}}$ and updates $Q(s_n,\ a_n)$.

$$Q(s_n,\ a_n) \leftarrow (1-\alpha) * Q(s_n,\ a_n) + \alpha[r_n + \gamma * \max\big(Q(s_{n+1}, a)\big)] \quad \textbf{(1)}$$

The learning rate, $\alpha \in [0,1)$, determines how quickly the agent learns from new information. As $\alpha \to 1$ the new information overrides the old information, whereas, $\alpha = 0$ means the algorithm doesn't learn any new information from the action. Similarly, $\gamma \in (0,1)$, determines how much to prioritize future rewards. As $\gamma \to 0$, the agent is only interested in instant gratification and doesn't consider future rewards.

Q-learning is an on-policy type of reinforcement learning. Policy is the strategy the agent takes to determine its next action and helps balance between exploration and exploitation. For our mouse, exploration would mean taking a path it hasn't tried yet and exploitation would be sticking to a path it knows produces a positive reward. If the mouse just explores, it will never converge on an optimal path. If it just exploits, it will learn the first path that gets it to the exit and always take that path. An ε-greedy policy can be implemented to ensure the agent exploits its current knowledge while occasionally exploring new actions. This policy works by generating a random number between 0 and 1. Let's take $\varepsilon = 0.1$ for example. If the random number is less than 0.1, then the agent randomly selects an action from the available actions. Otherwise, the agent exploits its knowledge to select the action with the highest estimated value. With $\varepsilon = 0.1$, the agent will exploit 90 percent of the time.

To summarize, let's look at a table of components of q-learning. Table 1 describes each of the components, how they relate to our mouse example and how they relate to the walking tour application.

Table 1: Q-learning Components with Examples

Q-Learning Components with Examples			
Component	**Description**	**Mouse Example**	**Walking Tour Example**
Agent	Entity interacting within the environment	Mouse	Algorithm
Environment	Domain in which the agent operates	Maze	Area around the start where the loop will be constructed and all venues in the selected categories
State	Where the agent is in the environment	Location of the mouse in the maze	All venues on the loop so far and the location of the last venue added to the loop
Action	What the agent does in its state	Move to an adjacent space in the maze	Add a new venue to the loop
Goal	What the agent is trying to accomplish	Exit the maze	Loop back to the start in N steps
Reward	Feedback for taking an action in a state	Lost time, cheese, cat, exit maze	Quality of the loop after adding the next stop
Q-value	Reward and discounted future rewards for a state-action pair	E.g. mouse will receive cheese and can reach the exit in two moves	E.g. the loop contains a venue in a category not previously visited, but will not be able to loop back to the start after the next step
Policy	How the agent will choose an action	Mouse will usually take a path it knows is rewarding, but will occasionally try something new	Algorithm will add a venue it knows will lead to a high-value, but will occasionally try a new venue according to ε-greedy
Learning Rate, alpha	$\alpha \in [0,1)$; weight of new information in learning	How much the mouse learns every time it takes an action	How much the algorithm weights current and future rewards when updating the value of that state-action pair
Discount Factor, gamma	$\gamma \in (0,1)$; weight of future rewards in learning	How will the mouse prioritize future rewards versus current rewards	How much the algorithm weights current rewards over future rewards when updating the value of that state-action pair

Further discussion and exploration of these components in our walking tour application can be found in the method section of this report.

Data

The algorithm will select venues from a list provided by Foursquare. Foursquare stores data for venues which can be accessed using various API endpoints. The SEARCH endpoint returns a list of venues within a given radius of a location that match the given parameters. The response of the SEARCH endpoint provides the id, name, location, and categories for each venue matching the search parameters. A category parameter can be included, which limits the results to the selected categories. Table 2, below, outlines the SEARCH parameters used in this project.

Table 2: SEARCH Endpoint Parameters

SEARCH Endpoint Parameters		
Name	**Description**	**Value**
ll	comma separated latitude and longitude values for the location of the search	The latitude and longitude of the start location
intent	intent of the search	The following intents will be explored to determine the optimal intent. checkin: venues a user is likely to checkin to at the current location and time browse: Finds venues in the area without considering the distance from ll
radius	radius, in meters, from the desired search area	The radius will be calculated using the distance the user is willing to travel and the number of stops they want to make
limit	number of results to return	The maximum number of results Foursquare returns is 50
categoryId	comma separated list of category Ids to search for	The categories the user is interested in

To better understand the Foursquare data and the SEARCH endpoint, we will look at an example. Let's search for venues within 300 meters of the Sheraton Dallas Hotel in Dallas, TX in the category 'Dessert Shop'. We will first look at the results for a search with a checkin intent, then compare those results to the same search with a browse intent. Table 3 outlines the parameters used in our first search. Refer to Table 2 for a description of these parameters.

Table 3: Example SEARCH Endpoint Parameters

Example 1: SEARCH Endpoint Parameters	
Name	**Value**
ll	32.785150, -96.794982
intent	checkin
radius	300
limit	50
categoryId	4bf58dd8d48988d1d0941735

Once we have defined the parameters we can conduct our search. From the response field we will extract relevant information and create a data frame. Table 4 shows us what this data frame will look like for our example. Each column represents a property and each row a venue.

Table 4: Example SEARCH Endpoint Results (intent=checkin)

Example SEARCH Endpoint Results (intent=checkin)						
	Name	**Category**	**Category ID**	**Latitude**	**Longitude**	**Distance from Start**
0	Yumi Yogurt	Ice Cream Shop	4bf58dd8d48988d1d0941735	32.7869	-96.7956	208
1	Chill Frozen Yogurt	Frozen Yogurt Shop	4bf58dd8d48988d1d0941735	32.7854	-96.7953	41
2	Kist Kitchen	Frozen Yogurt Shop	4bf58dd8d48988d1d0941735	32.7822	-96.7973	395

Notice we searched for ‘Dessert Shops’, but the category column doesn’t contain that category name. Foursquare classifies venues according to a hierarchical scheme. When you search using a category id, Foursquare returns all venues within that category and all sub-categories. The first category in the response is the lowest level category ascribed to that venue. This information will give us, and the user, more information about the venues used in the algorithm. The category ID field in Table 3 is the category id that was used in the search rather than the id of the first category returned. This field is to help train the algorithm to create a loop with variety in the categories of the venues visited.

The ‘Distance from Start’ is the distance in meters from the location of the search, in our example, the distance from the hotel. This will be used to train the algorithm to select venues that can loop back to the start. We can notice from this field that Kist Kitchen is 395 meters from the hotel, which is not with in

our radius. This is because the checkin intent does not consider the search radius, only venues a user is likely to checkin to for that location and time.

Now, let us look at the results of the request using the browse intent. In Table 5 we can observe the difference between the two search intents. The browse intent only returns venues within our given radius, making it the better choice for our algorithm.

Table 5: Example SEARCH Endpoint Results (intent=browse)

Example SEARCH Endpoint Results (intent=browse)						
	Name	**Category**	**Category ID**	**Latitude**	**Longitude**	**Distance from Start**
0	Chill Frozen Yogurt	Frozen Yogurt Shop	4bf58dd8d48988d1d0941735	32.7854	-96.7953	41
1	Yumi Yogurt	Ice Cream Shop	4bf58dd8d48988d1d0941735	32.7869	-96.7956	208