Author:

- **Sidharth**
- *2K18/ MC/ 114*

Objective:

Demonstrate Markov Chain. WAP to implement Markov Chain

1. Gambler Ruin Problem

2. Weather Forecasting Problem

Theory:

Modern probability theory studies chance processes for which the knowledge of previous outcomes influences predictions for future experiments. In principle, when we observe a sequence of chance experiments, all of the past outcomes could influence our predictions for the next experiment. For example, this should be the case in predicting a student’s grades on a sequence of exams in a course. But to allow this much generality would make it very difficult to prove general results.

A Markov chain is a stochastic process, but it differs from a general stochastic process in that a Markov chain must be "memory-less." That is, (the probability of) future actions are not dependent upon the steps that led up to the present state. This is called the Markov property. While the theory of Markov chains is important precisely because so many "everyday" processes satisfy the Markov property, there are many common examples of stochastic properties that do not satisfy the Markov property.

We describe a Markov chain as follows: We have a set of states, S = {s1, s2,...,sr}. The process starts in one of these states and moves successively from one state to another. Each move is called a step. If the chain is currently in state si, then it moves to state sj at the next step with a probability denoted by pij , and this probability does not depend upon which states the chain was in before the current state.

Code and Output:

1. Gambler Ruin Problem

> A gambler has a fortune of ₹20 and he bets ₹1 at a time and wins ₹1 with probability 3/10. He stops playing if he loses all his fortune or doubles it. Write the transition probability matrix that he loses his fortune at the end of 30 plays.

```
p = 0.3
n = 30
fortune = 20

import numpy as np
tm=np.zeros(shape=(int(2*fortune+1), int(2*fortune+1)))
print("Absorbing barriers are at", 0, "and", 2*fortune)
for i in range(0, int(2*fortune+1)):
    for j in range(0 ,int(2*fortune+1)):
        if i==0 or i==int(2*fortune):
            tm[i][i]=1
        else:
            if i+1==j:
                tm[i][j]=p
            elif i-1==j:
                tm[i][j]=1-p

b=np.linalg.matrix_power(tm, n)
print("Probability of gambler losing his fortune at the end of", n, '
print(b[int(fortune)][0])
```

```
Absorbing barriers are at 0 and 40
Probability of gambler losing his fortune at the end of 30 plays:
0.08067671520774745
```

2. Weather Forecasting Problem

> Provided with every states probability that rain occurs yesterday and today. Let it rained on both Saturday and Sunday. What is the probability that it will rain on Monday?

> We have Probabilities:

- State 0: Rained today and yesterday: 0.4
- State 1: Rained today but not yesterday: 0.2
- State 2: Rained yesterday but not today: 0.3
- State 3: Didn't rain today or yesterday: 0.5

```
import numpy as np
import math

state0 = 0.4
state1 = 0.2
state2 = 0.3
state3 = 0.5
wid = 4
hig = 4

tm = [[0 for x in range(wid)] for y in range(hig)]

tm[0][0] = state0
tm[0][2] = 1-state0
tm[1][0] = state1
tm[1][2] = 1-state1
tm[2][1] = state2
tm[2][3] = 1-state2
tm[3][1] = state3
tm[3][3] = 1-state3

print("The transition matrix:\n")
for i in a:
    print(i)

a=np.linalg.matrix_power(a, 2)
print("\nProbability of raining tommorow with the past two days havir
print(a[0][0]+a[0][1])
```

```
The transition matrix:

[0.08933236 0.26764818 0.26783304 0.37518642]
[0.08927768 0.26764124 0.26749552 0.37558556]
[0.08921606 0.26813975 0.26764124 0.37500295]
[0.0893301  0.26785925 0.2682754  0.37453525]

Probability of raining tommorow with the past two days having rained:
0.35714289990042936
```

Result:

Implemented Markov Chain for Gambler Ruin Problem and Weather Forecasting Problem.

Discussion:

Markov chains are an important concept in stochastic processes. They can be used to greatly simplify processes that satisfy the Markov property, namely that the future state of a stochastic variable is only dependent on its present state.