TDT 4171 - Artificial Intelligence Methods

Assignment 1

> Zachari THIRY - 01/17/2023

Exercise 1

Consider the set of all possible five-card poker hands dealt fairly from a standard deck of fifty-two cards.

> a. How many atomic events are there in the joint probability distribution (i.e., how many five-card hands are there)?

The number of different outcomes, or atomic events, is the number of combinations of 5 among 52 items :

Let N be the number of atomic events :

$$N = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$N = \binom{52}{5} = \frac{52!}{(47!) \cdot 5!} = 2.59896\text{e+}6$$

> b. What is the probability of each atomic event?

Under the assumption of a "hand dealt fairly", one can assume uniform distribution. As such, let ε be an atomic event, Ω the universe of possible :

$\forall \varepsilon \in \Omega, P(\varepsilon) = \frac{1}{2.598\text{e+}6}$

> c. What is the probability of being dealt a royal straight flush? Four of a kind?

There are only 4 royal straight flushes. To compute the probability of being dealt with it in a uniform environment, we sum the atomic events over their probability :

$$P(\text{" } being\ dealt\ with\ a\ straight\ royal\ flush\ \text{"}) = \frac{4}{2598960} = \frac{1}{1540000}$$

The probability of being dealt with 4 of a kind is described the following way :

- Chosse 1 number for the unique card
- Choose 1 number for the four identical cards
- Pick the four identical cards

- Pick one card out of the remaining ones

As a probability, we can write : let e = {"Be dealt 4 of a kind" } :

$$P(e) = \frac{1}{\binom{13}{1}\binom{12}{1}\binom{4}{4}\binom{48}{1}}$$

Exercise 2

Deciding to put probability theory to good use, we encounter a slot machine with three independent wheels, each producing one of the four symbols BAR, BELL, LEMON, or CHERRY with equal probability. The slot machine has the following payout scheme for a bet of 1 coin (where “?” denotes that we don’t care what comes up for that wheel):

- (BA3) BAR/BAR/BAR pays 20 coins
- (BE3) BELL/BELL/BELL pays 15 coins
- (LE3) LEMON/LEMON/LEMON pays 5 coins
- (CH3) CHERRY/CHERRY/CHERRY pays 3 coins
- (CH2) CHERRY/CHERRY/? pays 2 coins
- (CH1) CHERRY/?/? pays 1 coin

> a. Compute the expected “payback” percentage of the machine. In other words, for each coin played, what is the expected coin return?

Let's first determine some probabilities :
$P(BA_3) = P(BE_3) = P(LE_3) = P(CH_3) = \frac{1}{4^3}$
$P(CH_2) = \frac{1}{4^2} - P(CH_3) = 0.04685$
$P(CH_1) = \frac{1}{4} - P(CH_2) - P(CH_3) = 0.171875$
And now, looking at the expectation :

$E_\Omega(X) = \sum_{x \in X} f(x)P(x = \omega)$

In [1]:
```
E = 1 * 0.171875 + 2 * (0.04685) + (3 + 5 + 15 + 20) * 1/4**3
print(f"One can expect a return of {E} for each coin played")
```

```
One can expect a return of 0.93745 for each coin played
```

> b. Compute the probability that playing the slot machine once will result in a win.

The game resulting in a win means getting one of the possible outcomes :

- There are $4 * 4 * 4 = 64$ possible outcomes
- $4 * 4$ oucomes for "CH_1"
- 4 outcomes for "CH_2"
- 1 outcome for each of the XX_3 so 4 total"

$$P(\text{" }win\text{ "}) = \frac{\sum winning\ outcomes}{possible\ outcomes} = \frac{(16) + (4) + (4 * 1)}{64} = 0.328125$$

> c. Estimate the mean and median number of plays you can expect to make until you go broke,if you start with 10 coins. Run a simulation in Python to estimate this. Add your results to your PDF report.

In [2]:
```
import random
import numpy as np

class Machine:

    def __init__(self, initial_bet:int):
        self.classes = ["BAR", "BELL", "LEMON", "CHERRY"]
        self.balance = initial_bet
        self.iteration = 0

    def reward(self, output):
        reward = 0
        if output[0:1] == ["CHERRY"]: reward = 1
        if output[0:2] == ["CHERRY", "CHERRY"]: reward = 2
        if output == ["CHERRY", "CHERRY", "CHERRY"]: reward = 3
        if output == ["LEMON", "LEMON", "LEMON"]: reward = 5
        if output == ["BELL", "BELL", "BELL"]: reward = 15
        if output == ["BAR", "BAR", "BAR"]: reward = 20

        return reward

    def run(self):
        result = np.random.choice(self.classes, 3).tolist()
        self.balance += self.reward(result) - 1
        self.iteration += 1
        if self.balance == 0 :
            raise Exception("Ran out of balance")

    def get_iteration(self):
        return self.iteration
```

In [3]:
```
N = 1000
initial_bet = 10

Iterations = []

for i in range(N):
    M = Machine(initial_bet)
    while(True):
        try:
            M.run()
        except Exception:
            break
    Iterations.append(M.get_iteration())

average = sum(Iterations)/len(Iterations)
median = sorted(Iterations)[int(len(Iterations)/2)]

print(f"Average flight time is estimated to {average} for N = {N}")
print(f"Median flight time is estimated to {median} for N = {N}")
```

```
Average flight time is estimated to 238.26 for N = 1000
Median flight time is estimated to 20 for N = 1000
```

Exercise 3

This exercise consists of two parts that ask you to run simulations to compute the answers instead of trying to compute exact answers. Add your answers to your PDF report.

Part 1 :

Peter is interested in knowing the possibility that at least two people from a group of N people have a birthday on the same day. Your task is to find out what N has to be for this event to occur with at least 50% chance. We will disregard the existence of leap years and assume there are 365 days in a year that are equally likely to be the birthday of a randomly selected person.

> a. Create a function that takes N and computes the probability of the event via simulation.

In [4]:
```
def find_birthday_probability(N:int) -> float :
    P_collision = 1
    for i in range(0,N):
        P_collision *= (365-i)/365
    return 1-P_collision
```

In [5]:
```
N = 23
P_collision = find_birthday_probability(N)
print("The probability of having colliding birthday"+
      f"among {N} people is P_collision = {P_collision}")
```

```
The probability of having colliding birthdayamong 23 people is P_collision = 0.507
2972343239857
```

> b. Use the function created in the previous task to compute the probability of the event given N in the interval [10, 50]. In this interval, what is the proportion of N where the event happens with the least 50% chance? What is the smallest N where the probability of the event occurring is at least 50%?

In [6]:
```
Range = range(10,50)

print("N   | Probability")
min_N = None
count_N = 0

for N in range(10,50) :
    P_collision = find_birthday_probability(N)
    print(f"{N} | {P_collision}")

    if P_collision > 0.5 :
        count_N += 1
        min_N = N if min_N == None else min_N

print("#####################################################")
print(f"Minimum N such that P_collision > 0.5 is : {min_N}")
print(f"Proportion of Ns where P_collision > 0.5 is : {count_N/len(Range)}")
```

```
N  | Probability
10 | 0.11694817771107768
11 | 0.14114137832173312
12 | 0.1670247888380645
13 | 0.19441027523242949
14 | 0.223102512004973
15 | 0.25290131976368646
16 | 0.2836040052528501
17 | 0.3150076652965609
18 | 0.3469114178717896
19 | 0.37911852603153695
20 | 0.41143838358058027
21 | 0.443688335165206
22 | 0.4756953076625503
23 | 0.5072972343239857
24 | 0.538344257914529
25 | 0.568699703969464
26 | 0.598240820135939
27 | 0.6268592822632421
28 | 0.6544614723423995
29 | 0.6809685374777771
30 | 0.7063162427192688
31 | 0.7304546337286439
32 | 0.7533475278503208
33 | 0.7749718541757721
34 | 0.7953168646201543
35 | 0.8143832388747153
36 | 0.8321821063798795
37 | 0.8487340082163846
38 | 0.864067821082121
39 | 0.878219664366722
40 | 0.891231809817949
41 | 0.9031516114817354
42 | 0.9140304715618692
43 | 0.9239228556561199
44 | 0.9328853685514263
45 | 0.940975899465775
46 | 0.9482528433672548
47 | 0.9547744028332994
48 | 0.9605979728794225
49 | 0.9657796093226765
#####################################################
Minimum N such that P_collision > 0.5 is : 23
Proportion of Ns where P_collision > 0.5 is : 0.675
```

Part 2

Peter wants to form a group where every day of the year is a birthday (i.e., for every day of the year, there must be at least one person from the group who has a birthday). He starts with an empty group, and then proceeds with the following loop:

1. Add a random person to the group.
2. Check whether all days of the year are covered.
3. Go back to step 1 if not all days of the year have at least one birthday person from the group.

> a. How large a group should Peter expect to form? Make the same assumption about leap years as in Part 1.

In [7]:

```
import random

def check_group(Group):
    for i in range(1,366):
        if not i in Group:
            return False
    return True



def find_group_size(loops:int = 10):
    sizes = []
    for i in range(loops):
        group = []
        while not check_group(group):
            group.append(random.randrange(1,366))
        sizes.append(len(group))

    return sum(sizes)/len(sizes)
```

In [8]:

```
Loops = 20

print(f"Based on a {Loops} loops simulation, the expected "+
      f"size of the group is of {find_group_size(Loops)} persons.")
```

```
Based on a 20 loops simulation, the expected size of the group is of 2209.6 person
s.
```