Q.1 Birthday Paradox

*# Practical-1: Birthday Paradox*

*# Name: Angat Shah*

*# Enrollment No: 202203103510097*

*# Branch: B.Tech Computer Science and Engineering*

def birthdayParadox(n):

probability = 1

for i in range(n):

probability \*= (365 - i) / 365

commonProbability = 1 - probability

return commonProbability

numPeople = int(input("-->> Enter the Number of People in the Group: "))

result = birthdayParadox(numPeople)

print("\n--> Probability of at least Two People Sharing a Birthday in the Group of {} Individuals: {:.8f} or {:.4f}%\n".format(numPeople, result, (result \* 100)))

print("\n-\*-\*-\*-\*-\*-END OF PRACTICAL 1-\*-\*-\*-\*-\*-\n")

Q.2 The aim of this code is to simulate coin tosses and calculate the probabilities of getting heads and tails based on the number of tosses specified by the user.

*# Practical-2: The aim of this code is to simulate coin tosses and calculate the probabilities of getting heads and tails based on the number of tosses specified by the user.*

*# Name: Angat Shah*

*# Enrollment No: 202203103510097*

*# Branch: B.Tech Computer Science and Engineering*

import random

import numpy as np

import matplotlib.pyplot as plt

from scipy.stats import norm

def singleToss():

return random.choice(["H", "T"]) == "H"

def simulateTrial():

numHeads = sum(1 for \_ in range(10) if singleToss())

return numHeads / 10

def main():

numTrials = int(input("-->> Enter the number of Trials (Each Trial consist of 10 flips of a coin): "))

probabilities = [simulateTrial() for \_ in range(numTrials)]

meanProbability = np.mean(probabilities)

print(f"\n--> Probability of Getting a Head After {numTrials} trials of 10 Coin Tosses: {meanProbability} or {(meanProbability \* 100):.4f}%")

plt.xlabel('Probability of Heads')

plt.ylabel('Frequency')

plt.title('Distribution of Probability of Heads')

mu, std = np.mean(probabilities), np.std(probabilities)

xmin, xmax = plt.xlim()

x = np.linspace(xmin, xmax, 100)

p = norm.pdf(x, mu, std)

plt.plot(x, p, 'k', linewidth=2)

plt.show()

main()

print("\n-\*-\*-\*-\*-\*-END OF PRACTICAL 2-\*-\*-\*-\*-\*-\n")

Q.3 To check whether the email is spam or not spam there are 100 emails.

(1) 40 are label spam and 60 are label not spam.

(2) the spam mails are not spam considered based on single feature word “offer” . Out of 40 spam mails 30 contained the word ‘offer’ and out of 60 non-spam mails 5 contained the word ‘offer’.

*# Practical-3: To check whether the email is spam or not spam there are 100 emails.*

*# (1) 40 are label spam and 60 are label not spam.*

*# (2) the spam mails are not spam considered based on single feature word “offer” . Out of 40 spam mails 30 contains word ‘offer’ and out of 60 not spam mails 5 contained word ‘offer’.*

*# Name: Angat Shah*

*# Enrollment No: 202203103510097*

*# Branch: B.Tech Computer Science and Engineering*

def conditionalProbability(pSpam, pOfferGivenSpam):

pSpamGivenOffer = (pOfferGivenSpam \* pSpam) / pOffer

return pSpamGivenOffer

totalEmails = 100

spamEmails = 40

nonSpamEmails = totalEmails - spamEmails

offerInSpam = 30

offerInNonSpam = 5

pSpam = float(spamEmails) / totalEmails

pNonSpam = float(nonSpamEmails) / totalEmails

pOfferGivenSpam = float(offerInSpam) / spamEmails

pOfferGivenNonSpam = float(offerInNonSpam) / nonSpamEmails

pOffer = (offerInSpam + offerInNonSpam) / float(totalEmails)

result1 = conditionalProbability(pSpam, pOfferGivenSpam)

print("--> Probability of an Email Having the Keyword 'offer' being a Spam Email: {:.4f} or {:.2f}".format(result1,(result1\*100)))

result2 = conditionalProbability(pNonSpam, pOfferGivenNonSpam)

print("\n--> Probability of an Email having the Keyword 'offer' not being a Spam Email: {:.4f} or {:.2f}".format(result2,(result2\*100)))

print("\n-\*-\*-\*-\*-\*-END OF PRACTICAL 3-\*-\*-\*-\*-\*-\n")

Q.4 A supermarket wants to optimize its checkout process to minimize customer wait times and ensure efficient allocation of cashiers. Management has observed that customer arrivals follow a Poisson distribution with an average rate of 10 customers per hour. They want to determine the optimal number of checkout counters to open to minimize customer wait times and maximize resource utilization.

*# Practical-4: A supermarket wants to optimize its checkout process to minimize customer wait times and ensure efficient allocation of cashiers. Management has observed that customer arrivals follow a Poisson distribution with an average rate of 10 customers per hour. They want to determine the optimal number of checkout counters to open to minimize customer wait times and maximize resource utilization.*

*# Name: Angat Shah*

*# Enrollment No: 202203103510097*

*# Branch: B.Tech Computer Science and Engineering*

import random

import math

def generateCustomerArrivals(avgArrivalRate, simulationTime):

arrivals = []

totalArrivals = 0

time = 0

while time < simulationTime:

interArrivalTime = -1 / avgArrivalRate \* math.log(random.random())

time += interArrivalTime

if time < simulationTime:

arrivals.append(time)

totalArrivals += 1

else:

break

return arrivals

def generateExponential(avgServiceRate):

rand = random.random()

return -math.log(1 - rand) / avgServiceRate

def simulateCheckoutProcess(numCounters, avgServiceRate, arrivalTimes):

checkoutCounters = [[] for \_ in range(numCounters)]

waitTimes = []

for arrivalTime in arrivalTimes:

minWaitCounter = min(range(numCounters), key=lambda i: len(checkoutCounters[i]))

if checkoutCounters[minWaitCounter]:

lastCheckoutTime = max(checkoutCounters[minWaitCounter])

waitTime = max(0, lastCheckoutTime - arrivalTime)

else:

waitTime = 0

waitTimes.append(waitTime)

serviceTime = generateExponential(avgServiceRate)

checkoutTime = arrivalTime + waitTime + serviceTime

checkoutCounters[minWaitCounter].append(checkoutTime)

return waitTimes, checkoutCounters

def evaluatePerformance(waitTimes, checkoutCounters, simulationTime):

totalWaitTime = sum(waitTimes)

avgWaitTime = totalWaitTime / len(waitTimes)

totalServiceTime = sum(

[max(counter, default=simulationTime) - min(counter, default=simulationTime) for counter in checkoutCounters]

)

utilization = totalServiceTime / (len(checkoutCounters) \* simulationTime)

return avgWaitTime, utilization

def optimizeCheckoutProcess(avgArrivalRate, avgServiceTime, simulationTime):

avgServiceRate = 1 / avgServiceTime

bestNumCounters = None

minAvgWaitTime = float('inf')

maxUtilization = float('-inf')

for numCounters in range(1, 11):

arrivalTimes = generateCustomerArrivals(avgArrivalRate, simulationTime)

waitTimes, checkoutCounters = simulateCheckoutProcess(numCounters, avgServiceRate, arrivalTimes)

avgWaitTime, utilization = evaluatePerformance(waitTimes, checkoutCounters, simulationTime)

if avgWaitTime < minAvgWaitTime:

minAvgWaitTime = avgWaitTime

maxUtilization = utilization

bestNumCounters = numCounters

return bestNumCounters, minAvgWaitTime, maxUtilization

def main():

avgArrivalRate = 10

avgServiceTime = 4

avgServiceRate = avgServiceTime / 60

simulationTime = 8

bestNumCounters, minAvgWaitTime, maxUtilization = optimizeCheckoutProcess(avgArrivalRate,avgServiceRate,simulationTime)

print("--> Optimal Number of Checkout Counters: {}".format(bestNumCounters))

print("--> Minimum Average Customer Wait Time: {}".format(minAvgWaitTime))

print("--> Optimal Utilization of Checkout Counters at Peak Capacity.: {}\n".format(maxUtilization))

main()

print("\n-\*-\*-\*-\*-\*-END OF PRACTICAL 4-\*-\*-\*-\*-\*-\n")