Python: without numpy or sklearn

Q1: Given two matrices please print the product of those two matrices

```
Ex 1: A   = [[1 3 4]
             [2 5 7]
             [5 9 6]]
      B   = [[1 0 0]
             [0 1 0]
             [0 0 1]]
      A*B = [[1 3 4]
             [2 5 7]
             [5 9 6]]


Ex 2: A   = [[1 2]
             [3 4]]
      B   = [[1 2 3 4 5]
             [5 6 7 8 9]]
      A*B = [[11 14 17 20 23]
             [23 30 36 42 51]]

Ex 3: A   = [[1 2]
             [3 4]]
      B   = [[1 4]
             [5 6]
             [7 8]
             [9 6]]
      A*B =Not possible
```

In [34]:

```python
from operator import mul

# here A and B are list of lists
def matrix_mul(A, B):
    # transpose of a matrix B to get column at a time
    *B_t, = zip(*B)
    # calculate mat-mul i.e row_of_A * col_of_B at a time
    return [[sum(map(mul, r_a, c_b)) for c_b in B_t]  for r_a in A]

A = [[1,2], [3,4]]
B = [[1,2,3,4,5], [5,6,7,8,9]]
matrix_mul(A, B)
```

Out[34]:

```
[[11, 14, 17, 20, 23], [23, 30, 37, 44, 51]]
```

Q2: Select a number randomly with probability proportional to its magnitude from the given array of n elements

consider an experiment, selecting an element from the list A randomly with probability proportional to its magnitude. assume we are doing the same experiment for 100 times with replacement, in each experiment you will print a number that is selected randomly from A.

```
Ex 1: A = [0 5 27 6 13 28 100 45 10 79]
let f(x) denote the number of times x getting selected in 100
experiments.
f(100) > f(79) > f(45) > f(28) > f(27) > f(13) > f(10) > f(6) >
f(5) > f(0)
```

In [33]:

```python
from random import random
from itertools import accumulate

def ceil(l, target):
    '''
        Compute the interval upper-bound for target via binary search
    '''
    s = len(l)
    start, end = 0, s - 1
    while start <= end:
        mid = start + ((end - start) // 2)
        if target == l[mid]:
            return mid
        if target > l[mid]:
            start = mid + 1
        else:
            end = mid - 1
    return start

def pick_a_number_from_list(lst):
    '''
        Proportional Sampling Case
    '''
    # step1 - sum of all numbers
    s = sum(lst)
    # step2 - normalize all numbers
    n_lst = [e/s for e in lst]
    # step3 - accumulate
    *cum_norm_sum, = accumulate(n_lst)
    # step4 - find proper ceil idx
    idx = ceil(cum_norm_sum, random())
    selected_random_number = lst[idx]

    return selected_random_number

def sampling_based_on_magnitued(l, ntimes):
    '''
        Proportional sampling via theoretical formulation via cumulative sums
    '''
    d = dict.fromkeys(l, 0) # to keep track of freq of each member during tes
    r = ntimes  # times to perform test
    for i in range(1,r+1): # test
        number = pick_a_number_from_list(A)
        d[number] += 1
    print('Probabilities :')

    pairs = sorted(d.items(), key=lambda x: x[1], reverse=True)
    for pair in pairs:
        print(pair[0], '-> ', format(pair[1]*100/r, '.2f'), '%')

A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
sampling_based_on_magnitued(A, 100000)
```

```
Probabilities :
100 ->   31.78 %
79 ->   25.41 %
45 ->   14.42 %
```

```
28 ->  8.84 %
27 ->  8.58 %
13 ->  4.23 %
10 ->  3.21 %
6 ->  1.91 %
5 ->  1.63 %
0 ->  0.00 %
```

Q3: Replace the digits in the string with

consider a string that will have digits in that, we need to remove all the not digits and replace the digits with #

```
Ex 1: A = 234                 Output: ###
Ex 2: A = a2b3c4              Output: ###
Ex 3: A = abc                 Output:   (empty string)
Ex 5: A = #2a$#b%c%561#       Output: ####
```

In [29]:

```
import re

def replace_digits(String):
    pattern = r"[^\d]"
    repl = ""
    result = re.sub(pattern, repl, String, 0)
    return '#'*len(result)

A = '234'
replace_digits(A)
```

Out[29]:

```
'###'
```

Q4: Students marks dashboard

consider the marks list of class students given two lists

Students = ['student1','student2','student3','student4','student5','student6','student7','student8','student9','stu

Marks = [45, 78, 12, 14, 48, 43, 45, 98, 35, 80]

from the above two lists the Student[0] got Marks[0], Student[1] got Marks[1] and so on

your task is to print the name of students **a. Who got top 5 ranks, in the descending order of marks**

b. Who got least 5 ranks, in the increasing order of marks

d. Who got marks between >25th percentile <75th percentile, in the increasing order of marks

```
Ex 1:
Students=
['student1','student2','student3','student4','student5','student6','st

Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
a.
student8  98
student10 80
student2  78
student5  48
```

```
student7  47
b.
student3 12
student4 14
student9 35
student6 43
student1 45
c.
student9 35
student6 43
student1 45
student7 47
student5 48
```

In [36]:

```python
def ceil(l, target):
    '''
        Compute the interval upper-bound for target via binary search
    '''
    s = len(l)
    start, end = 0, s - 1
    while start <= end:
        mid = start + ((end - start) // 2)
        if target == l[mid]:
            return mid
        if target > l[mid]:
            start = mid + 1
        else:
            end = mid - 1
    return start

def floor(l, target):
    '''
        Compute the interval upper-bound for target via binary search
    '''
    s = len(l)
    start, end = 0, s - 1
    while start <= end:
        mid = start + ((end - start) // 2)
        if target == l[mid]:
            return mid
        if target > l[mid]:
            start = mid + 1
        else:
            end = mid - 1

    return end

def display_dash_board(students, marks):
    size = len(marks)
    # list of indices based on sorted marks
    marks_argi = sorted(range(size), key=marks.__getitem__)

    # write code for computing top top 5 students
    top_5_students = [students[i] for i in marks_argi[-1:-6:-1]]
    # write code for computing top least 5 students
    least_5_students = [students[i] for i in marks_argi[:5]]
    # write code for computing top least 5 students
    low, high = marks[marks_argi[0]], marks[marks_argi[-1]]
    d = high - low
    l = (0.25 * d) + low  # 25th percentile
    h = (0.75 * d) + low  # 75th percentile
    marks_s = [marks[i] for i in marks_argi]
```

```
        idx_l, idx_u = ceil(marks_s, l), floor(marks_s, h) # lower & upper bound
        students_within_25_and_75 = [students[i] for i in marks_argi[idx_l:idx_u+

        return top_5_students, least_5_students, students_within_25_and_75

    Students=['student1','student2','student3','student4','student5','student6','
    Marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
    display_dash_board(Students, Marks)
```

Out[36]:

```
([ 'student8', 'student10', 'student2', 'student5', 'student7'],
 ['student3', 'student4', 'student9', 'student6', 'student1'],
 ['student9', 'student6', 'student1', 'student7', 'student5'])
```

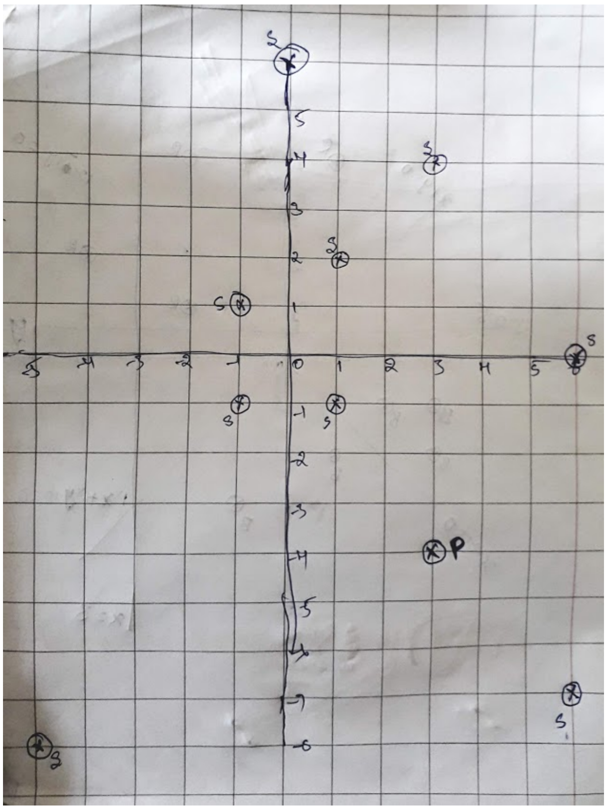
Q5: Find the closest points

consider you have given n data points in the form of list of tuples like S=[(x1,y1),(x2,y2),(x3,y3),(x4,y4),(x5,y5),..,(xn,yn)] and a point P=(p,q)
your task is to find 5 closest points(based on cosine distance) in S from P

cosine distance between two points (x,y) and (p,q) is defind as $cos^{-1}(\frac{(x\cdot p+y\cdot q)}{\sqrt{(x^2+y^2)}\cdot\sqrt{(p^2+q^2)}})$

```
Ex:

S= [(1,2),(3,4),(-1,1),(6,-7),(0, 6),(-5,-8),(-1,-1)(6,0),
(1,-1)]
P= (3,-4)
```



```
Output:
(6,-7)
(1,-1)
(6,0)
(-5,-8)
(-1,-1)
```

In [16]:

```
import math
import itertools
```

```
def calc_cos_sim(x,y):
    '''
    calculate the cosine similarity (ie in terms of anngle diff) between 2 ve
    NOTE : return metric is in radian & not angle for simplicity
    '''
    n = x[0]*y[0] + x[1]*y[1]
    d = math.sqrt((x[0]**2 + x[1]**2) * (y[0]**2 + y[1]**2))
    cos_sim = math.acos(n/d)
    return cos_sim

# here S is list of tuples and P is a tuple ot len=2
def closest_points_to_p(S, P):
    cos_dist = [round(calc_cos_sim(X,P),2) for X in S] # cos-sim between p &
    # get first closest point to P
    closest_points_to_p = itertools.islice(map(S.__getitem__, sorted(range(le
    return closest_points_to_p  # its list of tuples

S= [(1,2),(3,4),(-1,1),(6,-7),(0,6),(-5,-8),(-1,-1),(6,0),(1,-1)]
P= (3,-4)
*points, = closest_points_to_p(S, P)
print(points) #print the returned values
```

```
[(6, -7), (1, -1), (6, 0), (-5, -8), (-1, -1)]
```

Q6: Find Which line separates oranges and apples

consider you have given two set of data points in the form of list of tuples like

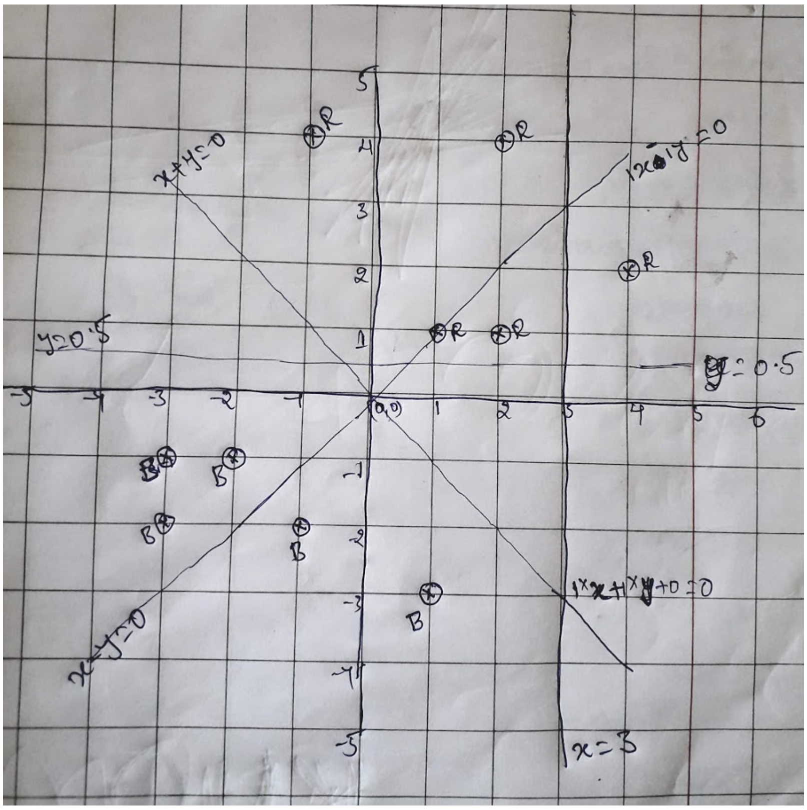
```
Red =[(R11,R12),(R21,R22),(R31,R32),(R41,R42),(R51,R52),..,
(Rn1,Rn2)]
Blue=[(B11,B12),(B21,B22),(B31,B32),(B41,B42),(B51,B52),..,
(Bm1,Bm2)]
```

and set of line equations(in the string formate, i.e list of strings)

```
Lines = [a1x+b1y+c1,a2x+b2y+c2,a3x+b3y+c3,a4x+b4y+c4,..,K lines]
Note: you need to string parsing here and get the coefficients
of x,y and intercept
```

your task is to for each line that is given print "YES"/"NO", you will print yes, if all the red points are one side of the line and blue points are other side of the line, otherwise no

```
Ex:
Red= [(1,1),(2,1),(4,2),(2,4), (-1,4)]
Blue= [(-2,-1),(-1,-2),(-3,-2),(-3,-1),(1,-3)]
Lines=["1x+1y+0","1x-1y+0","1x+0y-3","0x+1y-0.5"]
```



```
Output:
YES
NO
NO
YES
```

In [17]:

```python
import math
import re
from itertools import zip_longest
from operator import mul
import itertools as it

def extract_weights(s: str):
    ''' return the coefficients of eqn of form ax+by+c '''
    # TODO make pattern for nD general
    pattern = r"(.+)x(.+)y(.+)"  # for 2D only
    m = re.match(pattern, s)
    return list(map(eval, [m.group(1), m.group(2), m.group(3)])) if m else No

def extract_coeff(s: str):
    pattern = r'[a-zA-Z]+'
    s = re.sub(pattern, " ", s)
    return list(map(eval, s.split()))

def fit_to_line_sign(pt, eqn):
    ''' Apply the eqn to pt & return the sign of magnitude '''
    v = sum(it.starmap(mul, zip_longest(eqn, pt, fillvalue=1)))
    return math.copysign(1, v)

def i_am_the_one(red,blue,line):
    # extract the coefficients from line equation [a, b, c]
    w = extract_weights(line)

    # get only signs of eqn fit for all pts belonging to red
    red_signs = map(lambda r: fit_to_line_sign(r, w), red)
    sign = 0
    for i in red_signs:
        if not sign and i:
            sign = i
        elif not (i + sign): # conflicts in red grp
            return "NO"

    # get only signs of eqn fit for all pts belonging to blue
    blue_signs = map(lambda b: fit_to_line_sign(b, w), blue)
    sign = 0
    for i in blue_signs:
        if not sign and i:
            sign = i
        elif not (i + sign): # conflicts in red grp
            return "NO"

    return "YES"

Red= [(1,1),(2,1),(4,2),(2,4),(-1,4)]
Blue= [(-2,-1),(-1,-2),(-3,-2),(-3,-1),(1,-3)]
Lines=["1x+1y+0","1x-1y+0","1x+0y-3","0x+1y-0.5"]

for i in Lines:
    yes_or_no = i_am_the_one(Red, Blue, i)
    print(yes_or_no) # the returned value
```

```
YES
NO
NO
YES
```

Q7: Filling the missing values in the specified formate

You will be given a string with digits and '_'(missing value) symbols you have to replace the '_' symbols as explained

```
Ex 1: _, _, _, 24 ==> 24/4, 24/4, 24/4, 24/4 i.e we. have
distributed the 24 equally to all 4 places

Ex 2: 40, _, _, _, 60 ==> (60+40)/5,(60+40)/5,(60+40)/5,
(60+40)/5,(60+40)/5 ==> 20, 20, 20, 20, 20 i.e. the sum of
(60+40) is distributed qually to all 5 places

Ex 3: 80, _, _, _, _  ==> 80/5,80/5,80/5,80/5,80/5 ==> 16, 16,
16, 16, 16 i.e. the 80 is distributed qually to all 5 missing
values that are right to it

Ex 4: _, _, 30, _, _, _, 50, _, _
==> we will fill the missing values from left to right
    a. first we will distribute the 30 to left two missing
values (10, 10, 10, _, _, _, 50, _, _)
    b. now distribute the sum (10+50) missing values in between
(10, 10, 12, 12, 12, 12, 12, _, _)
    c. now we will distribute 12 to right side missing values
(10, 10, 12, 12, 12, 12, 4, 4, 4)
```

for a given string with comma seprate values, which will have both missing values numbers like ex: "_, _, x, _, _, _" you need fill the missing values Q: your program reads a string like ex: "_, _, x, _, _, _" and returns the filled sequence Ex:

```
Input1: "_,_,_,24"
Output1: 6,6,6,6

Input2: "40,_,_,_,60"
Output2: 20,20,20,20,20

Input3: "80,_,_,_,_"
Output3: 16,16,16,16,16

Input4: "_,_,30,_,_,_,50,_,_"
Output4: 10,10,12,12,12,12,4,4,4
```

In [19]:

```python
def curve_smoothing(string):
    '''
    Goal :- Divide the left & right val to in between _
    Idea ;- left & right pointer with right as leading to left
    Time :- complexity O(n^2)
    '''
    # get each character by removing the commas
    s = string.split(',')
    size = len(s)

    if size <= 1: # edge case
        return string

    lp = 0    # left Pointer
    # Check At Start (Edge Case 1 ie '_' or num)
    if s[0] != '_': # decide left value
        lv = s[0] = int(s[0])
    else:
        lv = 0

    for i, c in enumerate(s[1:], 1):
        if c != '_' and i != lp+1:   # fill space required
            r = int(c)   # right value
            cnt = i - lp + 1 # no. of character needs to be altered
            v = (r + lv) // cnt   # value needs to be fill at @cnt places
            s[lp:i+1] = [v]*cnt # alteration
            lp, lv = i, v   # update left ptr & left val

    # Check At End (Edge Case 2 ie '_' or num)
    if lp != size-1: # if last character is '_'
        cnt = size-lp
        v = s[lp] // cnt
        s[lp:] = [v]*cnt

    return ','.join(map(str, s))

S=  "_,_,30,_,_,_,50,_,_"
smoothed_values= curve_smoothing(S)
print(smoothed_values)
```

```
10,10,12,12,12,12,4,4,4
```

Q8: Filling the missing values in the specified formate

You will be given a list of lists, each sublist will be of length 2 i.e. [[x,y],[p,q],[l,m]..[r,s]] consider its like a martrix of n rows and two columns

1. the first column F will contain only 5 uniques values (F1, F2, F3, F4, F5)

2. the second column S will contain only 3 uniques values (S1, S2, S3)

```
your task is to find
a. Probability of P(F=F1|S==S1), P(F=F1|S==S2), P(F=F1|S==S3)
b. Probability of P(F=F2|S==S1), P(F=F2|S==S2), P(F=F2|S==S3)
c. Probability of P(F=F3|S==S1), P(F=F3|S==S2), P(F=F3|S==S3)
d. Probability of P(F=F4|S==S1), P(F=F4|S==S2), P(F=F4|S==S3)
e. Probability of P(F=F5|S==S1), P(F=F5|S==S2), P(F=F5|S==S3)
```

Ex:

```
[[F1,S1],[F2,S2],[F3,S3],[F1,S2],[F2,S3],[F3,S2],[F2,S1],
[F4,S1],[F4,S3],[F5,S1]]

a. P(F=F1|S==S1)=1/4, P(F=F1|S==S2)=1/3, P(F=F1|S==S3)=0/3
b. P(F=F2|S==S1)=1/4, P(F=F2|S==S2)=1/3, P(F=F2|S==S3)=1/3
c. P(F=F3|S==S1)=0/4, P(F=F3|S==S2)=1/3, P(F=F3|S==S3)=1/3
d. P(F=F4|S==S1)=1/4, P(F=F4|S==S2)=0/3, P(F=F4|S==S3)=1/3
e. P(F=F5|S==S1)=1/4, P(F=F5|S==S2)=0/3, P(F=F5|S==S3)=0/3
```

In [20]:

```
from collections import defaultdict
from fractions import Fraction

# you can free to change all these codes/structure
def compute_conditional_probabilites(A):
    '''
        P(A|B) = P(A.intersect(B)) / P(B)
    '''

    dm = defaultdict(lambda : defaultdict(int))  # data matrix for f.intersec
    n = len(A) # rows

    freq_s = defaultdict(int) # freq cnt for second col

    s1 = set() # unique vals

    for r, c in A:  # compute necessary probabilities
        freq_s[c] += 1
        dm[r][c] += 1
        s1.add(r)

    for i in s1:
        for j in freq_s.keys():
            #ans = Fraction(dm[i][j], freq_s[j]) if dm[i][j] and freq_s[j] el
            if dm[i][j] and freq_s[j]:
                ans = Fraction(dm[i][j], freq_s[j])
            else:
                ans = f'{dm[i][j]}/{freq_s[j]}'

            print(f'P(F={i}|S=={j})={ans}', end= ', ')
        print()


A = [['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],

compute_conditional_probabilites(A)
```

```
P(F=F1|S==S1)=1/4, P(F=F1|S==S2)=1/3, P(F=F1|S==S3)=0/3,
P(F=F4|S==S1)=1/4, P(F=F4|S==S2)=0/3, P(F=F4|S==S3)=1/3,
P(F=F5|S==S1)=1/4, P(F=F5|S==S2)=0/3, P(F=F5|S==S3)=0/3,
P(F=F2|S==S1)=1/4, P(F=F2|S==S2)=1/3, P(F=F2|S==S3)=1/3,
P(F=F3|S==S1)=0/4, P(F=F3|S==S2)=1/3, P(F=F3|S==S3)=1/3,
```

Q9: Given two sentances S1, S2

You will be given two sentances S1, S2 your task is to find

```
a. Number of common words between S1, S2
b. Words in S1 but not in S2
c. Words in S2 but not in S1
```

Ex:

```
S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
Output:
a. 7
b. ['first','F','5']
c. ['second','S','3']
```

In [23]:

```python
def string_features(S1, S2):
    ''' find common & disjoint words '''
    pattern = r'\s+'
    re_c = re.compile(pattern)

    s1_words = set(re_c.split(S1))
    s2_words = set(re_c.split(S2))
    common = s1_words.intersection(s2_words)

    a = len(common)
    b = s1_words.difference(common)
    c = s2_words.difference(common)

    return a, b, c

S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
a,b,c = string_features(S1, S2)
print('a :', a)
print('b :', b)
print('c :', c)
```

```
a : 7
b : {'5', 'F', 'first'}
c : {'3', 'S', 'second'}
```

Q10: Given two sentances S1, S2

You will be given a list of lists, each sublist will be of length 2 i.e. [[x,y],[p,q],[l,m]..[r,s]] consider its like a martrix of n rows and two columns

a. the first column Y will contain interger values

b. the second column Y_{score} will be having float values
Your task is to find the value of
$f(Y, Y_{score}) = -1 * \frac{1}{n}\Sigma_{foreachY,Y_{score}pair}(Ylog10(Y_{score}) + (1-Y)log10(1-Y_{score}))$
here n is the number of rows in the matrix

```
Ex:
[[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1,
0.9], [1, 0.8]]
output:
0.4243099
```

$$\frac{-1}{8} \cdot ((1 \cdot log_{10}(0.4) + 0 \cdot log_{10}(0.6)) + (0 \cdot log_{10}(0.5) + 1 \cdot log_{10}(0.5)) + \ldots + (1 \cdot log_{10}(0.8)$$

In [26]:

```
from math import log10
def compute_log_loss(A):
    '''
        Formula := -1/n * sum(yi * log10(pi) + (1-yi) * log10(1-pi))
    '''
    if not A:
        return 0
    n = len(A)
    loss = (-1/n)*sum(((y * log10(p)) + ((1-y) * log10(1-p)) for y, p in A))
    return round(loss, 5)

A = [[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1, 0.9], [1
loss = compute_log_loss(A)
print(loss)
```

```
0.42431
```