中山大学计算机学院

机器学习与数据挖掘

本科生实验报告

（2023学年秋季学期）

课程名称：Machine Learning

教学班级	专业(方向)	学号	姓名
计科二班	计算机科学与技术	21307185	张礼贤

EX1

题目分析

该练习要求使用蒙特卡洛方法对PI值进行模拟，即在单位正方形内投掷一定数量的点，统计落在距离原点为1的所有点，将统计得到的数量与总的点数相除，便可以毕竟PI/4的值，然后再×4便得到PI的值

代码展示

```
import random
import numpy as np
import math

times = 100 # 训练次数
nums = [20, 50, 100, 200, 300, 500, 1000, 5000] #投点个数数组
count = 0

for num in nums:
    nums_of_pi = []
    for t in range(times):
        count = 0
        for i in range(num):
            x = random.random()
            y = random.random()
            if(x**2 + y**2 <= 1):
                count += 1
        PI = count * 4 / num
        nums_of_pi.append(PI)
    print(f"With the node number: {num} \n\
        The mean value is : {np.mean(nums_of_pi):.4f},\n\
        The var is : {np.var(nums_of_pi) :.4f}\n\
        The loss is {100 * abs(math.pi - np.mean(nums_of_pi)) / math.pi :.4f}%")
```

实验结果分析

- 通过运行上面的代码，得到下表的统计结果：

投点个数	模拟值(mean)	方差(var)	误差
20	3.1140	0.1370	0.8783%
50	3.1936	00440	1.6554%

投点个数	模拟值(mean)	方差(var)	误差
100	3.1108	0.0317	0.9802%
200	3.1432	0.0131	0.0520%
300	3.1305	0.0101	0.3520%
500	3.1264	0.0072	0.4836
1000	3.1389	0.0035	0.0863
5000	3.1408	0.0006	0.0265%

- 通过上面的统计结果，可以看到，整体趋势是随着点数的增加，统计的结果越趋近于pi值，结果数组中的方差也随之下降，说明数据趋于稳定。但是由于模拟的随机性，可能会出现点数增大但是更加偏离正确的结果的情况。

EX2

题目分析

该练习的求解思路与第一问相近，只是统计的界限发生了变化，上一题是通过圆的边界进行界定，这一问是通过三次函数的y值界定，即将上一题的 radius < 1 条件改为 y < x**3 即可，剩余的步骤与第一题无异

代码展示

```
import random
import numpy as np
times = 100 # 训练次数
samples = [5, 10, 20, 30, 40, 50, 60, 70, 80, 100] #采样
次数
count = 0

for sample in samples:
    res = []
    for t in range(times):
        count = 0
        for i in range(sample):
            x = random.random()
            y = random.random()
            if(y <= x**3):
                count += 1
        value = count / sample
        res.append(value)
    print(f"With the node number: {sample} \n\
        The mean value is : {np.mean(res):.4f},\n\
        The var is : {np.var(res) :.4f}\n\
        The loss is {100 * abs(1 / 4 - np.mean(res)) /
0.25 :.4f}%")
```

实验结果分析

- 根据上面的代码统计得到如下表格:

投点个数	模拟值(mean)	方差(var)	误差
5	0.2560	0.0409	2.4000%
10	0.2410	0.0212	3.6000%
20	0.2535	0.0111	1.4000%
30	0.2413	0.0054	3.4667%
40	0.2470	0.0042	1.2000%
50	0.2368	0.0035	5.2800%
60	0.2530	0.0034	1.2000%
70	0.2521	0.0025	0.8571%
80	0.2432	0.0023	2.7000%
100	0.2469	0.0021	1.2400%

- 通过上面统计表格的结果可以得出，整体的趋势是随着投点个数的增加，逐渐向0.25逼近，并且由于模拟的随机性，会出现误差的浮动。

EX3

题目分析

与前两题不同的是，本题的模拟是基于二重积分，因此需要在一定范围的体积内模拟计算，即通过预估z该函数z轴的最大值，将整个函数界定在长方体内，在该长方体内投放随机的点，统计落在该函数界定的面下的点。得到落点的概率，再与长方体相乘，得到对该积分的粗略估计

代码展示

```
import random
import numpy as np
import math
from scipy.integrate import dblquad

times = 100 # 训练次数
samples = [10, 20, 30, 40, 50, 60, 70, 80, 100, 200, 500]
#采样次数
count = 0

x_low_bound = 2
y_low_bound = -1
x_up_bound = 4
y_up_bound = 1
z_max = 3**15    # 利用求偏导对上界进行粗略估计，为了方便采用
整数作为底

def func(x, y):
    return (y**2 * math.e**(-(y**2)) + x**4 * math.e**(-
(x**2))) / (x * math.e**(-(x**2)))

# 通过调用py内置的求二重积分的函数进行求解
result, _ = dblquad(func, x_low_bound, x_up_bound, lambda
x: y_low_bound, lambda x: y_up_bound)
print(f"The standard result is:{result}")

base_area = (y_up_bound - y_low_bound) * (x_up_bound -
x_low_bound) * z_max
for sample in samples:
    res = []
    for t in range(times):
        count = 0
        for i in range(sample):
            x = random.uniform(x_low_bound, x_up_bound)
            y = random.uniform(y_low_bound, y_up_bound)
            z = random.uniform(0, z_max)
            if( z <= func(x, y)):
                count += 1
        value = count / sample # 求出落点分布在函数中的概率
        res.append(value * base_area) # 将概率×基本体积，
得到积分的值
```

```
    print(f"With the node number: {sample} \n\
        The mean value is : {np.mean(res)}\n\
        The var is : {np.var(res)}\n\
        The loss is : {abs(result - np.mean(res)) /
result}")
```

实验结果分析

- 通过上面的代码，可以得出下面的统计表格：

投点个数	模拟值(mean)	方差(var)	误差
5.0000	114791.2560	1304526212951.6973	0.0162
10.0000	172186.8840	958629111032.6859	0.5243
20.0000	57395.6280	161418647562.2049	0.4919
30.0000	19131.8760	36236839248.6582	0.8306
40.0000	172186.8840	258599261910.8792	0.5243
50.0000	103312.1304	107919895798.7312	0.0854
60.0000	105225.3180	107886953217.5961	0.0685
70.0000	122990.6314	99163892192.5249	0.0888
80.0000	71744.5350	46325504721.2960	0.3649
100.0000	160707.7584	105943340930.6226	0.4227
200.0000	117661.0374	28157671225.2642	0.0416
500.0000	117087.0811	10009273852.1021	0.0365

- 分析上面的统计数据，可以看出随着投点数目的增加，误差有逐步减少的趋势，但是受随机投点的影响和样本量的不足，会导致误差出现随机浮动现象，并且再同一投点个数下模拟值极其不稳定，体现为方差的异常大

EX4

题目分析

- 每次模拟中，蚂蚁从点A(1, 1)开始，按照随机的方式选择上、下、左、右四个方向之一前进，直到达到点B(n, n)或者无路可走。在每一次模拟中，我们记录蚂蚁是否成功到达点B，并计算成功到达点B的概率。
- 为了满足问题中的条件(a)和(b)，我们需要在每次模拟中保持追踪蚂蚁所访问的位置，并确保它们没有访问除(4, 4)之外的位置超过一次。如果蚂蚁访问了(4, 4)超过两次或访问了其他位置两次以上或超出了边界，则不应该选择该方向

代码展示

```
import random
import numpy as np

def monte_carlo_simulation(n, num_simulations):
    success_count = 0  # 记录成功到达点B的次数

    for _ in range(num_simulations):
        ant_x, ant_y = 1, 1  # 蚂蚁的初始位置
        visited = set()  # 记录已经访问过的位置
        visited.add((ant_x, ant_y))
        visit_4_4 = 0 # 对位置(4,4)进行特殊判定
        while True:
            # 生成可选择的方向数组
            choices = ['up', 'down', 'left', 'right']
            valid_choices = [] # 合法的移动方向

            for c in choices:
                temp_x,temp_y= ant_x,ant_y
                if c == 'up':temp_x -= 1
                elif c == 'down':temp_x += 1
                elif c == 'left':temp_y -= 1
                elif c == 'right' :temp_y += 1

                if temp_x >= 1 and temp_x <= n and temp_y
>= 1 and temp_y <= n:
                    if((temp_x, temp_y) == (4, 4) and
visit_4_4 <= 1):
                        valid_choices.append(c)
                    elif (temp_x, temp_y) != (4, 4) and
(temp_x, temp_y) not in visited:
                        valid_choices.append(c)

            # 没有合法的移动方向时退出
            if len(valid_choices) == 0: break
            direction = random.choice(valid_choices)
            if direction == 'up':
                ant_x -= 1
            elif direction == 'down':
                ant_x += 1
            elif direction == 'left':
                ant_y -= 1
```

```
            elif direction == 'right':
                ant_y += 1

            if((ant_x, ant_y) == (4, 4)):visit_4_4 += 1

            # 将当前位置添加到已访问集合中
            visited.add((ant_x, ant_y))
            # 如果蚂蚁成功到达点B，增加成功次数并退出循环
            if (ant_x, ant_y) == (n, n):
                success_count += 1
                break

    # 计算概率P
    probability = success_count / num_simulations
    return probability

# 调用模拟函数
n = 7
num_simulations = 20000

times = 10
res = []
for i in range(times):
    result = monte_carlo_simulation(n, num_simulations)
    res.append(result)
print(res,np.mean(res))
```

实验结果分析

- 通过运行上面的代码，得到结果如下：

  [0.2574, 0.2545, 0.25615, 0.25395, 0.2572, 0.2547, 0.2589, 0.2545, 0.25105, 0.2565] 0.255485

- 通过对结果的分析，可以看到在随机模拟的时候，蚂蚁到达终点的概率接近于1 / 4

EX5

题目分析

题目中说进入A通路和进入BC通路有相同的概率，但是实际情况时满足其中一条通路即可，因此在这里给出严格按照题意的解题方法和实际模拟的情况

代码展示

```
import random

'''模拟函数'''
def simulate():
    if random.random() < 0.5: # 一半的概率进入组件A
        if random.random() < 0.85:# 表示A组件成功
            return True
        else :
            return False
    else :  # 另一半时间进入组件B
        if random.random() < 0.95 and random.random() <
0.9: # 表示BC组件成功
            return True
        else :
            return False

'''改进后的模拟函数'''
def simulate_change():
    if random.random() < 0.85:# 表示A组件成功
        return True
    if random.random() < 0.95 and random.random() < 0.9:
# 表示BC组件成功
        return True
    else :
        return False

simu_times = 10000
succ_times = 0
for i in range(simu_times):
    if simulate_change() == True:
        succ_times += 1

print(f"The reliability of the system simulated is:
{succ_times / simu_times:.4f}")
```

实验结果分析

- 如果按照题目中所说，一半时间在A一半时间在BC，因此需要对两条通路进行等概率取样，因此最终得到的结果为:

```
The reliability of the system simulated is:0.8573
```

- 但是如果按照真实的数理逻辑来实现，当A失败时，应该还可以选择BC，两次都失败了才能判定为false，因此最终结果为：

```
The reliability of the system simulated is:0.9792
```