revised_submission_q_learning

November 11, 2024

1 Q-Learning on Viral Load Data for Optimal Treatment Strategies

1.1 Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
import random
import itertools
```

[2]:
```
data = pd.read_excel('/Users/hamiddastgir/Desktop/mgddll6.xlsx')
```

[3]:
```
data = data.drop(columns='Unnamed: 0')
```

[4]:
```
data.head()
```

[4]:

	CASEID	VISIT	DNAME	DCLASS	NRTI	NNRTI	PI	OTHER	\
0	1000	40	Atripla	NNRTI NRTI	1	1	0	0	
1	1000	41	Atripla	NNRTI NRTI	1	1	0	0	
2	1000	42	Triumeq	OTHER	0	0	0	1	
3	1003	9	Retrovir Epivir	NRTI NRTI	1	0	0	0	
4	1003	10	Retrovir Epivir	NRTI NRTI	1	0	0	0	

	MISSING	VLOAD	…	LYMPLC	LYMPLC_INC	LPCTLC	LPCTLC_INC	LPMMLC	\
0	0	20.0	…	3	0.0	31.0	0.000000	-1	
1	0	20.0	…	3	0.0	23.0	-0.258065	-1	
2	0	20.0	…	3	0.0	33.0	0.434783	-1	
3	0	8200.0	…	3	0.0	22.2	0.000000	700	
4	0	3200.0	…	3	0.0	32.2	0.450450	1200	

	LPMMLC_INC	POLYLC	POLYLC_INC	PPCTLC	PPCTLC_INC
0	0.000000	3	0.0	58.0	0.000000
1	0.000000	3	0.0	66.0	0.137931
2	0.000000	3	0.0	56.0	-0.151515
3	0.000000	3	0.0	64.9	0.000000
4	0.714286	3	0.0	48.9	-0.246533

```
[5 rows x 51 columns]
```

1.2 Defining State and Action Spaces

[5]:
```
state_columns = ['HGB_LC', 'MCV_LC', 'PLATLC', 'WBC_LC', 'HSRAT']
states = data[state_columns].values
```

[6]:
```
action_columns = ['NRTI', 'NNRTI', 'PI', 'OTHER']
action = data[action_columns].values
```

[7]:
```
scaler = MinMaxScaler()
normalized_states = scaler.fit_transform(states)
```

[8]:
```
action_combinations = list(itertools.product([0, 1],␣
 ↪repeat=len(action_columns)))
```

[9]:
```
action_mapping = {combo: idx for idx, combo in enumerate(action_combinations)}
inverse_action_mapping = {idx: combo for combo, idx in action_mapping.items()}
```

[10]:
```
action_tuples = [tuple(a) for a in action]
action_indices = [action_mapping[a] for a in action_tuples]
```

[11]:
```
num_actions = len(action_combinations)
```

1.3 Discretizing the State Space

[12]:
```
num_bins = 10
state_bins = [np.linspace(0, 1, num_bins + 1) for _ in range(normalized_states.
 ↪shape[1])]
```

[13]:
```
def discretize_state(state):
    discretized_state = []
    for i in range(len(state)):
        discretized_value = np.digitize(state[i], state_bins[i]) - 1
        discretized_state.append(discretized_value)
    return tuple(discretized_state)
```

1.4 Initializing the Q-Table

[14]:
```
state_space = set()
for state in normalized_states:
    state_space.add(discretize_state(state))
state_mapping = {state: idx for idx, state in enumerate(state_space)}
inverse_state_mapping = {idx: state for state, idx in state_mapping.items()}

num_states = len(state_mapping)
q_table = np.zeros((num_states, num_actions))
```

1.5 Setting Q-Learning Parameters

[15]:
```
learning_rate_alpha = 0.1
discount_factor_gamma = 0.5
exploration_rate_epsilon = 0.1
# optional: exploration_rate_epsilon = min_epsilon + (max_epsilon -␣
 ↪min_epsilon) * np.exp(-decay_rate * episode)
```

[16]:
```
VLOAD = data['VLOAD'].values
```

1.6 Implementing the Q-Learning Algorithm

[17]:
```
for episode in range(1000):
    for i in range(len(normalized_states) - 1):
        state = normalized_states[i]
        discretized_state = discretize_state(state)
        state_index = state_mapping[discretized_state]

        if np.random.uniform(0, 1) < exploration_rate_epsilon:
            action_index = np.random.choice(num_actions)
        else:
            action_index = np.argmax(q_table[state_index, :])

        actual_action_index = action_indices[i]

        agent_action_vector = np.array(action_combinations[action_index])
        actual_action_vector = np.
 ↪array(action_combinations[actual_action_index])

        action_similarity = np.sum(agent_action_vector == actual_action_vector)␣
 ↪/ len(action_columns)

        reward = VLOAD[i] - VLOAD[i + 1]

        adjusted_reward = reward * action_similarity

        next_state = normalized_states[i + 1]
        discretized_next_state = discretize_state(next_state)
        next_state_index = state_mapping[discretized_next_state]
        next_max = np.max(q_table[next_state_index, :])

        q_table[state_index, action_index] = (1 - learning_rate_alpha) *␣
 ↪q_table[state_index, action_index] + \
                                             learning_rate_alpha *␣
 ↪(adjusted_reward + discount_factor_gamma * next_max)
```

1.7 Creating DataFrames and Saving Results

[18]:
```
q_table_df = pd.DataFrame(q_table, columns=[str(combo) for combo in␣
 ↪action_combinations])
q_table_df
```

[18]:

	(0, 0, 0, 0)	(0, 0, 0, 1)	(0, 0, 1, 0)	(0, 0, 1, 1)	(0, 1, 0, 0)	\
0	69.634334	58.043931	102.175767	94.646888	9.037365	
1	263.279839	277.466025	419.842723	260.095815	607.234597	
2	-7734.562584	3404.775431	-8006.327947	-4153.504642	-4605.789979	
3	20862.211110	24836.079175	25552.660389	22078.935838	7662.993823	
4	9889.184049	-58626.240855	-9539.181734	-6093.851737	-11214.149192	
..	…	…	…	…	…	
280	917.656118	1669.448479	845.843631	3114.733673	5177.531320	
281	-917.453615	-379.890305	-396.433663	136.309613	-1639.619523	
282	-1354.052400	-384.191501	-588.766254	-526.364593	-570.814610	
283	-129.028473	47.351375	-175.747250	-28.049956	37.010977	
284	335.168847	151.964422	185.943965	173.480070	189.928580	

	(0, 1, 0, 1)	(0, 1, 1, 0)	(0, 1, 1, 1)	(1, 0, 0, 0)	(1, 0, 0, 1)	\
0	16.543258	69.643344	47.528064	181.116367	64.071436	
1	274.946304	269.090210	224.038983	250.627027	264.932540	
2	-4937.870687	-4396.329168	-12397.463563	-11718.585998	-4762.859195	
3	9365.134870	21752.018974	1813.115387	-12583.415776	8030.017406	
4	-5968.770155	-7921.884869	-12625.374832	-5556.273060	-21053.399164	
..	…	…	…	…	…	
280	1846.143125	-6347.621435	1506.972485	1185.285034	-2656.070849	
281	-1100.933124	-352.703999	-628.557844	-272.813014	-421.808592	
282	-341.329358	3702.616943	-389.760018	-1161.085355	-2806.834219	
283	218.537955	42.008808	49.064695	2193.305316	54.927832	
284	108.456918	122.524434	56.045874	136.222487	158.204773	

	(1, 0, 1, 0)	(1, 0, 1, 1)	(1, 1, 0, 0)	(1, 1, 0, 1)	(1, 1, 1, 0)	\
0	147.966045	35.471849	85.917438	39.954635	121.153967	
1	254.199115	310.865897	290.954647	303.525054	50.074939	
2	-4630.450852	-11695.482990	-5600.824425	-8951.515746	-12290.964189	
3	-5006.865890	25741.650393	9174.212216	-21047.936557	-12874.564372	
4	-40409.088841	-20107.165681	-15232.251940	-9040.344493	-17376.820240	
..	…	…	…	…	…	
280	-450.743323	1856.697386	152.888464	-5406.024944	-36336.396487	
281	-135.138036	-285.782453	-1175.216524	-426.614180	-993.860506	
282	-690.940710	-1427.383014	-1889.837878	-1403.117139	-1296.746486	
283	29.474240	30.202012	-71.615575	117.605229	38.172827	
284	212.309424	146.584003	107.810551	140.823089	179.208479	

	(1, 1, 1, 1)
0	37.033890

```
1      144.382188
2    -4819.174043
3     -387.307192
4   -40158.239140
..            …
280 -26228.901328
281  -1218.098167
282  -3523.497773
283     62.356924
284     50.254248

[285 rows x 16 columns]
```

[20]:
```
state_index_to_normalized_state = {}
for state in normalized_states:
    discretized_state = discretize_state(state)
    state_index = state_mapping[discretized_state]
    state_index_to_normalized_state[state_index] = state

normalized_states_list = [state_index_to_normalized_state[idx] for idx in␣
 ↪state_indices]

normalized_state_df = pd.DataFrame(normalized_states_list, columns=[col +␣
 ↪'_norm' for col in state_columns])

full_df = pd.concat([state_df, normalized_state_df, q_table_df], axis=1)

full_df.to_csv('q_table_with_og_states.csv', index=False)
```

[ ]:

[ ]: