Python Mandatory Assignment

December 20, 2020

Python: without numpy or sklearn

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

[1]:
```
def matrix_mul(A, B):
    """
    This function takes two matrices A and B as inputs and returns the resultant␣
 ↪matrix after multiplying them.
    """
    r_a = len(A)
    c_a = len(A[0])
    r_b = len(B)
    c_b = len(B[0])
    if c_a != r_b:
        return 0
    else:
        AB = [[0 for i in range(c_b)] for j in range(r_a)]
        for i in range(c_a):
            for j in range(r_b):
                for k in range(r_b):
                    AB[i][j] += A[i][k] * B[k][j]
        return(AB)

# Taking the rows and columns of both matrices as inputs
r_a, c_a = input("Enter the number of rows and columns of Matrix A: ").split()
r_b, c_b = input("Enter the number of rows and columns of Matrix B: ").split()

# Declaring three matrices,one for A, one for B and the remaining one that will␣
 ↪be used to take the rows as input
A = []
B = []
temp = []

# Verifying whether the columns of A and rows of B match. If they don't print␣
 ↪error and exit. Else take their inputs and multiply them.
if c_a != r_b:
    print("Multiplication of Matrices with given dimensions is not possible.")
else:
```

```
    # Taking A's values as inputs
    print("Enter the rows of Matrix A: ")
    for i in range(int(r_a)):
        temp = [int(i) for i in input().split(" ")]
        A.append(temp)

    # Taking B's values as inputs
    print("Enter the rows of Matrix B: ")
    for i in range(int(r_a)):
        temp = [int(i) for i in input().split(" ")]
        B.append(temp)

    # Calling the function matrix_mul and storing the result in AB and printing␣
 ↪it.
    AB = matrix_mul(A,B)
    print("The Multiplication of the given matrices is:")
    for i in AB:
        print(i)
```

```
Enter the number of rows and columns of Matrix A: 3 3
Enter the number of rows and columns of Matrix B: 3 3
Enter the rows of Matrix A:
1 0 0
0 1 0
0 0 1
Enter the rows of Matrix B:
1 2 3
4 5 6
7 8 9
The Multiplication of the given matrices is:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
```

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.

[2]:
```
import random

def pick_a_number_from_list(A):
    """
    This function takes a list as an input and returns a single value after␣
 ↪using proportional sampling
    """
```

```
    # Calculating the sum of all elements in the list
    sum_A = sum(A)
    # A_d is an array after normalizing the values using the sum
    A_d = [i/sum_A for i in A]
    # Creating the cumulative normalized sum using the elements of the list and␣
 ↪storing the elements in A_t
    temp = 0
    A_t = [A_d[0]]
    for i in range(1,len(A_d)):
        A_t.append(A_t[i-1]+A_d[i])
    # Getting a random_num between 0 and 1 with uniform sampling.
    random_num = random.uniform(0,1)
    # Using proportional sampling rule to get the value based on the probability
    for i in range(len(A_t)):
        if random_num <= A_t[i]:
            return A[i]

def sampling_based_on_magnitude():
    """
    This function picks a random value from the list A based on it's size 100␣
 ↪times.
    """
    for i in range(1,100):
        number = pick_a_number_from_list(A)
        # disp_res is a dictionary that keeps a track of the number of times a␣
 ↪number has been picked.
        disp_res[number] += 1

A = [0, 5, 27, 6, 13, 28, 100, 45, 10, 79]
disp_res = {i:0 for i in A}

sampling_based_on_magnitude()

disp_res
```

[2]:
```
{0: 0, 5: 1, 27: 13, 6: 1, 13: 7, 28: 12, 100: 28, 45: 14, 10: 2, 79: 21}
```

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 #

[3]:
```
def replace_digits():
    """
    This function takes an input and prints a # everytime a number is found in␣
 ↪the string by iterating over it.
    """
```

```
    # Taking the input and storing it in a variable called 'string'
    string = input()
    # nums is a list of all numbers
    nums = list("0123456789")
    # Iterating over the individual characters of the string and checking if it␣
 ↪exists in the list nums. If it does, print a '#'
    for i in range(len(string)):
        if string[i] in nums:
            print("#", end="")

replace_digits()
```

```
#2a$#b%c%561#
####
```

Q4: Students marks dashboard

consider the marks list of class students given two lists Students = ['student1','student2','student3','student4','student5','student6','student7','student8','student9','student10'] 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

[4]:
```
import math

def display_dash_board(students, marks):
    """
    This function takes two lists, students and marks, and returns the names and␣
 ↪marks of top 5, last 5 and 25th to 75th percentile.
    """
    # marks_stu is a dictionary containing the scores as keys and their original␣
 ↪indices as items.
    marks_stu = {score:index for index,score in enumerate(marks)}

    # sorting the marks in descending order and using the dictionary to get the␣
 ↪indices and using the indices to get the names.
    marks.sort(reverse=True)
    top_5_students = {students[marks_stu[i]]:i for i in marks[:5]}

    # sorting the marks in ascending order and using the dictionary to get the␣
 ↪indices and using the indices to get the names.
    marks.sort()
    least_5_students = {students[marks_stu[i]]:i for i in marks[:5]}

    # a and b are the 25th and 75th percentile values and a_i and b_i are the␣
 ↪indices of the values.
    # we use these indices to get all the values between them.
```

```
    a = sorted(marks)[int(math.ceil((10 * 25) / 100)) - 1]
    b = sorted(marks)[int(math.ceil((10 * 75) / 100)) - 1]
    a_i, b_i = marks.index(a), marks.index(b)
    students_within_25_and_75 = {students[marks_stu[i]]:i for i in marks[a_i:
 ↪b_i]}

    return top_5_students, least_5_students, students_within_25_and_75

students=['student1','student2','student3','student4','student5','student6','student7','student
marks = [45, 78, 12, 14, 48, 43, 47, 98, 35, 80]

top_5_students, least_5_students, students_within_25_and_75 =␣
 ↪display_dash_board(students, marks)
print("Top 5 Students\n", top_5_students)
print("Last 5 Students:\n", least_5_students)
print("Students Between 25th and 75th Percentile:\n", students_within_25_and_75)
```

```
Top 5 Students
 {'student8': 98, 'student10': 80, 'student2': 78, 'student5': 48, 'student7':
47}
Last 5 Students:
 {'student3': 12, 'student4': 14, 'student9': 35, 'student6': 43, 'student1':
45}
Students Between 25th and 75th Percentile:
 {'student9': 35, 'student6': 43, 'student1': 45, 'student7': 47, 'student5':
48}
```

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 $cos^{-1}(\frac{(x\cdot p+y\cdot q)}{\sqrt{(x^2+y^2)}\cdot\sqrt{(p^2+q^2)}})$

[5]:
```
import math

def closest_points_to_p(S, P):
    """
    This function takes two inputs and returns the 5 closest points to P.
    """
    # points_dict is a dictionary with points as keys and the distance from P as␣
 ↪the item. Initially all the items are intialized to 0.
    points_dict = {point:0 for point in S}
    # changing the items of keys to the distance from P
    for i in S:
        points_dict[i] = math.acos(((P[0]*i[0])+(P[1]*i[1]))/(math.
 ↪sqrt(i[0]**2+i[1]**2)*math.sqrt(P[0]**2+P[1]**2)))
```

```
    # getting the 5 closest points by sorting the items by ascending order and␣
 ↪storing them in a list
    closest_points = list(i[0] for i in sorted(points_dict.items(),key=lambda␣
 ↪item: item[1])[:5])
    return closest_points

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)
```

```
[(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

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

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

[6]:
```
import math
import re

def get_coeffs(line):
    """
    this function takes a line as an input and compares it with ax+by+c and␣
 ↪returns the values of a,b,c
    """
    # splitting the line at x and y
    line_coefs = re.split('x|y',line)
    # since 'a' is at the beginning of the line it doesnt have a sign associated␣
 ↪with it
    a = float(line_coefs[0])
    # if the first value of the the second string is a '+', ignore it and␣
 ↪convert the remaining values to float and store it as b
    if line_coefs[1][0] == "+":
        b = float(line_coefs[1][1:])
    # if it's a '-', convert the rest of it to float and multiply with -1
    else:
        b = float(line_coefs[1][1:]) * -1
    # using the same logic as that of b to get the value of c
    if line_coefs[2][0] == "+":
        c = float(line_coefs[2][1:])
    else:
        c = float(line_coefs[2][1:]) * -1
    # returning the values of a,b,c
```

```
    return a,b,c


def get_same_side(red,blue,line):
    """
    This function takes red and blue points and a line as the input and prints␣
 ↪"YES" or "NO" depending on where the lie on the side of the line.
    """
    # using the get_coeffs function to get the values of a, b and c.
    a,b,c = get_coeffs(line)
    # red_sign is a variable that stores the position of point w.r.t the line.␣
 ↪We choose a point that's not lying on the line i.e., red_sign = 0
    red_sign = 0
    for i in range(len(red)):
        if red[i][0]*a+red[i][1]*b+c == 0:
            red_sign = 0
        elif red[i][0]*a+red[i][1]*b+c > 0:
            red_sign = 1
            break
        else:
            red_sign = -1
            break
    # we now insert blue points into the line and if we get atleast one point on␣
 ↪the same side of red, we abort and print "NO"
    for point in blue:
        if point[0]*a+point[1]*b+c == 0:
            blue_sign = 0
        elif point[0]*a+point[1]*b+c > 0:
            blue_sign = 1
        else:
            blue_sign = -1
        if blue_sign == red_sign:
            return "NO"
    # if all of the points lie on different sides and function hasn't returned␣
 ↪"NO", we return "YES"
    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 = get_same_side(Red, Blue, i)
    print(yes_or_no)
```

```
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

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:

[7]:
```
def mod_list(vals,value,a_i,b_i):
    """
    This function takes a list, a value and two indices and replaces all values␣
 ↪between the two indices with the value provided
    and returns the list.
    """
    for i in range(a_i,b_i+1):
        vals[i] = value
    return vals

def curve_smoothing(string):
    """
    This function takes a string as an input and returns a list after smoothing␣
 ↪it's values
    Approach:
    Keep track of the last numerical value and it's index.
    Keep track of number of '_'s encountered.
    If a numerical value is seen, replace all values between the last seen␣
 ↪number and current number.
    Handle outlier cases.
    """
    # vals is a list containing either '_' or numbers in the same order as that␣
 ↪of the string.
    vals = [int(i) if len(i) > 1 else i for i in string.split(",")]

    # last_val keeps track of the last seen numerical value
    # last_index is the index of the last seen numerical value
    # u_count keeps track of the number of underscores between two numerical␣
 ↪values
    last_val, last_index, u_count = 0,-1,0

    # now we iterate over the list of values and use mod_list function when␣
 ↪necessary.
    for i in range(len(vals)):
        if vals[i]=='_':
```

```
            u_count += 1
        elif vals[i] != '_' and u_count != 0:
            if last_index != -1:
                # when there's no previous numerical value i.e., list starts␣
↪with _
                vals = mod_list(vals, (last_val+vals[i])/(u_count+2),␣
↪last_index, i)
                u_count = 0
            else:
                vals = mod_list(vals, vals[i]/(u_count+1), 0, i)
                u_count = 0
            last_val, last_index = vals[i],i
        elif vals[i] != '_' and u_count == 0:
            last_val, last_index = vals[i],i
        if i == len(vals)-1 and vals[i]=="_":
            # when the last value of the list is an _
            vals = mod_list(vals, last_val/(u_count+1), last_index,i)
            u_count = 0
    return vals

S=  ["_,_,_,24","40,_,_,_,60","80,_,_,_,_","_,_,30,_,_,_,50,_,_"]

for i in S:
    smoothed_values= curve_smoothing(i)
    print(smoothed_values)
```

```
[6.0, 6.0, 6.0, 6.0]
[20.0, 20.0, 20.0, 20.0, 20.0]
[16.0, 16.0, 16.0, 16.0, 16.0]
[10.0, 10.0, 12.0, 12.0, 12.0, 12.0, 4.0, 4.0, 4.0]
```

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)

Ex:

[8]:
```
def count(A, condition):
    """
    This function takes a list and a condition and returns the number of times␣
↪the condition is present in the list.
    """
    count = 0
    for i in A:
        if i==condition:
            count += 1
    return count
```

```
def compute_conditional_probabilites(A):
    """
    This function takes a nested list A, and prints the required probabilities.
    """
    # fs and ss are two tuples containing the possible values of F and S
    fs = ('F1','F2','F3','F4','F5')
    ss = ('S1','S2','S3')

    # s_c is a list that tracks the counts of S present in the list
    s_c = [0, 0, 0]
    # counting the number of times each S is present
    for p in A:
        if p[1] == 'S1':
            s_c[0] += 1
        elif p[1] == 'S2':
            s_c[1] += 1
        else:
            s_c[2] += 1

    # we iterate over the two tuples fs and ss, to get all possible combinations␣
 ↪and use count function to get the probability
    for f in fs:
        for index, s in enumerate(ss):
            print(f"P(F={f}|S=={s})={count(A, [f,s])}/{s_c[index]}, ", end = " ")
        print()

A =␣
 ↪[['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['F2','S1'],['F4','S

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=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,
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,
```

Q9: Given two sentances S1, S2

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

Ex:

[9]:
```
def string_features(S1, S2):
    """
    This function takes two strings S1 and S2 and performs the required set␣
 ↪operations and prints it's outputs
    """
```

```
    # splitting the string and storing them in a set to allow us to perform set␣
 ↪operations on them
    ss1 = set(S1.split())
    ss2 = set(S2.split())

    # performing set operations and printing their outputs
    print(len(ss1.intersection(ss2)))
    print(ss1.difference(ss2))
    print(ss2.difference(ss1))

S1= "the first column F will contain only 5 uniques values"
S2= "the second column S will contain only 3 uniques values"
string_features(S1, S2)
```

```
7
{'5', 'first', 'F'}
{'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
$\frac{-1}{8} \cdot ((1 \cdot log_{10}(0.4) + 0 \cdot log_{10}(0.6)) + (0 \cdot log_{10}(0.5) + 1 \cdot log_{10}(0.5)) + ... + (1 \cdot log_{10}(0.8) + 0 \cdot log_{10}(0.2)))$

[10]:
```
from math import log10

def compute_log_loss(A):
    """
    This function takes a nested list of numbers and prints the log loss of the␣
 ↪values.
    """
    # n stores the number of values in the list
    n = len(A)
    # sum keeps track of the sigma part of the equation and we use a for loop to␣
 ↪iterate over the values and implement the equation
    sum = 0
    for pair in A:
        sum += ((pair[0]*log10(pair[1]))+(1-pair[0])*log10(1-pair[1]))

    # multiplying the sum with -1/n to get the log loss and returing the log loss
    loss = (-1)*(1/n)*sum
    return loss
```

```
A = [[1, 0.4], [0, 0.5], [0, 0.9], [0, 0.3], [0, 0.6], [1, 0.1], [1, 0.9], [1, 0.
 ↪8]]
loss = compute_log_loss(A)
print(loss)
```

```
0.42430993457031635
```