**National University of Computer & Emerging Sciences, Karachi Computer Science Department**

**Spring 2023, Lab Manual – 07**

|  |  |
| --- | --- |
| **Course Code: AI-2002** | **Course: Artificial Intelligence Lab** |
| **Instructor(s):** | **Mehak Mazhar, Sohail Ahmed, Zain Noreen, Sarah, Zarnain** |

# Objective

* Introduction to basics of probability theory
* Computing probabilities of a single observation & across a range of observations
* Explore how to use python to solve basic probability and Bayesian Network problems and then apply these skills to a couple of challenge problems.
* Introduction to Markov model libraries of python.
* Introduction to BN,DBN & HMM
* Explore how to use python to solve BN, DBN & HMM problems and then apply these skillsto a couple of challenge problems.
* Introduction to different Libraries of python.

## 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.

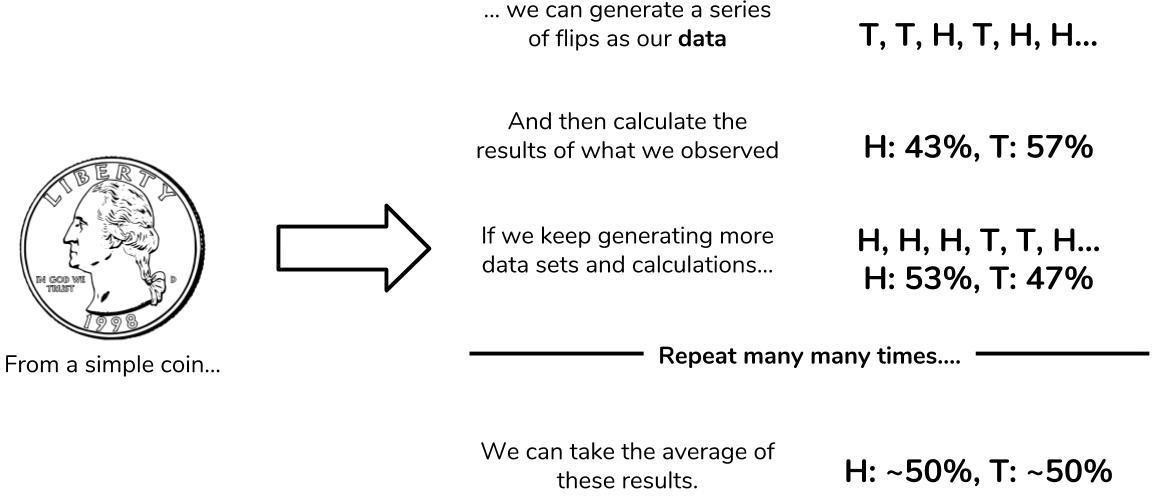
## From statistics to probability

## 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.

## Conditional Probability

## Conditional probability is the probability of an event A occurring given that event B has already occurred. It is denoted as P(A|B), where P represents probability. The formula for conditional probability is:

## P(A|B) = P(A ∩ B) / P(B)



|  |
| --- |
| **CODE 1** |
| **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 know the probability of drawing an Ace, you'll divide the number of possible event outcomes (4), by the sample space (52)** |
| import numpy as np  import pandas as pd  # Sample Space  cards = 52 #Outcomes  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 2** |
| **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 of Hearts.** |
| import numpy as np # linear algebra  import pandas as pd  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\_probability) + '%') |

# 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 decisionmaking 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.

**Elements of Bayesian Network**

The generalized form of Bayesian network that represents and solve decision problems underuncertain knowledge is known as an Influence diagram. A Bayesian network graph is made up of various elements.

**Nodes (or Vertices):** Nodes represent random variables in the domain being modeled. Each node corresponds to a specific variable or feature. These variables can be observable or latent (hidden).

**Edges**: Edges are directed connections between nodes, indicating direct probabilistic dependencies between variables. An edge from node A to node B means that the value of B is dependent on the value of A.

**Conditional Probability Tables (CPTs**): Each node (except for the root nodes) in a Bayesian network is associated with a conditional probability table. These tables specify the conditional probability distribution of a node given its parents (i.e., nodes that have edges pointing to it).

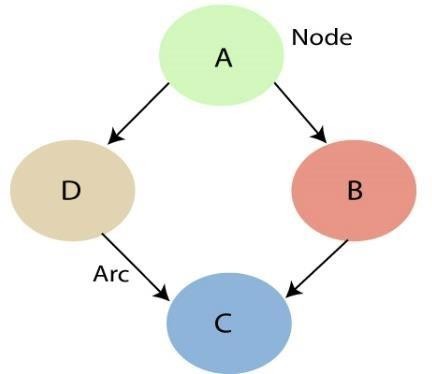
**Parent Nodes:** For a given node in the network, its parent nodes are the nodes that have a directed edge pointing to it. The value of the child node is probabilistically dependent on the values of its parent nodes.

**Root Nodes**: Root nodes are nodes in the network that have no incoming edges. They represent variables that are not influenced by any other variables in the network. In many cases, root nodes are the observed variables or inputs to the system.

**Leaf Nodes:** Leaf nodes are nodes in the network that have no outgoing edges. They represent variables that do not directly influence any other variables in the network. Leaf nodes often represent observed variables or outputs of the system.

**Network Structure:** The overall structure of the network defines how nodes are connected through edges. This structure determines the conditional independence relationships between variables and facilitates efficient probabilistic inference.

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

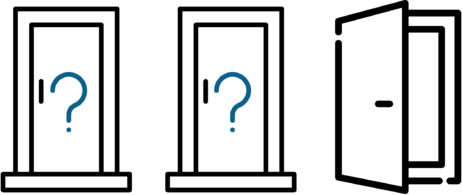


## Bayesian Networks Python

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 randomdoor, say #2. On the other hand, the host knows where the car is hidden and he opensanother door, say #1 (behind which there is a goat). Here’s the catch, you’re now given achoice, the host will ask you if you want to pick door #3 instead of your first choice i.e. #2.

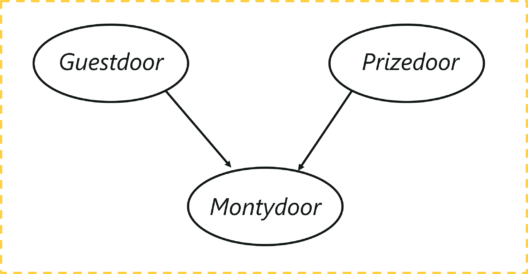


Is it better if you switch your choice or should you stick to your first choice?

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:
* The door selected by the Guest
* The door containing the prize (car)
* 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.

|  |
| --- |
| **Monty Hall Problem** |
| #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 door  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 Networks" )  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 zip( 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 ))) |

|  |
| --- |
| **EXAMPLE 2**  Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds to minor earthquakes. Harry has two neighbors David and Sophia, who have taken responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he gets confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of a Burglary Alarm. |
| from pomegranate import \*  # Define the states  burglary = DiscreteDistribution({'True': 0.001, 'False': 0.999})  earthquake = DiscreteDistribution({'True': 0.002, 'False': 0.998})  alarm = ConditionalProbabilityTable(      [['True', 'True', 'True', 0.95],       ['True', 'True', 'False', 0.05],       ['True', 'False', 'True', 0.94],       ['True', 'False', 'False', 0.06],       ['False', 'True', 'True', 0.29],       ['False', 'True', 'False', 0.71],       ['False', 'False', 'True', 0.001],       ['False', 'False', 'False', 0.999]],      [burglary, earthquake]  )  david\_calls = ConditionalProbabilityTable(      [['True', 'True', 0.9],       ['True', 'False', 0.1],       ['False', 'True', 0.05],       ['False', 'False', 0.95]],      [alarm]  )  sophia\_calls = ConditionalProbabilityTable(      [['True', 'True', 0.7],       ['True', 'False', 0.3],       ['False', 'True', 0.01],       ['False', 'False', 0.99]],      [alarm]  )  # Define the nodes  s1 = State(burglary, name="burglary")  s2 = State(earthquake, name="earthquake")  s3 = State(alarm, name="alarm")  s4 = State(david\_calls, name="david\_calls")  s5 = State(sophia\_calls, name="sophia\_calls")  # Create the Bayesian network  network = BayesianNetwork("Burglary Alarm")  network.add\_states(s1, s2, s3, s4, s5)  network.add\_edge(s1, s3)  network.add\_edge(s2, s3)  network.add\_edge(s3, s4)  network.add\_edge(s3, s5)  network.bake()  # Calculate the probability of the alarm given burglary  prob\_alarm\_given\_burglary = network.predict\_proba({'burglary': 'False'})[2].parameters[0]['True']  print("Probability of Alarm given Burglary:", prob\_alarm\_given\_burglary) |

# Markov Model

Markov models are a type of probabilistic model that is used to predict the future state of a system, based on its current state. In other words, Markov models are used to predict the future state based on the current hidden or observed states. Markov model is a finite-state machine where each state has an associated probability of being in any other state after one step. They can be used to model real-world problems where hiddenand observable states are involved. Markov models can be classified into hidden and observable based on the type of information available to use for making predictions or decisions.

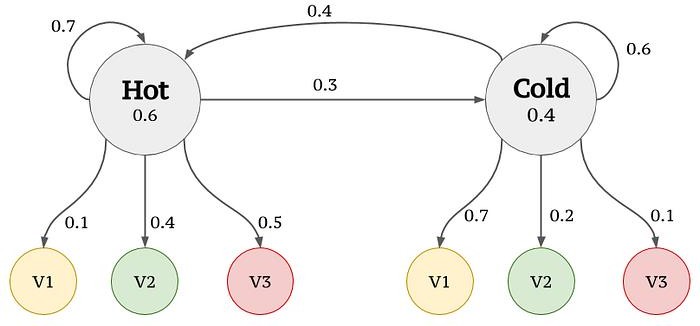
To better understand Markov models, let’s look at an example. Say you have a bag ofmarbles that contains four marbles: two red marbles and two blue marbles. You randomly select a marble from the bag, note its color, and then put it back in the bag.After repeating this process several times, you begin to notice a pattern: The probabilityof selecting a red marble is always two out of four, or 50%. This is because the probabilityof selecting a particular color of marble is determined by the number of that color of marble in the bag. In other words, the past history (i.e., the contents of the bag) determines the future state (i.e., the probability of selecting a particular color of marble).

# Hidden Markov Model

Hidden Markov models (HMMs) are a type of statistical modeling that has been used for several years. They have been applied in different fields such as medicine, computer science, and data science. The Hidden Markov model (HMM) is the foundation of many modern-day data science algorithms. It has been used in [datascience](https://vitalflux.com/category/data-science) to make efficient use of observations for successful predictions or decision- making processes.

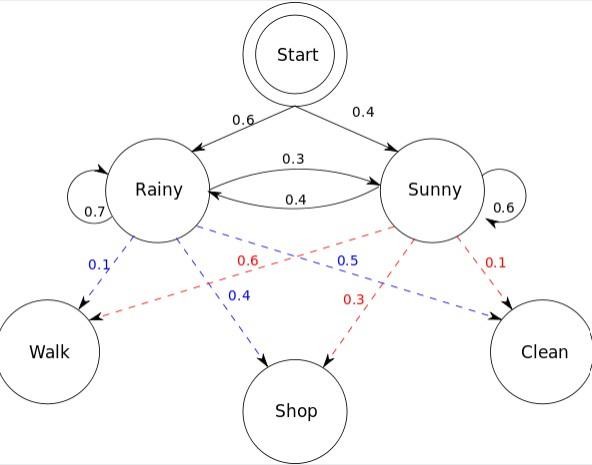
The [hidden Markov model (HMM)](https://en.wikipedia.org/wiki/Hidden_Markov_model) is another type of Markov model where there are few states which are hidden. This is where HMM differs from a Markov chain. HMM is a statistical model in which the system being modeled are Markov processes with unobserved or hidden states. It is a hidden variable model which can give an observation of another hidden state with the help of the Markov assumption. The hiddenstate is the term given to the next possible variable which cannot be directly observed but can be inferred by observing one or more states according to Markov’s assumption. **Markov assumption is the assumption that a hidden variable is dependent only on the previous hidden state.** Mathematically, the probability of being in a state ata time t depends only on the state at the time (t-1). It is termed a **limited horizon assumption**. Another Markov assumption states that the **conditional distribution over thenext state, given the current state, doesn’t change over time**. This is also termeda **stationary process assumption**.

A Markov model is made up of two components: the state transition and hidden random variables that are conditioned on each other. A hidden Markov model consists of five important components:



Let’s understand the above using the hidden Markov model representation shown below:

The hidden Markov model in the above diagram represents the process of predicting whether someone will be found to be walking, shopping, or cleaning on a particular day depending upon whether the day is rainy or sunny. The following represents five components of the hidden Markov model in the above diagram



**Elements of Hidden Markov Model:**

**Hidden States:** These are the unobservable states of the system that evolve over time according to a Markov process. Each hidden state represents a particular situation or condition of the system at a given time step.

**Observations:** At each time step, the system emits an observable event or observation, which depends on the underlying hidden state. Observations provide partial information about the hidden states.

**Transition Probabilities:** These represent the probabilities of transitioning from one hidden state to another. Transition probabilities capture the dynamics of the system and are typically represented by a transition matrix.

**Emission Probabilities:** These represent the probabilities of emitting each observation given the current hidden state. Emission probabilities capture the relationship between hidden states and observable events and are typically represented by an emission matrix.

**Initial State Distribution:** This represents the probability distribution of the initial hidden state. It specifies the likelihood of the system being in each hidden state at the beginning of the sequence.

|  |
| --- |
| **HMM** |
| **EXAMPLE**  A vending machine company is trying to optimize its inventory management system. They have two types of vending machines: one for drinks and another for snacks. Each vending machine can be in either a high-demand state or a low-demand state. The company wants to use a Hidden Markov Model (HMM) to predict the demand state of each vending machine based on sales data.  **Task:** The company has collected sales data for a week from both types of vending machines. Based on this data, they want to train an HMM to predict the demand state of each vending machine (high-demand or low-demand) and determine the transition and emission probabilities. |
| from hmmlearn import hmm  import numpy as np  # Define the model  model = hmm.MultinomialHMM(n\_components=2, n\_iter=100)  # Define the transition probabilities  model.transmat\_ = np.array([[0.8, 0.2],  # Transition probabilities for drinks vending machine                              [0.6, 0.4]]) # Transition probabilities for snacks vending machine  # Define the emission probabilities  model.emissionprob\_ = np.array([[0.1, 0.4, 0.5],  # Emission probabilities for drinks vending machine                                  [0.7, 0.2, 0.1]]) # Emission probabilities for snacks vending machine  # Define the starting probabilities  model.startprob\_ = np.array([0.6, 0.4])  # Starting probabilities for both vending machines  # Define the observed sequence  observed\_sequence = np.array([0, 1, 0, 0, 1])  # Sales data for a week (0 represents low demand, 1 represents high demand)  # Fit the model to the observed sequence  model.fit(observed\_sequence.reshape(-1, 1))  # Print the model parameters  print("Transition probabilities:")  print(model.transmat\_)  print("\nEmission probabilities:")  print(model.emissionprob\_)  print("\nStarting probabilities:")  print(model.startprob\_)  # Infer the hidden state sequence of the observed sequence  hidden\_states = model.predict(observed\_sequence.reshape(-1, 1))  print("\nHidden state sequence:")  print(hidden\_states) |