k213238-lab-9

April 25, 2024

[3]:
```python
#TASK 1
import itertools
def checkTwoBoys(combines):
  boys = 0
  for i in combines:
    if i == 'B':
      boys += 1
  if boys == 2:
    return True
  return False
numberOfChildren = 4
numberOfBoys = int(input("Enter the number of boys: "))
numberOfGirls = int(input("Enter the number of girls: "))
data = []
for i in range(numberOfBoys):
  data.append('B')
for i in range(numberOfGirls):
  data.append('G')
combinations = list(itertools.product(data, repeat=len(data)))
combinations = set(combinations)
combinations = list(combinations)
result = 0

for comb in combinations:
    if checkTwoBoys(comb):
      result += 1
print("Probability of Two Boys = " + str((result/len(combinations)* 100)) + "%")
```

```
Enter the number of boys: 2
Enter the number of girls: 2
Probability of Two Boys = 37.5%
```

[2]:
```python
#TASK 2
# Sample Space
die_outcomes = 6

# Probabilities of getting an odd and even number
prob_odd = 1 / 6
```

```
prob_even = 2 / 6

# Event E includes outcomes 1, 2, and 3
prob_E = prob_odd + prob_even + prob_odd

# Print probability rounded to two decimal places
print("Probability of event E:", round(prob_E,2))
# Probability Percent Code
probability_percent = prob_E * 100
# Print probability percent rounded to one decimal place
print(str(round(probability_percent, 0)) + '%')
```

```
Probability of event E: 0.67
67.0%
```

[4]:
```
#Task 3
import numpy as np
import pandas as pd

def event_probability(event_outcomes, sample_space):
    probability = (event_outcomes / sample_space) * 100
    return round(probability, 1)

# Sample Space
total_marbles = 30

# Determine the probability of drawing a blue marble
blue_marbles = 20
blue_probability = event_probability(blue_marbles, total_marbles)

# Determine the probability of drawing a red marble given that it is blue
red_marbles_given_blue = 10  # Since there are 10 red marbles
red_probability_given_blue = event_probability(red_marbles_given_blue,␣
 ↪blue_marbles)

# Print each probability
print("Probability of drawing a blue marble:", blue_probability, "%")
print("Probability of drawing a red marble given that it is blue:",␣
 ↪red_probability_given_blue, "%")
```

```
Probability of drawing a blue marble: 66.7 %
Probability of drawing a red marble given that it is blue: 50.0 %
```

[11]:
```
!pip install hmmlearn
```

```
Requirement already satisfied: hmmlearn in /usr/local/lib/python3.10/dist-
packages (0.3.2)
Requirement already satisfied: numpy>=1.10 in /usr/local/lib/python3.10/dist-
```

```
packages (from hmmlearn) (1.25.2)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in
/usr/local/lib/python3.10/dist-packages (from hmmlearn) (1.2.2)
Requirement already satisfied: scipy>=0.19 in /usr/local/lib/python3.10/dist-
packages (from hmmlearn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (1.4.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-
learn!=0.22.0,>=0.16->hmmlearn) (3.4.0)
```

[54]:
```
#Task 4
from hmmlearn import hmm

#define states
states = ["healthy","sick"]
n_states = len(states)
observations = ["cough","no cough"]
n_observations = len(observations)

#define transition tables
state_prob = np.array([0.7,0.3])
trans_prob = np.array([[0.8,0.2],[0.4,0.6]])
emission_prob = np.array([[0.1,0.9],[0.9,0.1]])

#create model
model = hmm.CategoricalHMM(n_components=n_states)
model.startprob_ = state_prob
model.transmat_ = trans_prob
model.emissionprob_ = emission_prob

#predict
observ_seq = np.array([0,0,1,1,1,0,1]).reshape(-1,1)
log_probability, hidden_states = model.decode(observ_seq, lengths␣
 ↪=len(observ_seq), algorithm ='viterbi' )
print('Log Probability :',log_probability)
print("Most likely hidden states:", hidden_states)
```

```
Log Probability : -6.3406285165075404
Most likely hidden states: [1 1 0 0 0 1 0]
```

[13]:
```
#Task 5
# Define probabilities
P_P1 = 0.30
P_P2 = 0.20
P_P3 = 0.50
```

```
P_D_given_P1 = 0.01
P_D_given_P2 = 0.03
P_D_given_P3 = 0.02

# Calculate P(D)
P_D = P_D_given_P1 * P_P1 + P_D_given_P2 * P_P2 + P_D_given_P3 * P_P3

# Calculate P(Pj|D) for each plan
P_P1_given_D = (P_D_given_P1 * P_P1) / P_D
P_P2_given_D = (P_D_given_P2 * P_P2) / P_D
P_P3_given_D = (P_D_given_P3 * P_P3) / P_D

# Print results
print("Probability of a defective product overall (P(D)): ", P_D)
print("Probability of using Plan 1 given a defective product (P(P1|D)): ",␣
 ↪P_P1_given_D)
print("Probability of using Plan 2 given a defective product (P(P2|D)): ",␣
 ↪P_P2_given_D)
print("Probability of using Plan 3 given a defective product (P(P3|D)): ",␣
 ↪P_P3_given_D)
```

```
Probability of a defective product overall (P(D)):  0.019000000000000003
Probability of using Plan 1 given a defective product (P(P1|D)):
0.15789473684210525
Probability of using Plan 2 given a defective product (P(P2|D)):
0.3157894736842105
Probability of using Plan 3 given a defective product (P(P3|D)):
0.5263157894736842
```

[31]:
```
!pip install pomegranate==v0.14.9
```

```
Requirement already satisfied: pomegranate==v0.14.9 in
/usr/local/lib/python3.10/dist-packages (0.14.9)
Requirement already satisfied: cython<3.0.0,>=0.22.1 in
/usr/local/lib/python3.10/dist-packages (from pomegranate==v0.14.9) (0.29.37)
Requirement already satisfied: numpy>=1.20.0 in /usr/local/lib/python3.10/dist-
packages (from pomegranate==v0.14.9) (1.25.2)
Requirement already satisfied: joblib>=0.9.0b4 in
/usr/local/lib/python3.10/dist-packages (from pomegranate==v0.14.9) (1.4.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.10/dist-
packages (from pomegranate==v0.14.9) (3.3)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.10/dist-
packages (from pomegranate==v0.14.9) (1.11.4)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages
(from pomegranate==v0.14.9) (6.0.1)
```

[56]:
```
#Task 6
from pomegranate import *

# Define the distributions for each box
box1 = DiscreteDistribution({'gold': 2, 'silver': 0})
box2 = DiscreteDistribution({'gold': 0, 'silver': 2})
box3 = DiscreteDistribution({'gold': 1, 'silver': 1})

# Define the states representing each box
s1 = State(box1, name="Box 1")
s2 = State(box2, name="Box 2")
s3 = State(box3, name="Box 3")

# Create the Bayesian network
network = BayesianNetwork("Boxes and Coins")
network.add_states(s1, s2, s3)  # Add states to the network

# Define the edges
network.add_edge(s1, s1)
network.add_edge(s1, s2)
network.add_edge(s1, s3)
network.add_edge(s2, s1)
network.add_edge(s2, s2)
network.add_edge(s2, s3)
network.add_edge(s3, s1)
network.add_edge(s3, s2)
network.add_edge(s3, s3)

# Bake the network
network.bake()

# Calculate the conditional probability for each box
for state in [s1, s2, s3]:
    prob_gold_given_gold_drawn = network.predict_proba({'Box 1': 'gold'})[0].
 ↪parameters[0]['gold']
    print("Probability of the other coin being gold in␣
 ↪",prob_gold_given_gold_drawn)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-56-e42312278897> in <cell line: 32>()
     31 # Calculate the conditional probability for each box
     32 for state in [s1, s2, s3]:
---> 33     prob_gold_given_gold_drawn = network.predict_proba({'Box 1':␣
 ↪'gold'})[0].parameters[0]['gold']
     34     print("Probability of the other coin being gold in␣
 ↪",prob_gold_given_gold_drawn)
```

```
/usr/local/lib/python3.10/dist-packages/pomegranate/BayesianNetwork.pyx in␣
↪pomegranate.BayesianNetwork.BayesianNetwork.predict_proba()

/usr/local/lib/python3.10/dist-packages/pomegranate/FactorGraph.pyx in␣
↪pomegranate.FactorGraph.FactorGraph.predict_proba()

/usr/local/lib/python3.10/dist-packages/pomegranate/distributions/
↪DiscreteDistribution.pyx in pomegranate.distributions.DiscreteDistribution.
↪DiscreteDistribution.__mul__()

AttributeError: 'NoneType' object has no attribute 'keys'
```