k214577-lab8

April 25, 2024

[4]:

```python
#TASK NUMBER 1
from itertools import product

# Define the sample space for four children, each can be a boy (B) or a girl (G)
sample_space = list(product("BG", repeat=4))

# Count outcomes with exactly two boys
count_2_boys = sum(1 for outcome in sample_space if outcome.count('B') == 2)

# Calculate probability of having exactly two boys
probability_2_boys = count_2_boys / len(sample_space)

print(sample_space)
print("Total number of outcomes:", len(sample_space))
print("Number of outcomes with 2 boys:", count_2_boys)
print("Probability of having exactly two boys:", probability_2_boys)
```

```
[('B', 'B', 'B', 'B'), ('B', 'B', 'B', 'G'), ('B', 'B', 'G', 'B'), ('B', 'B',
'G', 'G'), ('B', 'G', 'B', 'B'), ('B', 'G', 'B', 'G'), ('B', 'G', 'G', 'B'),
('B', 'G', 'G', 'G'), ('G', 'B', 'B', 'B'), ('G', 'B', 'B', 'G'), ('G', 'B',
'G', 'B'), ('G', 'B', 'G', 'G'), ('G', 'G', 'B', 'B'), ('G', 'G', 'B', 'G'),
('G', 'G', 'G', 'B'), ('G', 'G', 'G', 'G')]
Total number of outcomes: 16
Number of outcomes with 2 boys: 6
Probability of having exactly two boys: 0.375
```

[5]:

```python
#TASK NUMBER 2
# Define the sample space for a six-sided die
sample_space = [1, 2, 3, 4, 5, 6]

# Define the event E: getting a number less than 4
event_E = [1, 2, 3]

# Calculate probability of event E
prob_E = len([outcome for outcome in sample_space if outcome in event_E]) /␣
↪len(sample_space)

print("Probability of rolling a number less than 4:", prob_E)
```

```
Probability of rolling a number less than 4: 0.5
```

[8]:
```python
#TASK NUMBER 3
red_marbles = 10
blue_marbles = 20
total_marbles = red_marbles + blue_marbles
prob_blue = blue_marbles / total_marbles
prob_red = red_marbles / total_marbles
prob_red_given_blue = prob_red / prob_blue
print("Probability of drawing a red marble given that it is blue:",␣
↪prob_red_given_blue)
```

```
Probability of drawing a red marble given that it is blue: 0.5
```

[9]:
```python
#TASK NUMBER 4
import numpy as np

# Define states and observations
states = ['healthy', 'sick']
observations = ['cough', 'no cough']

# Define transition and emission probabilities
transition_probabilities = np.array([[0.7, 0.3], [0.4, 0.6]])
emission_probabilities = np.array([[0.1, 0.9], [0.8, 0.2]])
initial_state_probabilities = np.array([0.5, 0.5])

def viterbi(obs, states, start_p, trans_p, emit_p):
    V = [{}]
    path = {}

    # Initialize the path and probability for each state based on the first␣
↪observation
    for state in states:
        V[0][state] = start_p[states.index(state)] * emit_p[states.
↪index(state)][observations.index(obs[0])]
        path[state] = [state]

    # Build the Viterbi graph
    for t in range(1, len(obs)):
        V.append({})
        newpath = {}

        for cur_state in states:
            max_prob, max_state = max((V[t-1][prev_state] * trans_p[states.
↪index(prev_state)][states.index(cur_state)] * emit_p[states.
↪index(cur_state)][observations.index(obs[t])], prev_state) for prev_state in␣
↪states)
```

```
            V[t][cur_state] = max_prob
            newpath[cur_state] = path[max_state] + [cur_state]

        path = newpath

    # Find the most likely final state and path
    max_prob, max_state = max((V[len(obs) - 1][state], state) for state in␣
 ↪states)
    return max_prob, path[max_state]

# Define observation sequence and perform Viterbi algorithm
observation_sequence = ['cough', 'no cough', 'cough']
probability, most_likely_states = viterbi(observation_sequence, states,␣
 ↪initial_state_probabilities, transition_probabilities,␣
 ↪emission_probabilities)

print("Most likely sequence of states:", most_likely_states)
print("Probability of the sequence:", probability)
```

```
Most likely sequence of states: ['sick', 'healthy', 'sick']
Probability of the sequence: 0.03456000000000001
```

[11]:

```
#TASK NUMBER 5
# Define the probabilities of selecting each plan
prob_P1 = 0.30
prob_P2 = 0.20
prob_P3 = 0.50

# Define the probabilities of a defect given each plan
prob_defect_given_P1 = 0.01
prob_defect_given_P2 = 0.03
prob_defect_given_P3 = 0.02

# Calculate the total probability of a defect occurring
prob_defect = (prob_P1 * prob_defect_given_P1 +
               prob_P2 * prob_defect_given_P2 +
               prob_P3 * prob_defect_given_P3)

# Calculate the posterior probabilities of each plan given a defect
posterior_P1 = (prob_P1 * prob_defect_given_P1) / prob_defect
posterior_P2 = (prob_P2 * prob_defect_given_P2) / prob_defect
posterior_P3 = (prob_P3 * prob_defect_given_P3) / prob_defect

# Determine the most likely plan responsible for a defective product
plans = [(posterior_P1, "Plan 1"), (posterior_P2, "Plan 2"), (posterior_P3,␣
 ↪"Plan 3")]
most_likely_plan = max(plans, key=lambda x: x[0])  # Use a lambda for clarity
```

```
# Output the result
print("Most likely plan responsible for the defective product:",␣
    ↪most_likely_plan[1])
```

```
Most likely plan responsible for the defective product: Plan 3
```

[10]:
```
#TASK NUMBER 6
# Define the three boxes and their content
box1 = {'gold': 2, 'silver': 0}
box2 = {'gold': 0, 'silver': 2}
box3 = {'gold': 1, 'silver': 1}

# Define the probabilities of picking each box
p_box = [1/3, 1/3, 1/3]  # Uniform distribution for box selection

# Calculate the probability of picking a gold coin
p_gold = sum(box['gold'] / sum(box.values()) * p for box, p in zip([box1, box2,␣
    ↪box3], p_box))

# Calculate the conditional probabilities for the other coin being gold
p_other_gold_given_box = [1 if 'gold' in box and box['gold'] == 2 else 0 for␣
    ↪box in [box1, box2, box3]]

print("Probability of picking a gold coin:", p_gold)
print("Probabilities that the other coin is gold given the box selection:",␣
    ↪p_other_gold_given_box)
```

```
Probability of picking a gold coin: 0.5
Probabilities that the other coin is gold given the box selection: [1, 0, 0]
```