MoMeDa_2

May 16, 2024

1 Moderne Methoden der Datenanalyse SS2024

2 Practical Exercise 2

2.1 Exercise 2.1 (Voluntary)

"Should I carry an umbrella or should I risk to get wet?" A possible answer to this question is to look at the weather forecast. But as we all know it is not always reliable. Let's assume that if it will rain, the forecast predicts this correctly in 80 % of the cases. If it will not rain, the forecast is assumed to be accurate in 90 % of the cases. In Sun-City the a-priori rain probability is only 5 %, in Equal-City it's 50 % and in Rain-City it's 95 %. Calculate (on a sheet of paper or with a short program) the four probabilities that it will (not) rain if (no) rain is predicted for the three cities.

There are two different risks of a wrong decision: - Carry an umbrella, but it does not rain. - Don't carry an umbrella in case it rains.

Which are the three possible strategies and which of them is the optimal one to minimize the risk of a wrong decision in each of the three cities? Calculate and compare the risk for each of the three possible strategies. Determine the optimal strategy when the second risk is considered 10 or 100 times more serious.

[36]:
```
def weather(iCity):
    # A: it rains     B: forecasts predict rain

    pRainPredicted_if_Rain = 0.8          # P(B|A)
    pRainNotPredicted_if_NotRain = 0.9    # P(not B| not A)

    pRain = [0.05, 0.5, 0.95]             # prior P(A)
    cityName = ["Sun-City", "Equal-City", "Rain-City"]

    # P(not B|A) = 1 - P(B|A)
    pRainNotPredicted_if_Rain = 1. - pRainPredicted_if_Rain
    # P(B|not A) = 1 - P(not B|not A)
    pRainPredicted_if_NotRain = 1. - pRainNotPredicted_if_NotRain
    # P(not A) = 1 - P(A)
    pNotRain = 1. - pRain[iCity]


    # P(A and B) = P(A) * P(B|A)
```

```
    pRain_and_RainPredicted = pRain[iCity] * pRainPredicted_if_Rain
    # P(not A and B) = P(not A) * P(B|not A)
    pNotRain_and_RainPredicted = pNotRain * pRainPredicted_if_NotRain
    # P(A and not B) = P(A) * P(not B|A)
    pRain_and_RainNotPredicted = pRain[iCity] * pRainNotPredicted_if_Rain


    # P(B) = P(A and B) + P(not A and B)
    pRainPredicted = pRain_and_RainPredicted + pNotRain_and_RainPredicted
    # P(not B) = 1 - P(B)
    pRainNotPredicted = 1. - pRainPredicted

    # TODO : Find all the four probability combinations using Bayes Theorem and print the results for each city

    # TODO : Evaluate the risk assesment
    # Risk1: carry an umbrella but it does not rain
    # Risk2: not carry an umbrella and it rains
    # Strategies: A. always carry an umbrella , B. look at forecasts, C. never carry an umbrella

    print('\n')
    print('Strategy A: always carry an umbrella with you')
    risk1_always = pNotRain;
    risk2_always = 0.
    print('   Risk 1: you carry an umbrella but it does not rain  p = {0:.1f} %'.format(risk1_always * 100))
    print('   Risk 2: you don\'t carry an umbrella and it rains   p = {0:.1f} %'.format(risk2_always * 100))

    # TODO: Determine the optimal strategy when Risk2 is considered 10 or 100 times more serious.

    # TODO: Evaluate the results for each city
```

```
The history saving thread hit an unexpected error (OperationalError('attempt to write a readonly database')).History will not be written to the database.
```

2.2 Exercise 2.2 (Obligatory)

Test your calculations for the previous exercise 'experimentally' by writing a Monte Carlo simulation: Simulate N weather events, generate uniformly distributed random numbers between 0 and 1 for the rain forecast and compare them to the corresponding probability given in the above exercise. Make sure that you only use the values given in the text in your code. Then count the number of events in each category (rain and no rain predicted). Finally, use these numbers to determine the fraction of wrong weather forecasts and compare this number to the probability calculated before.

Repeat the simulation for different values of N.

Use the any of the random number generators introduced in the first exercise sheet to setup this Monte Carlo experiment

[32]:
```
import random
import numpy as np
#from ROOT import gRandom

def weatherMC(iCity, nDays):

    # A: it rains     B: forecasts predict rain

    pRainPredicted_if_Rain = 0.8          # P(B|A)
    pRainNotPredicted_if_NotRain = 0.9    # P(not B| not A)

    pRain = [0.05, 0.5, 0.95]             # prior P(A)
    cityName = ["Sun-City", "Equal-City", "Rain-City"]

    # Counter
    countRain = 0
    countNotRain = 0
    countRainPredicted = 0
    countRainNotPredicted = 0
    countRain_RainPredicted = 0
    countRain_RainNotPredicted = 0
    countNotRain_RainPredicted = 0
    countNotRain_RainNotPredicted = 0

    #gRandom.SetSeed()

    rainy = np.random.rand(nDays) < pRain[iCity]
    r2 = np.random.rand(nDays)
    pred = (r2 < pRainPredicted_if_Rain)*rainy + (r2 >␣
 ↪pRainNotPredicted_if_NotRain)*(1-rainy)

    countRain = np.sum(rainy)
    countNotRain = np.sum(1-rainy)
    countRainPredicted = np.sum(pred)
    countRainNotPredicted = np.sum(1-pred)
    countRain_RainPredicted = np.sum(rainy*pred)
    countRain_RainNotPredicted = np.sum(rainy*(1-pred))
    countNotRain_RainPredicted = np.sum((1-rainy)*pred)
    countNotRain_RainNotPredicted = np.sum((1-rainy)*(1-pred))

    #Print results
    print(f"P(rain)={countRain/nDays}, P(¬ rain)={countNotRain/nDays}") #
```

```
    print(f"P(rainPred)={countRainPredicted/nDays},␣
↪P(¬rainPred)={countRainNotPredicted/nDays}")
    print(f"P(rain   rainPred)={countRain_RainPredicted/nDays}, P(rain ␣
↪¬rainPred)={countRain_RainNotPredicted/nDays}, P(¬rain ␣
↪rainPred)={countNotRain_RainPredicted/nDays}, P(¬rain ␣
↪¬rainPred)={countNotRain_RainNotPredicted/nDays}")

    print(f"P( (rain   ¬rainPred) + (¬rain   rainPred) )=␣
↪{(countRain_RainNotPredicted+countNotRain_RainPredicted)/nDays}")
    print("")
   print(f"P_rainpred(¬rain) = {countNotRain_RainPredicted/countRainPredicted}")
    print(f"P_¬rainpred(rain) = {countRain_RainNotPredicted/
↪countRainNotPredicted}")
```

[33]:
```
#weatherMC(0,100)
#weatherMC(1,100)
#weatherMC(2,100)

print("")

weatherMC(0,1000000)
#weatherMC(1,1000000)
#weatherMC(2,1000000)
```

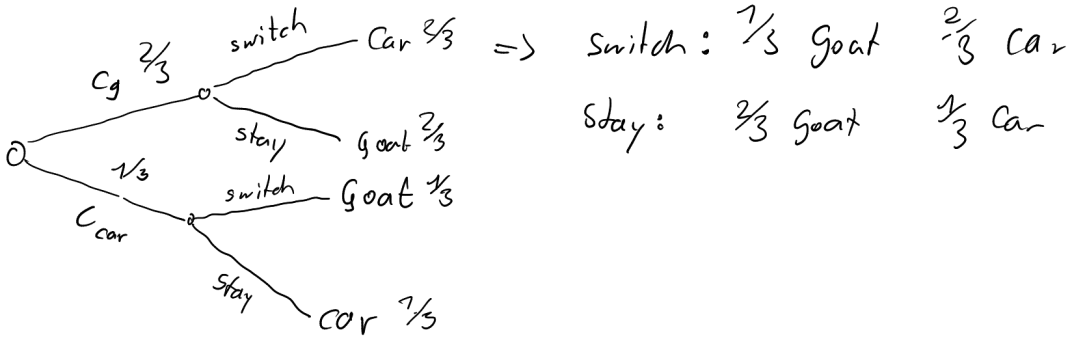
```
P(rain)=0.049969, P(¬ rain)=0.950031
P(rainPred)=0.135605, P(¬rainPred)=0.864395
P(rain   rainPred)=0.039911, P(rain   ¬rainPred)=0.010058, P(¬rain
rainPred)=0.095694, P(¬rain   ¬rainPred)=0.854337
P( (rain   ¬rainPred) + (¬rain   rainPred) )= 0.105752

P_rainpred(¬rain) = 0.7056819438811254
P_¬rainpred(rain) = 0.011635884057635687
```

2.3 Exercise 2.3 (Obligatory)

A famous logical decision problem is the so-called "Monty Hall Dilemma", named after the host of an American television game show ("Let's make a deal"). In German it is referred to as the "Ziegenproblem". Imagine you are contestant in a game show. There are three doors with an automobile behind one of them and goats behind the other two doors. The automobile and the two goats are assigned randomly to the three doors and you don't have any prior knowledge about where the goats or the automobile are. "First you point toward a door," says the host of the show. "Then I'll open one of the other doors to reveal a goat. After I've shown you the goat, you make your final choice whether to stick with your initial choice of doors, or to switch to the remaining door. You win whatever is behind the door." You begin by pointing to door number 1. The host shows you that door number 3 has a goat. What do you think - shall you stay with your initial

choice (door 1) or switch to door number 2 to win the car? What are the probabilities to win the car? Calculate the probabilities by hand and write a programme that simulates a large number of such games to validate your calculated results.

2.4 Theoretical Calculation



Python Approach

[47]:
```
import random

def MontyHall_calculation():
    #When you first walk into the gameshow, the probability of the car being␣
 ↪behind one of the 3 doors is:
    p_1 = 1./3
    p_2 = 1./3
    p_3 = 1./3

    #You choose door 1.
    #If the prize is behind door 1 too, find the probability that the host␣
 ↪reveals door 3.
    1/2

    #If the prize is behind door 2, find the probability that the host reveals␣
 ↪door 3. And the same if prize is behind door 3.
    1, 0

    #The host reveals door 3 has a cute snow goat.
    # Find The probability of the car being behind door 2, given that door 3 has␣
 ↪been revealed.
    2/3 == 1/3 + 1/3

    # TODO: Calculate the probalitiy of winning the car after switching the door␣
 ↪2 and also the probability to win a car after sticking to door 1
    2/3, 1/3
```

[34]:
```
def MontyHall(Switch_door, Ngames):
    doors = ["A", "B", "C"]

    goat_pos = np.random.randint(3, size=Ngames)
    p_stay = np.sum(goat_pos == 0)/Ngames

    p_switch = np.sum((1-(goat_pos == 0)))/Ngames
    print(p_switch if Switch_door else p_stay)
```

[35]:
```
MontyHall(False, 100000)
MontyHall(True, 100000)
```

```
0.33364
0.66465
```

[ ]:
```

```