National University of Computer & Emerging Sciences, Karachi
Computer Science Department Spring 2023, Lab Manual – 08

Course Code: AI-2002	Course : Artificial Intelligence Lab
Instructor(s):	Kariz Kamal, Sania Uroof, Saeeda Kanwal, Omar Qureshi, Shakir Hussain, Ali Fatmi

Contents:

Objective

1. *Introduction to basics of probability theory*
2. *Computing probabilities of a single observation & across a range of observations*
3. Explore *how to use python to solve basic probability and* Bayesian Network *problems and then apply these skills to a couple of challenge problems.*
4. Introduction to different *probability and* Bayesian Network Libraries of python.

Conditional Probability

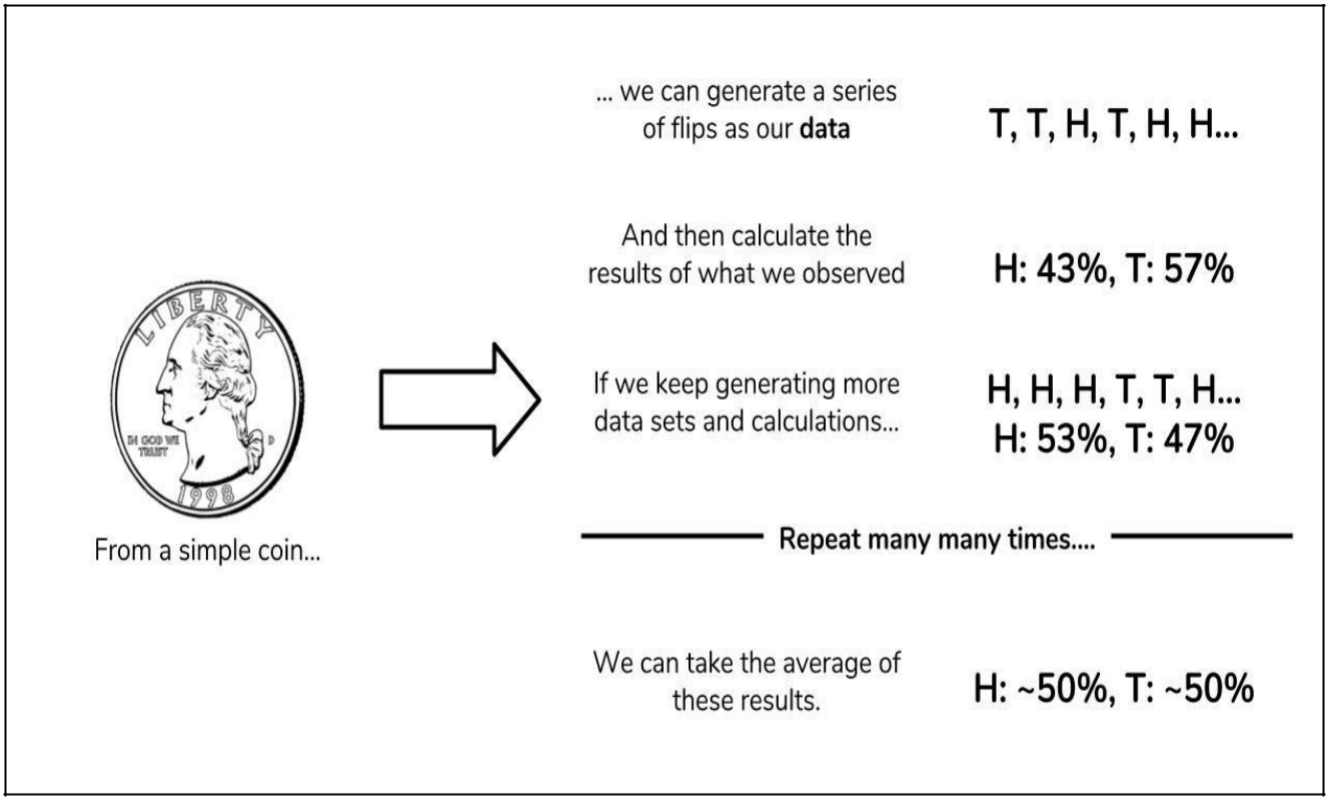
At the most basic level, probability seeks to answer the question, "What is the chance of an event happening?" An **event** is some outcome of interest. To calculate the chance of an event happening, we also need to consider all the other events that can occur. The quintessential representation of probability is the humble coin toss. In a coin toss the only events that can happen are:

- Flipping a heads
- Flipping a tails

*These two events form the **sample space**, the set of all possible events that can happen. To calculate the probability of an event occurring, we count how many times are event of interest can occur (say flipping heads) and dividing it by the sample space. Thus, probability will tell us that an ideal coin will have a 1-in-2 chance of being heads or tails. By looking at the events that can occur, probability gives us a framework for making* predictions *about how often events will happen. However, even though it seems obvious, if we actually try to toss some coins, we're likely to get an abnormally high or low counts of heads every once in a while. If we don't want to make the assumption that the coin is fair, what can we do? We can gather data! We can use statistics to calculate probabilities based on observations from the real world and check how it compares to the ideal.*

From statistics to probability

Our data will be generated by flipping a coin 10 times and counting how many times we get heads. We will call a set of 10 coin tosses a trial. *Our data point will be the number of heads we observe. We may not get the "ideal" 5 heads, but we won't worry too much since one trial is only one data point. If we perform many, many trials, we expect the* average *number of heads over all of our trials to approach the 50%. The code below simulates 10, 100, 1000, and 1000000 trials, and then calculates the average proportion of heads observed. Our process is summarized in the image below as well.*



Code # 01

```
"""
There are 52 cards In a standard deck of cards and of those 52 cards, 4
are Aces.If you follow the example of the coin flipping from above to
kn ow the probabilityof drawing an Ace, you'll divide the number of
possibl e event outcomes (4), by the sample space (52) """
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv) import matplotlib.pyplot as plt
# Sample Space
cards = 52
# Outcome
s aces =
4
# Divide possible outcomes by the sample
set ace_probability = aces / cards
# Print probability rounded to two decimal
places print(round(ace_probability, 2))
# Ace Probability Percent Code
ace_probability_percent = ace_probability * 100
# Print probability percent rounded to one decimal place
print(str(round(ace_probability_percent, 0)) + '%')
```

Code # 02

```
"""the probability of drawing a card that is a Heart, a face card (such
as Jacks, Queens, or Kings), or a combination of both, such as a Queen o
f Hearts."""
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv) import matplotlib.pyplot as plt
# Create function that returns probability percent rounded to one decima
l place
def event_probability(event_outcomes, sample_space):
    probability = (event_outcomes / sample_space) *
    100 return round(probability, 1)
# Sample Space
cards = 52
# Determine the probability of drawing a heart
hearts = 13
heart_probability = event_probability(hearts, cards)
# Determine the probability of drawing a face
card face_cards = 12
face_card_probability = event_probability(face_cards, cards)
# Determine the probability of drawing the queen of hearts
queen_of_hearts = 1
queen_of_hearts_probability = event_probability(queen_of_hearts, cards)
# Print each probability
print("Probability of Heart :- ",str(heart_probability) + '%')
print("Probability of Face Card :- ",str(face_card_probability) + '%')
print("Probability of Queen of Hearts :- ",str(queen_of_hearts_probabili
ty) + '%')
```

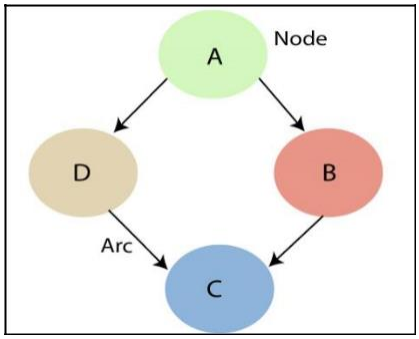
Bayesian Network

A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph. It is also called a Bayes network, belief network, decision network, or Bayesian model. Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection. Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty**.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- Directed Acyclic Graph
- Table of conditional probabilities.

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an Influence diagram. A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each node corresponds to the random variables, and a variable can be continuous or discrete.
- Arc or directed arrows represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.
  These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other
- In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.
- If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.
- Node C is independent of node A.

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution **$P(X_i \mid Parent(X_i))$**, which determines the effect of the parent on that node.

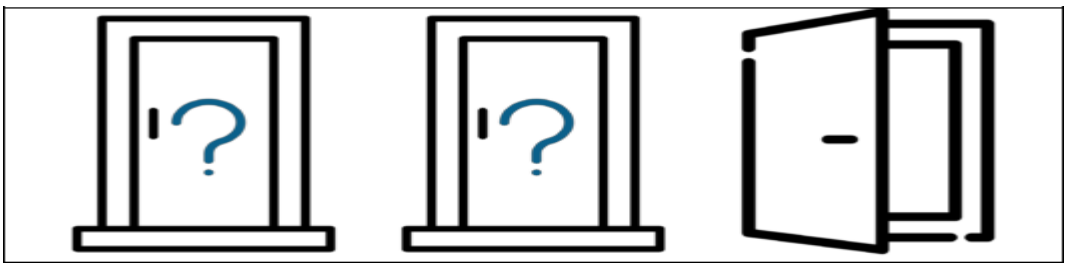
Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

Bayesian Networks Python

In this demo, we'll be using Bayesian Networks to solve the famous Monty Hall Problem. For those of you who don't know what the Monty Hall problem is, let me explain:

The Monty Hall problem named after the host of the TV series, 'Let's Make A Deal', is a paradoxical probability puzzle that has been confusing people for over a decade.

So this is how it works. The game involves three doors, given that behind one of these doors is a car and the remaining two have goats behind them. So you start by picking a random door, say #2. On the other hand, the host knows where the car is hidden and he opens another door, say #1 (behind which there is a goat). Here's the catch, you're now given a choice, the host will ask you if you want to pick door #3 instead of your first choice i.e. #2.



Is it better if you switch your choice or should you stick to your first choice?

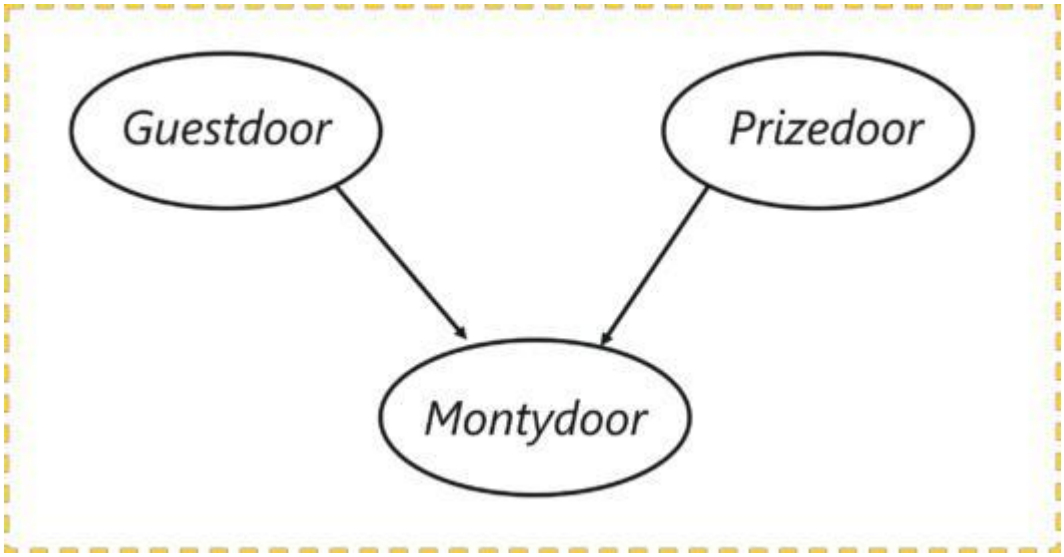
This is exactly what we're going to model. We'll be creating a Bayesian Network to understand the probability of winning if the participant decides to switch his choice.

Now let's get started.

The first step is to build a Directed Acyclic Graph.

The graph has three nodes, each representing the door chosen by:

1. The door selected by the Guest
2. The door containing the prize (car)
3. The door Monty chooses to open



Let's understand the dependencies here, the door selected by the guest and the door containing the car are completely random processes. However, the door Monty chooses to open is dependent on both the doors; the door selected by the guest, and the door the prize is behind. Monty has to choose in such a way that the door does not contain the prize and it cannot be the one chosen by the guest.

```
#Import required packages
import math
from pomegranate import *
 # Initially the door selected by the guest is completely random
guest =DiscreteDistribution( { 'A': 1./3, 'B': 1./3, 'C': 1./3 } )
 # The door containing the prize is also a random process
prize =DiscreteDistribution( { 'A': 1./3, 'B': 1./3, 'C': 1./3 } )
 # The door Monty picks, depends on the choice of the guest and the prize
d oor
monty =ConditionalProbabilityTable( [[ 'A',
'A', 'A', 0.0 ],
[ 'A', 'A', 'B', 0.5 ],
[ 'A', 'A', 'C', 0.5 ],
[ 'A', 'B', 'A', 0.0 ],
[ 'A', 'B', 'B', 0.0 ],
[ 'A', 'B', 'C', 1.0 ],
[ 'A', 'C', 'A', 0.0 ],
[ 'A', 'C', 'B', 1.0 ],
[ 'A', 'C', 'C', 0.0 ],
[ 'B', 'A', 'A', 0.0 ],
[ 'B', 'A', 'B', 0.0 ],
[ 'B', 'A', 'C', 1.0 ],
[ 'B', 'B', 'A', 0.5 ],
[ 'B', 'B', 'B', 0.0 ],
[ 'B', 'B', 'C', 0.5 ],
[ 'B', 'C', 'A', 1.0 ],
[ 'B', 'C', 'B', 0.0 ],
[ 'B', 'C', 'C', 0.0 ],
[ 'C', 'A', 'A', 0.0 ],
[ 'C', 'A', 'B', 1.0 ],
[ 'C', 'A', 'C', 0.0 ],
[ 'C', 'B', 'A', 1.0 ],
[ 'C', 'B', 'B', 0.0 ],
[ 'C', 'B', 'C', 0.0 ],
[ 'C', 'C', 'A', 0.5 ],
[ 'C', 'C', 'B', 0.5 ],
[ 'C', 'C', 'C', 0.0 ]], [guest, prize] )

d1 = State( guest, name="guest" )
d2 = State( prize, name="prize" )
d3 = State( monty, name="monty" )

#Building the Bayesian Network
network = BayesianNetwork( "Solving the Monty Hall Problem With Bayesian
Ne tworks" )
network.add_states(d1, d2, d3)
network.add_edge(d1, d3)
network.add_edge(d2, d3)
network.bake()
```

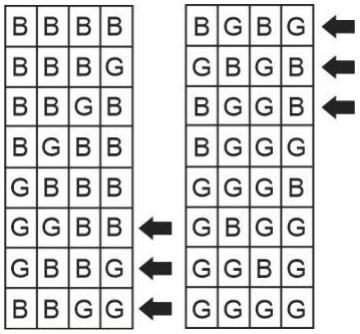
```
beliefs = network.predict_proba({ 'guest' : 'A' })
beliefs = map(str, beliefs)
print("n".join( "{}t{}".format( state.name, belief ) for state, belief in z
ip( network.states, beliefs ) ))
```

```
beliefs = network.predict_proba({'guest' : 'A', 'monty' : 'B'})
print("n".join( "{}t{}".format( state.name, str(belief) ) for state, belief
 in zip( network.states, beliefs )))
```

TASKS

Write Python Program for each task, create sample space and calculate the probability and show the output.

1. *Two unbiased dice are thrown once and the total score is observed. Use a simulation to find the estimated probability that the total score is* ***even or greater than 7****?*
2. A box contains 10 white balls, 20 reds and 30 greens. Draw 5 balls with replacement… what is the probability that:
   a. 3 white or 2 red
   b. All 5 are the same color
3. *The sample space for four sibling children. Each row in the sample space contains 1 of 16 possible outcomes. Every outcome represents a unique combination of four children. The sex of each child is indicated by a letter: B for boy and G for girl. Outcomes with two boys are marked by an arrow. Calculate the probability of two boys.*



4. A coin is tossed twice. What is the probability that at least 1 head occurs? The sample space for this experiment is S = {HH, HT, TH, TT}.
5. A die is loaded in such a way that an even number is twice as likely to occur as an odd number. If E is the event that a number less than 4 occurs on a single toss of the die, find P (E).
6. Let A be the event that an even number turns up and let B be the event that a number divisible by 3 occurs. Find P (A ∪ B) and P (A ∩ B).
7. A manufacturing firm employs three analytical plans for the design and development of a particular product. For cost reasons, all three are used at varying times. In fact, plans 1, 2, and 3 are used for 30%, 20%, and 50% of the products, respectively. The defect rate is different for the three procedures as follows: P(D | P1)=0.01, P(D | P2)=0.03, P(D | P3)=0.02, where P(D | Pj) is the probability of a defective product, given plan j. If a random product was observed and found to be defective, which plan was most likely used and thus responsible?