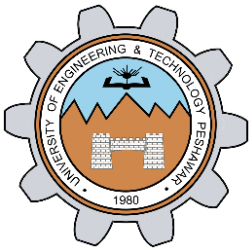
PROBABILITY METHODS IN ENGINEERING

ASSIGNMENT # 05



Spring 2023

CSE-209 Probability Methods In Engineering

Submitted by: **Shahzad Bangash, Suleman Shah , Ali Asghar**

Registration No. : **21PWCSE1980, 21PWCSE1983, 21PWCSE2059**

Class Section: **C**

"On my honor, as students of University of Engineering and Technology, We have neither given nor received unauthorized assistance on this academic work."

Submitted to:

Dr. Amaad Khalil

DATE: 20 / June / 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Probability Concepts and Implementation in Python

Introduction:

Probability theory is a fundamental branch of mathematics that deals with the study of uncertainty and randomness. It provides a framework for understanding and quantifying the likelihood of events occurring. In this report, we will explore various probability concepts and their implementation in Python.

1. Probability of an Event:

The probability of an event is a measure of the likelihood of that event occurring. It is represented by a number between 0 and 1, where 0 indicates impossibility and 1 indicates certainty. In Python, we can calculate the probability of an event by dividing the favorable outcomes by the total possible outcomes.

Example:

Consider a sample space with elements {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. The probability of choosing '10' can be calculated as follows:

Code & Output:

HeadTails_Prob.py

D: > UNi > PME > Python > HeadTails_Prob.py > ...

```
1   sample_space = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '10'}
2   probability_10 = 1 / len(sample_space)
3   print(f'Probability of choosing 10 is {probability_10}')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Admin> python -u "d:\UNi\PME\Python\HeadTails_Prob.py"
Probability of choosing 10 is 0.1
PS C:\Users\Admin>
```

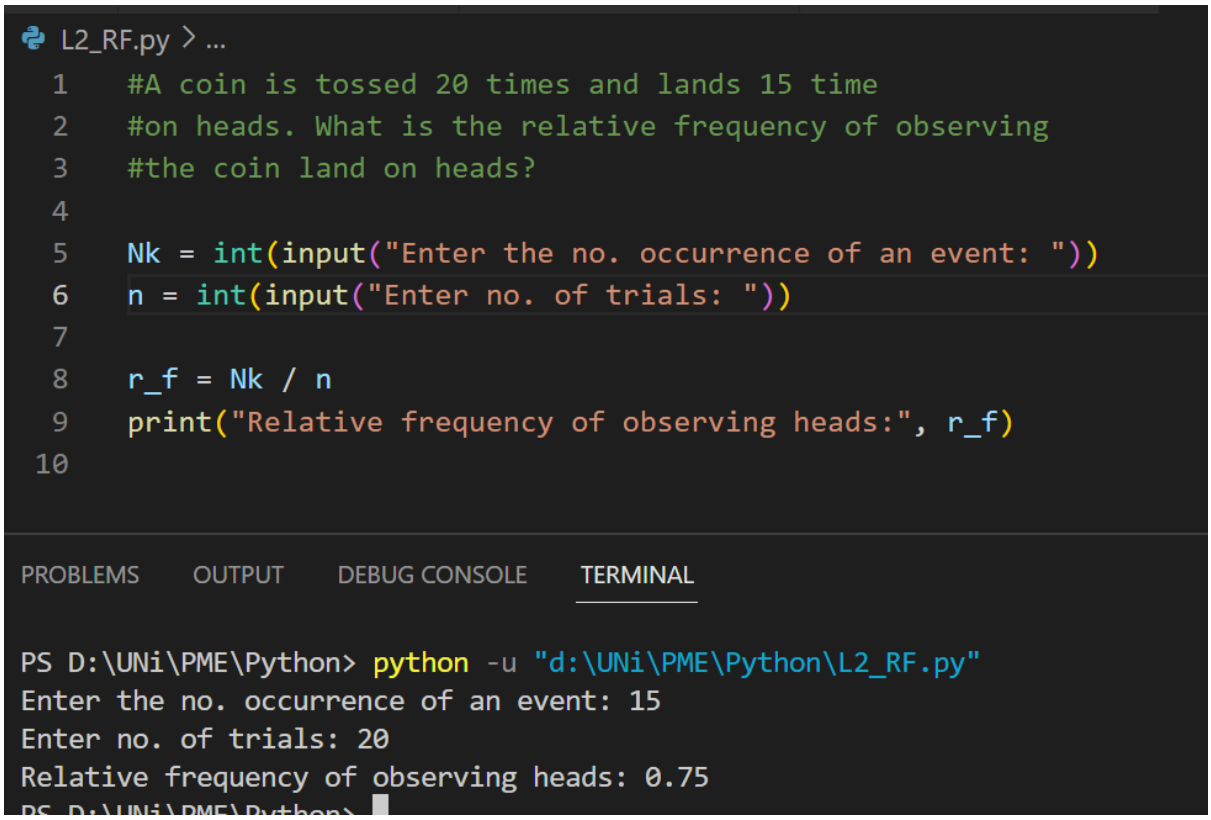
2. Relative Frequency:

Relative frequency is a concept used to estimate the probability of an event based on observed data. It is calculated by dividing the number of times the event occurred by the total number of trials or observations. In Python, we can compute the relative frequency using simple arithmetic operations.

Example:

Suppose a coin is tossed 20 times, and it lands on heads 15 times. The relative frequency of observing heads can be calculated as follows:

Code & Output:



3. Conditional Probability:

Conditional probability measures the likelihood of an event occurring given that another event has already occurred. It is calculated by dividing the probability of the intersection of the two events by the probability of the condition event. In Python, we can define a function to calculate the conditional probability.

Example:

Consider a scenario where 25% of a class passed both quizzes, and 42% passed the first quiz. We can calculate the percent of those who passed the first quiz and also passed the second quiz using the conditional probability formula.

Code and Output:

L6_ConditionalProbability.py > ...

```python
#A teacher gave his class two quizzes. 25% of the class
#passed both quizzes and 42% of the class passed the first
#quiz. What percent of those who passed the first quiz also
#passed the second quiz?

def calc_conditional_probability(p_a_and_b, p_b):
    return p_a_and_b / p_b

p_a_and_b = 0.25  # Probability of both A and B occurring
p_b = 0.42  # Probability of B occurring
conditional_prob = calc_conditional_probability(p_a_and_b, p_b)
print("Conditional Probability: {:.4f}".format(conditional_prob))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\UNi\PME\Python> python -u "d:\UNi\PME\Python\L6_ConditionalProbability.py"
Conditional Probability: 0.5952
PS D:\UNi\PME\Python>
```

4. Total Probability:

The total probability theorem allows us to calculate the probability of an event by considering all possible ways it can occur. In Python, we can define a function to compute the total probability of an event by summing the probabilities of all relevant outcomes.

Example:

An urn contains two black balls and three white balls. Two balls are selected at random from the urn without replacement and the sequence of colors is noted. Find the probability that the second ball is white (irrespective of first outcome).

Code:

L7_TotalProbability.py > probability_of_event

```python
#An urn contains two black balls and three white balls. Two
#balls are selected at random from the urn without
#replacement and the sequence of colors is noted. Find the
#probability that the second ball is white (irrespective of
#first outcome).

def probability_of_event(nodes, name):
    probability = 0

    def node_probability(node, name):
        if node.name == name:
            return [node.probability]

        if len(node.childrens) == 0:
            return []

        probabilities = []
        for children in node.childrens:
            for probability in node_probability(children, name):
                probabilities.append(node.probability * probability)

        return probabilities

    for node in nodes:
```

L7_TotalProbability.py > ...

```python
    for node in nodes:
        probability += sum(node_probability(node, name))

    return probability

class Node:
    def __init__(self, name, probability, childrens=[]):
        self.name = name
        self.probability = probability
        self.childrens = childrens

a1_h1 = Node('A1', 0.25)
a2_h1 = Node('A2', 0.75)
h1 = Node('H1', 0.4, [ a1_h1, a2_h1 ])

a1_h2 = Node('A1', 0.5)
a2_h2 = Node('A2', 0.5)
h2 = Node('H2', 0.6, [ a1_h2, a2_h2 ])

a1_probability = probability_of_event([h1, h2], 'A1')
a2_probability = probability_of_event([h1, h2], 'A2')
print(float(a1_probability))
print(float(a2_probability))
```

Output:

```
PS D:\UNi\PME\Python> python -u "d:\UNi\PME\Python\L7_TotalProbability.py"
Probability of Black = 0.4000
Probability of White = 0.6000
PS D:\UNi\PME\Python>
```

5. Bayes' Theorem:

Bayes' theorem provides a way to update the probability of an event based on new evidence or information. It is derived from conditional probability and can be implemented in Python using a function.

Example:

In a city, 55% of the adults are males. 10% of males use credit card, whereas 2% of females use credit card. One adult is randomly selected for a survey about usage of credit card. Find the probability that the randomly selected person is a male given that this selected person uses a credit card.

Code:

L8_BayesTheorem.py | L10_Bernoulli.py | L10_Binomial.py | L10_Geometric.py

L8_BayesTheorem.py > ...

```python
#In a city, 55% of the adults are males. 10% of males use credit card, whereas 2%
#adult is randomly selected for a survey about usage of credit card. Find the pro
#is a male given that this selected person uses a credit card

from TotalProbability import *
def bayesTheorem(pA, pB, pBA):
    return pA * pBA / pB

a1_h1 = Node('A1', 0.1)
a2_h1 = Node('A2', 0.9)
h1 = Node('H1', 0.55, [ a1_h1, a2_h1 ])
a1_h2 = Node('A1', 0.02)
a2_h2 = Node('A2', 0.98)
h2 = Node('H2', 0.45, [ a1_h2, a2_h2 ])

p_M = 0.55
p_M_C = 0.1
p_M_N_C = 0.9

c_probability = probability_of_event([h1, h2], 'A1') #calculate the total probabi
p_C_M = bayesTheorem(p_M_C, c_probability, p_M)
print(p_C_M)
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\UNi\PME\Python> python -u "d:\UNi\PME\Python\L8_BayesTheorem.py"
0.8593750000000001
PS D:\UNi\PME\Python>
```

6. Bernoulli Distribution:

The Bernoulli distribution models a random experiment with two possible outcomes: success (usually denoted as 1) and failure (usually denoted as 0). It is characterized by a single parameter, p, which represents the probability of success. In Python, we can generate random samples from a Bernoulli distribution using the NumPy library.

Example:

Suppose we want to simulate the outcome of flipping a fair coin, where heads is considered a success. We can use the Bernoulli distribution to model this experiment.

Code:

L10_Bernoulli.py > ...

```python
from math import factorial
from math import pow

def Binomial(n, k,p, q):
    probabilities = []

    for k_val in range(k,n + 1):
        binomial_coefficient = factorial(n) // (factorial(k_val) * factorial(n - k_val))
        probability = binomial_coefficient * pow(p, k_val) * pow(q, n - k_val)
        probabilities.append(probability)
        print("For K =", k_val, "P = {:.4f}".format(probability))

n = int(input("Enter Value of n "))
k = int(input("Enter start value of k "))

p = float(input("Enter Value of success probability "))

Binomial(n, k, p, 1 - p)

```

Output:

```
PS D:\UNi\PME\Python> python -u "d:\UNi\PME\Python\L10_Bernoulli.py"
Enter Value of n 1
Enter start value of k 0
Enter Value of success probability 0.5
For K = 0 P = 0.5000
For K = 1 P = 0.5000
PS D:\UNi\PME\Python>
```

7. Binomial Distribution:

The binomial distribution describes the probability of obtaining a certain number of successes in a fixed number of independent Bernoulli trials. In Python, we can calculate the probabilities of different outcomes using the binomial coefficient and the formula for binomial distribution.

Example:

Suppose that a coin is tossed three times. If we assume that the tosses are independent and the probability of getting a heads is 0.4. Find the probabilities of 0, 1, 2 and 3 heads

Code:

L10_Binomial.py > ...

```python
#Suppose that a coin is tossed three times. If we assume that
#the tosses are independent and the probability of getting a
#heads is 0.4. Find the probabilities of 0, 1, 2 and 3 heads

from math import factorial
from math import pow

def Binomial(n, k,p, q):
    probabilities = []

    for k_val in range(k,n + 1):
        binomial_coefficient = factorial(n) // (factorial(k_val) * factorial(n - k_val))
        probability = binomial_coefficient * pow(p, k_val) * pow(q, n - k_val)
        probabilities.append(probability)
        print("For K =", k_val, "P = {:.4f}".format(probability))

n = int(input("Enter Value of n "))
k = int(input("Enter start value of k "))

p = float(input("Enter Value of success probability "))

Binomial(n, k, p, 1 - p)
```

Output:

```
PS D:\UNi\PME\Python> python -u "d:\UNi\PME\Python\L10_Binomial.py"
Enter Value of n 3
Enter start value of k 0
Enter Value of success probability 0.4
For K = 0 P = 0.2160
For K = 1 P = 0.4320
For K = 2 P = 0.2880
For K = 3 P = 0.0640
PS D:\UNi\PME\Python>
```

8. Geometric Distribution:

The geometric distribution is a probability distribution that models the number of trials needed to achieve the first success in a sequence of independent Bernoulli trials. It is often used to describe situations where a repeated experiment with two possible outcomes (success or failure) is performed until the first success occurs.

Example:

What is the probability that the coin has to be flipped i) 4 times ii) more than 4 times, for getting heads for the first #time? The probability of heads is 0.6 and the probability of tails is 0.4.

Code:

L10_Geometric.py > ...

```python
#What is the probability that the coin has to be flipped i) 4
#times ii) more than 4 times, for getting heads for the first
#time? The probability of heads is 0.6 and the probability of tails is 0.4.
def geometric_probability(m, p):
    probability = pow(1 - p, m - 1) * p
    return probability

# Case 1: Probability of getting heads for the first time after 4 coin flips
m = 4
p_heads = 0.6
prob_4_flips = geometric_probability(m, p_heads)
print("Probability of getting heads after 4 flips: {:.4f}" .format(prob_4_flips))

# Case 2: Probability of getting heads for the first time after more than 4 coin flips
prob_more_than_4_flips = pow(1 - p_heads, m)
print("Probability of getting heads after more than 4 flips: {:.4f}".format(prob_more_than_4_flips))
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\UNi\PME\Python> python -u "d:\UNi\PME\Python\L10_Geometric.py"
Probability of getting heads after 4 flips: 0.0384
Probability of getting heads after more than 4 flips: 0.0256
PS D:\UNi\PME\Python>
```