Python_Mandatory_Assignment

June 14, 2020

Python: without numpy or sklearn

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

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

def matrix_mul(A,B):
    if len(A[0])!=len(B):
        return "Multiplication not possible"
    result=[]
    for i in range(len(A)):
        row=[]
        for j in range(len(B[0])):
            sum=0
            for k in range(len(A[0])):
                sum+=A[i][k]*B[k][j]
            row.append(sum)
        result.append(row)
    return result

matrix_mul(A,B)
```

[1]:
```
[[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.

[2]:
```
from random import uniform
# write your python code here
# you can take the above example as sample input for your program to test
# it should work for any general input try not to hard code for only given␣
 ↪input examples
# you can free to change all these codes/structure
```

```
A=[0,5,27,6,13,28,100,45,10,79]
def pick_a_number_from_list(A):
    # your code here for picking an element from with the probability␣
 ↪propotional to its magnitude
    s=sum(A)
    A_dash=[]
    for ele in A:
        A_dash.append(ele/s)

    A_tilde=[]
    csum=0
    for prob in A_dash:
        csum=csum+prob
        A_tilde.append(csum)

    r=uniform(0.0,1.0)
    for i in range(len(A)):
        if r<=A_tilde[i]:
            return A[i]#selected_random_number

def sampling_based_on_magnitued():
    for i in range(0,100):
        number = pick_a_number_from_list(A)
        print(number)

sampling_based_on_magnitued()
```

```
79
79
79
100
45
79
45
27
13
100
100
10
79
45
100
79
100
28
100
100
```

100
45
6
45
13
79
45
79
100
6
45
100
100
100
79
45
45
10
27
79
5
100
100
79
13
28
79
28
28
79
100
27
28
100
27
79
45
79
100
100
100
79
27
100
28
45
100
100

```
100
28
100
45
100
100
79
28
79
45
45
79
79
79
28
79
79
45
45
79
100
100
45
27
79
79
45
79
13
100
79
79
```

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]:
```
String='#2a$#b%c%561#'
def replace_digits(String):
    # write your code
    s=''
    for character in String:
        if character.isdigit():
            s+='#'
    if len(s)==0:
        return '(empty string)'
    return s # modified string which is after replacing the # with digits
```

```
replace_digits(String)
```

[3]: '####'

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]:
```
Students=['student1','student2','student3','student4','student5','student6','student7','studen
Marks=[45, 78, 12, 14, 48, 43, 47, 98, 35, 80]
n=len(Marks)
for i in range(n):
    for j in range(i+1,n):
        if Marks[i]>Marks[j]:
            temp=Marks[i]
            Marks[i]=Marks[j]
            Marks[j]=temp

            temp=Students[i]
            Students[i]=Students[j]
            Students[j]=temp

top_5_students=dict(zip(Students[::-1][0:5],Marks[::-1][0:5]))
least_5_students=dict(zip(Students[0:5],Marks[0:5]))
students_within_25_and_75=dict(zip(Students,Marks))

print("top 5 Students")
for key,value in top_5_students.items():
    print(key,' ',value)

print("\nleast 5 Students")
for key,value in least_5_students.items():
    print(key,' ',value)

print("\nstudents_within_25_and_75")
diff=max(students_within_25_and_75.values())-min(students_within_25_and_75.
 ↪values())
percent_25=diff*0.25
percent_75=diff*0.75
for key,value in students_within_25_and_75.items():
    if percent_25<value<percent_75:
        print(key,' ',value)
```

```
top 5 Students
student8   98
student10   80
student2   78
student5   48
student7   47

least 5 Students
student3   12
student4   14
student9   35
student6   43
student1   45

students_within_25_and_75
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):
    distance=[]
    for i in range(len(S)):
        d=math.acos(((S[i][0]*P[0])+(S[i][1]*P[1]))/((math.
 ↪sqrt((S[i][0]**2)+(S[i][1]**2)))*(math.sqrt((P[0]**2)+(P[1]**2)))))
        distance.append(d)
    dictionary=dict(zip(S,distance))
    sortvalues=sorted(dictionary.values())[0:5]
    closest_points_to_p=[]
    for i in sortvalues:
        for k,v in dictionary.items():
            if v==i:
                closest_points_to_p.append(k)
                break
    return closest_points_to_p  #  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)
for p in points:
    print(p) #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

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 i_am_the_one(red,blue,line):
    coeff=[]
    for x in re.split('x|y',line): #https://stackoverflow.com/questions/4998629/
 ↪split-string-with-multiple-delimiters-in-python
        coeff.append(float(x))
    a=coeff[0]
    b=coeff[1]
    c=coeff[2]

    R=[]
    B=[]
    for i in range(len(red)):
        R.append((a*red[i][0])+(b*red[i][1])+c)
    for j in range(len(blue)):
        B.append((a*blue[j][0])+(b*blue[j][1])+c)

    count_r=0
    for k in range(len(red)):
        if R[k]<0:
            count_r+=1

    count_b=0
    for l in range(len(blue)):
        if B[l]>0:
            count_b+=1
```

```
    if (count_r==len(red) and count_b==len(blue)) or (count_r==0 and␣
↪count_b==0):
        return 'YES'
    else:
        return 'NO'

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

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 curve_smoothing(string):
    s=string.split(',')
    count=0
    mid1_index=0
    for i in range(len(s)):
        if s[i]=='_':
            count+=1
        else:
            for k in range(i+1):
                s[k]=str((int(s[i])/(count+1)))
            mid1_index=i
            break

    mid1_value=float(s[mid1_index])
    mid2_index=0

    for j in range(mid1_index+1,len(s)):
        if s[j]!='_':
            mid2_index=j
```

```
            break

    if mid1_index!=mid2_index:
        mid_count=mid2_index-mid1_index+1
        mid2_value=float(s[mid2_index])
        for p in range(mid1_index,mid2_index+1):
            s[p]=str((mid1_value+mid2_value)/mid_count)

    if mid1_index!=len(s)-1:
        last_value=float(s[mid2_index])
        for q in range(mid2_index,len(s)):
            s[q]=str(last_value/(len(s)-mid2_index))

    return s #list of values
#S="_,_,_,24"
#S="40,_,_,_,60"
#S="80,_,_,_,_"
S=  "_,_,30,_,_,_,50,_,_"
#S="20,_,_,30, _, _,10,_,_,_,_,110"
smoothed_values= curve_smoothing(S)
print(smoothed_values)
```

```
['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]:
```
# write your python code here
# you can take the above example as sample input for your program to test
# it should work for any general input try not to hard code for only given␣
 ↪input strings



# you can free to change all these codes/structure
def compute_conditional_probabilites(A):
    n=len(A)
    cs1=0
    cs2=0
    cs3=0
    for i in range(n):
        if A[i][1]=='S1':
            cs1+=1
        if A[i][1]=='S2':
```

```
            cs2+=1
        if A[i][1]=='S3':
            cs3+=1

    for f in list(set([k[0] for k in A])):
        c1=0
        c2=0
        c3=0
        for i in range(n):
            if A[i][0]==f and A[i][1]=='S1':
                c1+=1
            if A[i][0]==f and A[i][1]=='S2':
                c2+=1
            if A[i][0]==f and A[i][1]=='S3':
                c3+=1
        print("P(F={}|S==S1)={}".format(f,c1/cs1))
        print("P(F={}|S==S2)={}".format(f,c2/cs2))
        print("P(F={}|S==S3)={}".format(f,c3/cs3))
A =␣
 ↪[['F1','S1'],['F2','S2'],['F3','S3'],['F1','S2'],['F2','S3'],['F3','S2'],['F2','S1'],['F4',

compute_conditional_probabilites(A)
```

```
P(F=F3|S==S1)=0.0
P(F=F3|S==S2)=0.3333333333333333
P(F=F3|S==S3)=0.3333333333333333
P(F=F5|S==S1)=0.25
P(F=F5|S==S2)=0.0
P(F=F5|S==S3)=0.0
P(F=F2|S==S1)=0.25
P(F=F2|S==S2)=0.3333333333333333
P(F=F2|S==S3)=0.3333333333333333
P(F=F1|S==S1)=0.25
P(F=F1|S==S2)=0.3333333333333333
P(F=F1|S==S3)=0.0
P(F=F4|S==S1)=0.25
P(F=F4|S==S2)=0.0
P(F=F4|S==S3)=0.3333333333333333
```

Q9: Given two sentances S1, S2

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

Ex:

[9]:
```
# write your python code here
# you can take the above example as sample input for your program to test
# it should work for any general input try not to hard code for only given␣
 ↪input strings
```

```
# you can free to change all these codes/structure
def string_features(S1, S2):
    a=0
    b=[]
    c=[]
    for ch1 in set(S1.split()):
        for ch2 in set(S2.split()):
            if ch1==ch2:
                a+=1

    for ch1 in S1.split():
        if ch1 not in S2.split():
            b.append(ch1)

    for ch2 in S2.split():
        if ch2 not in S1.split():
            c.append(ch2)

    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)
print(b)
print(c)
```

```
7
['first', 'F', '5']
['second', 'S', '3']
```

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]:
```
# write your python code here
# you can take the above example as sample input for your program to test
# it should work for any general input try not to hard code for only given␣
↪input strings
# you can free to change all these codes/structure
import math
```

```
def compute_log_loss(A):
    sum=0
    for i in range(len(A)):
        y=A[i][0]
        y_score=A[i][1]
        sum+=y*math.log10(y_score)+(1-y)*math.log10(1-y_score)
    loss=-sum/len(A)
    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
```

[ ]: