

```

1 #Data cleaning and formating
2 # This section Code is given by TA
3 import numpy as np
4 def clean_data(line):
5     return line.replace('(', '').replace(')', '').replace(' ', '').strip().split(',')
6
7 def fetch_data(filename):
8     with open(filename, 'r') as f:
9         input_data = f.readlines()
10    clean_input = list(map(clean_data, input_data))
11    f.close()
12    return clean_input
13
14
15 def readFile(dataset_path):
16     input_data = fetch_data(dataset_path)
17     input_np = np.array(input_data)
18     return input_np
19
20 training_data = './set1.txt'
21 test_data = './test1.txt'
22 large_120_data = './set2.txt'
23 input_test_data='./testInput.txt'
24 set1 = readFile(training_data)
25
26 test1 = readFile(test_data)
27
28 set2 = readFile(large_120_data)
29
30 test2=readFile(input_test_data)
31
32 #set1(14 data point array),set2(120 data point array), test1(given test data point array, test2(any input data point array))

```

if you want to see how data is formated print all 4 array varibale, set1,2, test1,2

```

1 #my code for data formatting and data cleaning from text file and convert into csv
2 set1 = open('set1.txt','r')
3 set1Data=set1.readlines()
4

1 set2 = open('set2.txt','r')
2 set2Data=set2.readlines()

1 setT = open('test1.txt','r')
2 setTestData=setT.readlines()

```

- x,y,a[x= height, y= weight, a=age] > for 14 data points
- x2,y2,a2=[same as above] > for 120 data points
- tx,ty,ta=[same as above] > for given test data point

```

1 x=[]
2 y=[]
3 a=[]
4 g=[]
5 x2=[]
6 y2=[]
7 a2=[]
8 g2=[]
9 tx=[]
10 ty=[]
11 ta=[]

1 FileIOprations = open("set1.csv","w")
2 FileIOprations.write("height,weight,age,gender\n")

```

25

```

1 FileIOprations2 = open("set2.csv","w")
2 FileIOprations2.write("height,weight,age,gender\n")

```

25

```
1 FileIOoprationsT = open("test1.csv", "w")
2 FileIOoprationsT.write("height,weight,age\n")
```

18

```
1 for index in set1Data:
2     FormatedList=index[3:].split(' ')
3     FormatedList=FormatedList[0:-1]
4     FinalString=(FormatedList[0][:-1])+' ,'+(FormatedList[1][:-1])+' ,'+(FormatedList[2][:-2])+' ,'+FormatedList[3]+'\n'
5     x.append(float(FormatedList[0][:-1]))
6     y.append(float(FormatedList[1][:-1]))
7     a.append(int(FormatedList[2][:-2]))
8     g.append((FormatedList[3]))
9     # print(FinalString)
10    FileIOoprations.write(FinalString)
11
```

```
1 for index in set2Data:
2     FormatedList=index[3:].split(' ')
3     FormatedList=FormatedList[0:-1]
4     FinalString=(FormatedList[0][:-1])+' ,'+(FormatedList[1][:-1])+' ,'+(FormatedList[2][:-2])+' ,'+FormatedList[3]+'\n'
5     x2.append(float(FormatedList[0][:-1]))
6     y2.append(float(FormatedList[1][:-1]))
7     a2.append(int(FormatedList[2][:-2]))
8     g2.append((FormatedList[3]))
9     FileIOoprations2.write(FinalString)
```

```
1 for i in range(len(setTestData)):
2     FormatedList=setTestData[i][2:].split(' ')
3     if i<len(setTestData)-1:
4         FormatedList=FormatedList[0:-1]
5     else:
6         FormatedList=FormatedList
7     FinalString=(FormatedList[0][:-1])+' ,'+(FormatedList[1][:-1])+' ,'+(FormatedList[2][:-1])+'\n'
8     tx.append(float(FormatedList[0][:-1]))
9     ty.append(float(FormatedList[1][:-1]))
10    ta.append(int(FormatedList[2][:-1]))
11    FileIOoprationsT.write(FinalString)
12
```

```
1 FileIOoprations.close()
```

```
1 FileIOoprations2.close()
```

```
1 FileIOoprationsT.close()
```

```
1 queue=[]
2 for i in range(len(tx)):
3     queue.append([tx[i],ty[i],ta[i]])
```

```
1 #Cartesian Distance calculation
2 import math
3 def dis(a,ta,b,tb,c,tc):
4     return math.sqrt((a-ta)**2+(b-tb)**2+(c-tc)**2)
```

```
1 #Cartesian distance calculation for 14 data point
2 import math
3 def CartesianDistanceForTest1():
4     ans=[]
5     ans1=[]
6     ans2=[]
7     ans3=[]
8     q0=queue[0]
9     q1=queue[1]
10    q2=queue[2]
11    q3=queue[3]
12    for i in range(len(x)):
13        ans.append(math.sqrt((x[i]-q0[0])**2+(y[i]-q0[1])**2+(a[i]-q0[2])**2))
14        ans1.append(math.sqrt((x[i]-q1[0])**2+(y[i]-q1[1])**2+(a[i]-q1[2])**2))
15        ans2.append(math.sqrt((x[i]-q2[0])**2+(y[i]-q2[1])**2+(a[i]-q2[2])**2))
16        ans3.append(math.sqrt((x[i]-q3[0])**2+(y[i]-q3[1])**2+(a[i]-q3[2])**2))
17    return ans,ans1,ans2,ans3
18 #uncomment below line to see Cartesian distance
19 # CartesianDistanceForTest1()
```

```
1 ##Cartesian distance calculation for 120 data point
```

```

2 def CartesianDistanceForTest2():
3     masterAns=[]
4     for i in range(len(test2)):
5         ans=[]
6         for j in range(len(x)):
7             ans.append(dis(x[j],float(test2[i][0]),y[j],float(test2[i][1]),a[j],int(test2[i][2])))
8         masterAns.append(ans)
9     return masterAns

1 # Cartesian prediction for 14 data point
2 def CartesianPredictionForTest1(inpu):
3     allans=CartesianDistanceForTest1()
4     ans=allans[0]
5     ans1=allans[1]
6     ans2=allans[2]
7     ans3=allans[3]
8     c=math.ceil(inpu/2)
9     l1=sorted(ans)[:inpu]
10    l2=sorted(ans1)[:inpu]
11    l3=sorted(ans2)[:inpu]
12    l4=sorted(ans3)[:inpu]
13    i1=[]
14    i2=[]
15    i3=[]
16    i4=[]
17    m1=0
18    m2=0
19    m3=0
20    m4=0
21    f1=0
22    f2=0
23    f3=0
24    f4=0
25    answer=[]
26    for i in range(0,inpu):
27        i1.append(ans.index(l1[i]))
28        i2.append(ans1.index(l2[i]))
29        i3.append(ans2.index(l3[i]))
30        i4.append(ans3.index(l4[i]))
31        if g[i1[i]] == 'M':
32            m1+=1
33        else:
34            f1+=1
35        if g[i2[i]] == 'M':
36            m2+=1
37        else:
38            f2+=1
39        if g[i3[i]] == 'M':
40            m3+=1
41        else:
42            f3+=1
43        if g[i4[i]] == 'M':
44            m4+=1
45        else:
46            f4+=1
47    if m1>=c:
48        answer.append('M')
49    else:
50        answer.append('W')
51    if m2>=c:
52        answer.append('M')
53    else:
54        answer.append('W')
55    if m3>=c:
56        answer.append('M')
57    else:
58        answer.append('W')
59    if m4>=c:
60        answer.append('M')
61    else:
62        answer.append('W')
63    return answer
64
```

```

1 # Cartesian prediction for 120 data point
2 def CartesianPredictionForTest2(inpu):
3     allans=[]
4     allans=CartesianDistanceForTest2()
5     # print(allans[0])
6     c=math.ceil(inpu/2)
7     sortedList=[]
```

```

8 for i in range(len(allans)):
9     sortedList.append(sorted(allans[i])[:inpu])
10 l1=sorted(allans[0])[:inpu]
11 # print(sortedList)
12 answer=[]
13 masterIndexAns=[]
14 for i in range(len(allans)):
15     ans=[]
16     m,f=0,0
17     for j in range(inpu):
18         ans.append(allans[i].index(sortedList[i][j]))
19         if g[ans[j]] == 'M':
20             m+=1
21         else:
22             f+=1
23     if m>=c:
24         answer.append('M')
25     else:
26         answer.append('W')
27     # print(answer,m,f)
28     masterIndexAns.append(ans)
29 return answer
30 # print(masterIndexAns)

1 #manhattan distance calculation
2 def ManDis(a,ta,b,tb,c,tc):
3     return (abs(a-ta)+abs(b-tb)+abs(c-tc))

1 #Manhattan Dsiatance for 14 data point
2 import math
3 def ManhattanDistanceForTest1():
4     ans=[]
5     ans1=[]
6     ans2=[]
7     ans3=[]
8     p = ([x[0],y[0],a[0]])
9     q=[1.62065758929,59.376557437583,32]
10    q1=[1.7793983848363, 72.071775670801, 36]
11    q2=[1.7004576585974, 66.267508112786, 31]
12    q3=[1.6591086215159, 61.751621901787, 29]
13    for i in range(len(x)):
14        t=[]
15        t1=[]
16        t2=[]
17        t3=[]
18        t = (abs(x[i]-q[0])+abs(y[i]-q[1])+abs(a[i]-q[2]))
19        ans.append(t)
20        t1=(abs(x[i]-q1[0])+abs(y[i]-q1[1])+abs(a[i]-q1[2]))
21        ans1.append(t1)
22        t2=(abs(x[i]-q2[0])+abs(y[i]-q2[1])+abs(a[i]-q2[2]))
23        ans2.append(t2)
24        t3=(abs(x[i]-q3[0])+abs(y[i]-q3[1])+abs(a[i]-q3[2]))
25        ans3.append(t3)
26    return ans,ans1,ans2,ans3

1 #Manhattan Dsiatance for 120 data point
2 def ManhattanDistanceForTest2():
3     masterAns=[]
4     for i in range(len(test2)):
5         ans=[]
6         for j in range(len(x)):
7             ans.append(ManDis(x[j],float(test2[i][0]),y[j],float(test2[i][1]),a[j],int(test2[i][2])))
8         masterAns.append(ans)
9     return masterAns

1 # manhattan prediction for 14 data point
2 def ManhattanPredictionForTest1(inpu):
3     allans=ManhattanDistanceForTest1()
4     ans=allans[0]
5     ans1=allans[1]
6     ans2=allans[2]
7     ans3=allans[3]
8     c=math.ceil(inpu/2)
9     l1=sorted(ans)[:inpu]
10    l2=sorted(ans1)[:inpu]
11    l3=sorted(ans2)[:inpu]
12    l4=sorted(ans3)[:inpu]
13    i1=[]
14    i2=[]
15    i3=[]
16    i4=[]

```

```

17 m1=0
18 m2=0
19 m3=0
20 m4=0
21 f1=0
22 f2=0
23 f3=0
24 f4=0
25 answer=[]
26 for i in range(0,inpu):
27     i1.append(ans.index(l1[i]))
28     i2.append(ans1.index(l2[i]))
29     i3.append(ans2.index(l3[i]))
30     i4.append(ans3.index(l4[i]))
31     if g[i1[i]] == 'M':
32         m1+=1
33     else:
34         f1+=1
35     if g[i2[i]] == 'M':
36         m2+=1
37     else:
38         f2+=1
39     if g[i3[i]] == 'M':
40         m3+=1
41     else:
42         f3+=1
43     if g[i4[i]] == 'M':
44         m4+=1
45     else:
46         f4+=1
47 if m1>=c:
48     answer.append('M')
49 else:
50     answer.append('W')
51 if m2>=c:
52     answer.append('M')
53 else:
54     answer.append('W')
55 if m3>=c:
56     answer.append('M')
57 else:
58     answer.append('W')
59 if m4>=c:
60     answer.append('M')
61 else:
62     answer.append('W')
63 return answer

```

```

1 # manhattan prediction for 120 data point
2 def ManhattanPredictionForTest2(inpu):
3     allans=[]
4     allans=ManhattanDistanceForTest2()
5     c=math.ceil(inpu/2)
6     sortedList=[]
7     for i in range(len(allans)):
8         sortedList.append(sorted(allans[i])[:inpu])
9     l1=sorted(allans[0])[:inpu]
10    answer=[]
11    masterIndexAns=[]
12    for i in range(len(allans)):
13        ans=[]
14        m,f=0,0
15        for j in range(inpu):
16            ans.append(allans[i].index(sortedList[i][j]))
17            if g[ans[j]] == 'M':
18                m+=1
19            else:
20                f+=1
21        if m>=c:
22            answer.append('M')
23        else:
24            answer.append('W')
25        masterIndexAns.append(ans)
26    return answer

```

```

1 def MinDis(a,ta,b,tb,c,tc):
2     return ((abs(pow(a-ta,3))+abs(pow(b-tb,3))+abs(pow(c-tc,3)))**(1/3))

```

```

1 #Minkowski Distance
2 import math
3 def MinkowskiDistanceForTest1():

```

```

4 ans=[]
5 ans1=[]
6 ans2=[]
7 ans3=[]
8
9 p = ([x[0],y[0],a[0]])
10 q0=[1.62065758929,59.376557437583,32]
11 q1=[1.7793983848363, 72.071775670801, 36]
12 q2=[1.7004576585974, 66.267508112786, 31]
13 q3=[1.6591086215159, 61.751621901787, 29]
14 for i in range(len(x)):
15     ans.append((abs(pow(x[i]-q0[0],3))+abs(pow(y[i]-q0[1],3))+abs(pow(a[i]-q0[2],3)))**(1/3))
16     ans1.append((abs(pow(x[i]-q1[0],3))+abs(pow(y[i]-q1[1],3))+abs(pow(a[i]-q1[2],3)))**(1/3))
17     ans2.append((abs(pow(x[i]-q2[0],3))+abs(pow(y[i]-q2[1],3))+abs(pow(a[i]-q2[2],3)))**(1/3))
18     ans3.append((abs(pow(x[i]-q3[0],3))+abs(pow(y[i]-q3[1],3))+abs(pow(a[i]-q3[2],3)))**(1/3))
19 return ans,ans1,ans2,ans3

1 def MinkowskiDistanceForTest2():
2     masterAns=[]
3     for i in range(len(test2)):
4         ans=[]
5         for j in range(len(x)):
6             ans.append(MinDis(x[j],float(test2[i][0]),y[j],float(test2[i][1]),a[j],int(test2[i][2])))
7         masterAns.append(ans)
8     return masterAns

1 def MinkowskiPredictionForTest1(inpu):
2     allans=MinkowskiDistanceForTest1()
3     ans=allans[0]
4     ans1=allans[1]
5     ans2=allans[2]
6     ans3=allans[3]
7     c=math.ceil(inpu/2)
8     l1=sorted(ans)[:inpu]
9     l2=sorted(ans1)[:inpu]
10    l3=sorted(ans2)[:inpu]
11    l4=sorted(ans3)[:inpu]
12    i1=[]
13    i2=[]
14    i3=[]
15    i4=[]
16    m1=0
17    m2=0
18    m3=0
19    m4=0
20    f1=0
21    f2=0
22    f3=0
23    f4=0
24    answer=[]
25    for i in range(0,inpu):
26        i1.append(ans.index(l1[i]))
27        i2.append(ans1.index(l2[i]))
28        i3.append(ans2.index(l3[i]))
29        i4.append(ans3.index(l4[i]))
30        if g[i1[i]] == 'M':
31            m1+=1
32        else:
33            f1+=1
34        if g[i2[i]] == 'M':
35            m2+=1
36        else:
37            f2+=1
38        if g[i3[i]] == 'M':
39            m3+=1
40        else:
41            f3+=1
42        if g[i4[i]] == 'M':
43            m4+=1
44        else:
45            f4+=1
46    if m1>=c:
47        answer.append('M')
48    else:
49        answer.append('W')
50    if m2>=c:
51        answer.append('M')
52    else:
53        answer.append('W')
54    if m3>=c:
55        answer.append('M')

```

```

56     else:
57         answer.append('W')
58     if m4>=c:
59         answer.append('M')
60     else:
61         answer.append('W')
62     return answer

1 def MinkowskiPredictionForTest2(inpu):
2     allans=[]
3     allans=MinkowskiDistanceForTest2()
4     # print(allans[0])
5     c=math.ceil(inpu/2)
6     sortedList=[]
7     for i in range(len(allans)):
8         sortedList.append(sorted(allans[i])[:inpu])
9     l1=sorted(allans[0])[:inpu]
10    # print(sortedList)
11    answer=[]
12    masterIndexAns=[]
13    for i in range(len(allans)):
14        ans=[]
15        m,f=0,0
16        for j in range(inpu):
17            ans.append(allans[i].index(sortedList[i][j]))
18            if g[ans[j]] == 'M':
19                m+=1
20            else:
21                f+=1
22        if m>=c:
23            answer.append('M')
24        else:
25            answer.append('W')
26        # print(answer,m,f)
27        masterIndexAns.append(ans)
28    return answer
29    # print(masterIndexAns)

```

** Question: 1 C**

```

1 # data formatting for as per leave-out condition
2 masterans=[]
3 masterGen=[]
4 masterTrainingDataX=[]
5 masterTrainingDataY=[]
6 masterTrainingDataA=[]
7 for i in range(len(x2)):
8     testx=0
9     testy=0
10    testa=0
11    trainx=[]
12    trainy=[]
13    traina=[]
14    traing=[]
15    ans=[]
16    for j in range(len(x2)):
17        if i==j:
18            testx=x2[j]
19            testy=y2[j]
20            testa=a2[j]
21        else:
22            trainx.append(x2[j])
23            trainy.append(y2[j])
24            traina.append(a2[j])
25            traing.append(g2[j])
26    for k in range(len(x2)-1):
27        temp=dis(trainx[k],testx,trainy[k],testy,traina[k],testa)
28        ans.append(temp)
29    masterTrainingDataX.append(trainx)
30    masterTrainingDataY.append(trainy)
31    masterTrainingDataA.append(traina)
32    masterans.append(ans)
33    masterGen.append(traing)

1 import math
2 def LeaveOutPrediction(inpu):
3     sucess=0
4     for i in range(0,len(masterGen)):
5         tempArr=masterans[i]
6         tempgen=masterGen[i]

```

```

7 # print(len(tempArr),len(tempgen))
8 for j in range(len(masterGen)-1):
9     ans=tempArr
10    c=math.ceil(inpu/2)
11    l1=sorted(ans)[:inpu]
12    i1=[]
13    m=0
14    f=0
15    answerr=[]
16    for k in range(inpu):
17        i1.append(ans.index(l1[k]))
18        if tempgen[i1[k]] == 'M':
19            m+=1
20        else:
21            f+=1
22    if m>=c and g2[i] == 'M':
23        sucess+=1
24        answerr.append('M')
25    elif f>=c and g2[i] == 'W':
26        sucess+=1
27        answerr.append('W')
28    # print(m,f,sucess,g2[i])
29    print("Accuracy for k =",inpu,"is", (sucess/len(x2))*100,"%")
30    print("Erroneous for k =",inpu,"is", (1-sucess/len(x2))*100,"%")
31
32

```

We can say that we got best accuracy with 0.63 or 63% for k=9 from output of LeaveOutPrediction(int) function.

Now we are removing age data from prediction algorithm and see what will be accuracy

```

1 import math
2 def dist(a,ta,b,tb):
3     return math.sqrt((a-ta)**2+(b-tb)**2)

1 #age removing and data fornmatting
2 masterAns=[]
3 masterGender=[]
4 for i in range(len(x2)):
5     testx=0
6     testy=0
7     trainx=[]
8     trainy=[]
9     traing=[]
10    ans=[]
11    for j in range(len(x2)):
12        if i==j:
13            testx=x2[j]
14            testy=y2[j]
15        else:
16            trainx.append(x2[j])
17            trainy.append(y2[j])
18            traing.append(g2[j])
19    for k in range(len(x2)-1):
20        temp=dist(trainx[k],testx,trainy[k],testy)
21        ans.append(temp)
22    masterAns.append(ans)
23    masterGender.append(traing)
24

```

```

1 import math
2 def LeaveOutPredictionFor1D(inpu):
3     sucess=0
4     for i in range(0,len(masterGen)):
5         tempArr=masterAns[i]
6         tempgen=masterGender[i]
7         # print(len(tempArr),len(tempgen))
8         for j in range(len(masterGen)-1):
9             ans=tempArr
10            c=math.ceil(inpu/2)
11            l1=sorted(ans)[:inpu]
12            i1=[]
13            m=0
14            f=0

```

```

15     anserr=[]
16     for k in range(inpu):
17         i1.append(ans.index(l1[k]))
18         if tempgen[i1[k]] == 'M':
19             m+=1
20         else:
21             f+=1
22     if m>=c and g2[i] == 'M':
23         sucess+=1
24         anserr.append('M')
25     elif f>=c and g2[i] == 'W':
26         sucess+=1
27         anserr.append('W')
28     # print(m,f,sucess,g2[i])
29     print("Accuracy when age data is removed for k =",inpu,"is", (sucess/len(x2))*100,"%")
30     print("Erroneous when age data is removed for k =",inpu,"is", (1-sucess/len(x2))*100,"%")

```

Finished Question 1 Here..... Hurrayyyyyy

```

1 #mean calculation
2 import math
3 def mean(arr):
4     su=sum(arr)
5     ave=su/len(arr)
6     return ave

```

```

1 # standard deviation calculation
2 import math
3 def sd(arr):
4     me = mean(arr)
5     var = sum((i - me)**2 for i in arr) / (len(arr)-1)
6     sdev = pow(var,0.5)
7     return sdev
8

```

```

1 #for M height
2 import math
3 def hForM():
4     arrHM=[]
5     for i in range(len(x)):
6         if g[i] == 'M':
7             arrHM.append(x[i])
8     return arrHM

```

```

1 #for M weight
2 import math
3 def wForM():
4     arrHM=[]
5     for i in range(len(x)):
6         if g[i] == 'M':
7             arrHM.append(y[i])
8     return arrHM

```

```

1 #for M age
2 import math
3 def aForM():
4     arrHM=[]
5     for i in range(len(x)):
6         if g[i] == 'M':
7             arrHM.append(a[i])
8     return arrHM

```

```

1 #for W height
2 import math
3 def hForW():
4     arrHW=[]
5     for i in range(len(x)):
6         if g[i] == 'W':
7             arrHW.append(x[i])
8     return arrHW

```

```

1 #for W weight
2 import math
3 def wForW():
4     arrHW=[]
5     for i in range(len(x)):

```

```

6   if g[i] == 'W':
7     arrHW.append(y[i])
8   return arrHW

```

```

1 #for W age
2 import math
3 def aForW():
4   arrHW=[]
5   for i in range(len(x)):
6     if g[i] == 'W':
7       arrHW.append(a[i])
8   return arrHW

```

```

1 meanHM=mean(hForM())
2 meanWM=mean(wForM())
3 meanAM=mean(aForM())
4 meanHW=mean(hForW())
5 meanWW=mean(wForW())
6 meanAW=mean(aForW())
7 sdHM=sd(hForM())
8 sdWM=sd(wForM())
9 sdAM=sd(aForM())
10 sdHW=sd(hForW())
11 sdWW=sd(wForW())
12 sdAW=sd(aForW())
13 # all nessary variable for gussian naive baysen calculations

```

$$\mu = \frac{1}{n} \sum_{i=1}^n x_i$$

Mean

$$\sigma = \left[\frac{1}{n-1} \sum_{i=1}^n (x_i - \mu)^2 \right]^{0.5}$$

Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Normal distribution

```

1 # normal distrubution calculation for gussian naive baysen for Women
2 def naiveBayseW(h,w,a):
3   ans=0
4   con=1/math.sqrt(2*math.pi*(sdAW**2))
5   e=-((a - meanAW)**2)/(2*(sdAW**2))
6   con2=math.exp(e)
7   aw=con*con2
8
9   con=1/math.sqrt(2*math.pi*(sdHW**2))
10  e=-((h - meanHW)**2)/(2*(sdHW**2))
11  con2=math.exp(e)
12  hw=con*con2
13
14  con=1/math.sqrt(2*math.pi*(sdWW**2))
15  e=-((w - meanWW)**2)/(2*(sdWW**2))
16  con2=math.exp(e)
17  ww=con*con2
18  ans=ww*aw*hw
19  return ans
20

```

```

1 # normal distrubution calculation for gussian naive baysen for Men
2 def naiveBayseM(h,w,a):
3   ans=0
4   con=1/math.sqrt(2*math.pi*(sdAM**2))
5   e=-((a - meanAM)**2)/(2*(sdAM**2))
6   con2=math.exp(e)
7   am=con*con2
8   con=1/math.sqrt(2*math.pi*(sdHM**2))
9   e=-((h - meanHM)**2)/(2*(sdHM**2))
10  con2=math.exp(e)
11  hm=con*con2
12
13  con2=math.exp(-((w - meanWM)**2)/(2*(sdWM**2)))/math.sqrt(2*math.pi*(sdWM**2))
14  ansm=am*hm*con2
15  return ansm

```

```

1 #prediction based on prediction answer
2 def naiveBaysespredictionForTest1():
3   ansArr=[]

```

```

4 for i in range(len(tx)):
5     if naiveBayseM(queue[i][0],queue[i][1],queue[i][2]) > naiveBayseW(queue[i][0],queue[i][1],queue[i][2]):
6         ansArr.append("M")
7     else:
8         ansArr.append("W")
9     # print(naiveBayseM(queue[i][0],queue[i][1],queue[i][2]),naiveBayseW(queue[i][0],queue[i][1],queue[i][2]))
10    print("Naive bayses prediction answer for 2 A : ",ansArr)

```

```

1 ##prediction based on prediction answer for diffrent test inputs from testInput.txt file
2 def naiveBaysesPredictionForTest2():
3     ansArr=[]
4     for i in range(len(test2)):
5         if naiveBayseM(float(test2[i][0]),float(test2[i][1]),int(test2[i][2])) > naiveBayseW(float(test2[i][0]),float(test2[i][1]),int(test2[i][2])):
6             ansArr.append("M")
7         else:
8             ansArr.append("W")
9     print("Naive bayses prediction answer from testInput.txt fil 2 B : ",ansArr)

```

▼ Data formatting and mean , sd calucation for leave out

```

1 #for M Height for C
2 import math
3 def hmForTest2():
4     arr=[]
5     for i in range(len(x2)):
6         ans=[]
7         for j in range(len(masterTrainingDataX[0])):
8             if masterGender[i][j] == 'M':
9                 ans.append(masterTrainingDataX[i][j])
10        arr.append(ans)
11    # print(ans)
12    return arr

```

```

1 #for M weight for c
2 import math
3 def wmForTest2():
4     arr=[]
5     for i in range(len(x2)):
6         ans=[]
7         for j in range(len(masterTrainingDataX[0])):
8             if masterGender[i][j] == 'M':
9                 ans.append(masterTrainingDataY[i][j])
10        arr.append(ans)
11    # print(ans)
12    return arr

```

```

1 #for M weight for C
2 import math
3 def amForTest2():
4     arr=[]
5     for i in range(len(x2)):
6         ans=[]
7         for j in range(len(masterTrainingDataX[0])):
8             if masterGender[i][j] == 'M':
9                 ans.append(masterTrainingDataA[i][j])
10        arr.append(ans)
11    # print(ans)
12    return arr

```

```

1 #for W height for C
2 import math
3 def hwForTest2():
4     arr=[]
5     for i in range(len(x2)):
6         ans=[]
7         for j in range(len(masterTrainingDataX[0])):
8             if masterGender[i][j] == 'W':
9                 ans.append(masterTrainingDataX[i][j])
10        arr.append(ans)
11    # print(ans)
12    return arr

```

```

1 #for W weight for C
2 import math
3 def wwForTest2():
4     arr=[]
5     for i in range(len(x2)):

```

```

6     ans=[]
7     for j in range(len(masterTrainingDataX[0])):
8         if masterGender[i][j] == 'W':
9             ans.append(masterTrainingDataY[i][j])
10    arr.append(ans)
11 # print(ans)
12 return arr
13

1 #for W age for C
2 import math
3 def awForTest2():
4     arr=[]
5     for i in range(len(x2)):
6         ans=[]
7         for j in range(len(masterTrainingDataX[0])):
8             if masterGender[i][j] == 'W':
9                 ans.append(masterTrainingDataA[i][j])
10    arr.append(ans)
11 # print(ans)
12 return arr
13

1 #MEAN CALCULATION FOR 2 C
2 import math
3 def meanTest2(arr):
4     ans=[]
5     k=0
6     for i in range(len(arr)):
7         su=sum(arr[i])
8         ave = su/len(arr[i])
9         k+=1
10    ans.append(ave)
11 return ans

1 #SD CALCULATION FOR 2 C
2 import math
3 def sdTest2(arr):
4     ans=[]
5     k=0
6     for i in range(len(arr)):
7         me = mean(arr[i])
8         var = sum((i - me)**2 for i in arr[i]) / (len(arr[i])-1)
9         sdev = pow(var,0.5)
10    k+=1
11    ans.append(sdev)
12 return ans

1 meanHMTTest2=[]
2 meanAMTest2=[]
3 meanWMTest2=[]
4 meanHWTest2=[]
5 meanWWTest2=[]
6 meanAWTest2=[]
7 sdHMTTest2=[]
8 sdWMTest2=[]
9 sdAMTest2=[]
10 sdHWTest2=[]
11 sdWWTest2=[]
12 sdAWTest2=[]
13
14 meanHMTTest2=meanTest2(hmForTest2())
15 meanWMTest2=meanTest2(wmForTest2())
16 meanAMTest2=meanTest2(amForTest2())
17 meanHWTest2=meanTest2(hwForTest2())
18 meanWWTest2=meanTest2(wwForTest2())
19 meanAWTest2=meanTest2(awForTest2())
20 sdHMTTest2=sdTest2(hmForTest2())
21 sdWMTest2=sdTest2(wmForTest2())
22 sdAMTest2=sdTest2(amForTest2())
23 sdHWTest2=sdTest2(hwForTest2())
24 sdWWTest2=sdTest2(wwForTest2())
25 sdAWTest2=sdTest2(awForTest2())
26
27 #ALL NECESSARY VARIABLES FOR CALCULATION(EG. meanHMTTest2= mean for height label for men for 2C )

```

1 #Gaussian naive bayes calculation for 2C for Women
2 def naiveBayseWTest2():
3 meanAW=meanAWTest2

```

4 meanHW=meanHWTest2
5 meanWW=meanWWTest2
6 sdAW=sdAWTest2
7 sdHW=sdHWTest2
8 sdWW=sdWWTest2
9 ans=[]
10 for i in range(len(sdAWTest2)):
11     con=1/math.sqrt(2*math.pi*(float(sdAW[i])**2))
12     e=-((a2[i] - meanAW[i])**2)/(2*(sdAW[i]**2)))
13     con2=math.exp(e)
14     aw=con*con2
15
16     con=1/math.sqrt(2*math.pi*(sdHW[i]**2))
17     e=-((x2[i] - meanHW[i])**2)/(2*(sdHW[i]**2)))
18     con2=math.exp(e)
19     hw=con*con2
20
21     con=1/math.sqrt(2*math.pi*(sdWW[i]**2))
22     e=-((y2[i]- meanWW[i])**2)/(2*(sdWW[i]**2)))
23     con2=math.exp(e)
24     ww=con*con2
25     ans.append(ww*aw*hw)
26 return ans

```

```

1 #Gussian naive baysen calculation for 2C for Men
2 def naiveBayseMTest2():
3     meanAM=meanAMTest2
4     meanHM=meanHMTTest2
5     meanWM=meanWMTTest2
6     sdAM=sdAMTest2
7     sdHM=sdHMTTest2
8     sdWM=sdWMTTest2
9     ans=[]
10    for i in range(len(sdAMTest2)):
11        # print(i)
12        con=1/math.sqrt(2*math.pi*(float(sdAM[i])**2))
13        e=-((a2[i] - meanAM[i])**2)/(2*(sdAM[i]**2)))
14        con2=math.exp(e)
15        aw=con*con2
16
17        con=1/math.sqrt(2*math.pi*(sdHM[i]**2))
18        e=-((x2[i] - meanHM[i])**2)/(2*(sdHM[i]**2)))
19        con2=math.exp(e)
20        hw=con*con2
21
22        con=1/math.sqrt(2*math.pi*(sdWM[i]**2))
23        e=-((y2[i]- meanWM[i])**2)/(2*(sdWM[i]**2)))
24        con2=math.exp(e)
25        ww=con*con2
26        ans.append(ww*aw*hw)
27    return ans
28

```

```

1 #prediction ans for leave out array for 2 C
2 def naiveBaysesPredictionFor1C():
3     ansArr=[]
4     for i in range(len(x2)):
5         if naiveBayseMTest2()[i] > naiveBayseWTest2()[i]:
6             ansArr.append("M")
7         else:
8             ansArr.append("W")
9     # print("Naive bayses prediction answer for 1 A : ",ansArr)
10    return ansArr

```

Without age

```

1 #Gussian naive baysen calculation for 2C for Women
2 def naiveBayseWTest2():
3     meanAW=meanAWTest2
4     meanHW=meanHWTest2
5     meanWW=meanWWTest2
6     sdAW=sdAWTest2
7     sdHW=sdHWTest2
8     sdWW=sdWWTest2
9     ans=[]
10    for i in range(len(sdAWTest2)):
11
12        con=1/math.sqrt(2*math.pi*(sdHW[i]**2))
13        e=-((x2[i] - meanHW[i])**2)/(2*(sdHW[i]**2)))
14        con2=math.exp(e)

```

```

15     hw=con*con2
16
17     con=1/math.sqrt(2*math.pi*(sdWW[i]**2))
18     e=-((y2[i]- meanWW[i])**2)/(2*(sdWW[i]**2)))
19     con2=math.exp(e)
20     ww=con*con2
21     ans.append(ww*hw)
22 return ans

```

```

1 #Gussian naive baysen calculation for 2C for Men
2 def naiveBayseMTest2():
3     meanAM=meanAMTest2
4     meanHM=meanHMTTest2
5     meanWM=meanWMTTest2
6     sdAM=sdAMTest2
7     sdHM=sdHMTTest2
8     sdWM=sdWMTTest2
9     ans=[]
10    for i in range(len(sdAMTest2)):
11
12        con=1/math.sqrt(2*math.pi*(sdHM[i]**2))
13        e=-((x2[i] - meanHM[i])**2)/(2*(sdHM[i]**2)))
14        con2=math.exp(e)
15        hw=con*con2
16
17        con=1/math.sqrt(2*math.pi*(sdWM[i]**2))
18        e=-((y2[i]- meanWM[i])**2)/(2*(sdWM[i]**2)))
19        con2=math.exp(e)
20        ww=con*con2
21        ans.append(ww*hw)
22    return ans
23

```

```

1 #prediction ans for leave out array for 2 C
2 def naiveBaysesPredictionFor1C():
3     ansArr=[]
4     for i in range(len(x2)):
5         if naiveBayseMTest2()[i] > naiveBayseWTest2()[i]:
6             ansArr.append("M")
7         else:
8             ansArr.append("W")
9     # print("Naive bayses prediction answer for 1 A : ",ansArr)
10    return ansArr

```

```

1 print("For K=1, metric = euclidean --> ",CartesianPredictionForTest1(1))
2 print("For K=1, metric = Manhattan --> ",ManhattanPredictionForTest1(1))
3 print("For K=1, metric = Minkowski --> ",MinkowskiPredictionForTest1(1))

For K=1, metric = euclidean --> ['W', 'W', 'W', 'W']
For K=1, metric = Manhattan --> ['W', 'W', 'W', 'W']
For K=1, metric = Minkowski --> ['W', 'W', 'W', 'W']

```

```

1 print("For K=3, metric = euclidean --> ",CartesianPredictionForTest1(3))
2 print("For K=3, metric = Manhattan --> ",ManhattanPredictionForTest1(3))
3 print("For K=3, metric = Minkowski --> ",MinkowskiPredictionForTest1(3))

For K=3, metric = euclidean --> ['W', 'M', 'W', 'W']
For K=3, metric = Manhattan --> ['W', 'M', 'W', 'W']
For K=3, metric = Minkowski --> ['W', 'M', 'W', 'W']

```

```

1 print("For K=7, metric = euclidean --> ",CartesianPredictionForTest1(7))
2 print("For K=7, metric = Manhattan --> ",ManhattanPredictionForTest1(7))
3 print("For K=7, metric = Minkowski --> ",MinkowskiPredictionForTest1(7))

For K=7, metric = euclidean --> ['W', 'M', 'W', 'W']
For K=7, metric = Manhattan --> ['W', 'M', 'W', 'W']
For K=7, metric = Minkowski --> ['W', 'M', 'W', 'W']

```

#enter K value below

to change input data, open file testInput.txt and change data and then save it and run code

to change input training data change dataset into set1.txt

```
1 #enter K value below
2 #to change input data, open file testInput.txt and change data and then save it and run code
3 #to change input training data change dataset into set1.txt
4 inp=int(input("Enter any K value to all function's result with that K value: "))
5
6 print("For K=",inp," metric = euclidean --> ",CartesianPredictionForTest1(inp))
7 print("For K=",inp," metric = Manhattan --> ",ManhattanPredictionForTest1(inp))
8 print("For K=",inp," metric = Minkowski --> ",MinkowskiPredictionForTest1(inp))
9
```

```
Enter any K value to all function's result with that K value: 7
For K= 7 metric = euclidean --> ['W', 'M', 'W', 'W']
For K= 7 metric = Manhattan --> ['W', 'M', 'W', 'W']
For K= 7 metric = Minkowski --> ['W', 'M', 'W', 'W']
```

▼ answer 1 C

```
1 LeaveOutPrediction(1)
```

```
Accuracy for k = 1 is 55.00000000000001 %
Erroneous for k = 1 is 44.99999999999999 %
```

```
1 LeaveOutPrediction(3)
```

```
Accuracy for k = 3 is 61.66666666666667 %
Erroneous for k = 3 is 38.33333333333333 %
```

```
1 LeaveOutPrediction(7)
```

```
Accuracy for k = 7 is 60.83333333333333 %
Erroneous for k = 7 is 39.16666666666667 %
```

▼ best answer for k=9

```
1 LeaveOutPrediction(9)
```

```
Accuracy for k = 9 is 63.33333333333333 %
Erroneous for k = 9 is 36.66666666666667 %
```

```
1 LeaveOutPrediction(11)
```

```
Accuracy for k = 11 is 59.16666666666664 %
Erroneous for k = 11 is 40.83333333333336 %
```

▼ answer 1 D

Accuracy is increased with lesser data in knn algorithm. and for k=3 I got best answer when age data is missing which is 7% more accuracy than age is present for previous question.

```
1 LeaveOutPredictionFor1D(1)
```

```
Accuracy when age data is removed for k = 1 is 62.5 %
Erroneous when age data is removed for k = 1 is 37.5 %
```

```
1 LeaveOutPredictionFor1D(3)
```

```
Accuracy when age data is removed for k = 3 is 70.8333333333334 %
Erroneous when age data is removed for k = 3 is 29.16666666666664 %
```

```
1 LeaveOutPredictionFor1D(7)
```

```
Accuracy when age data is removed for k = 7 is 63.3333333333333 %
Erroneous when age data is removed for k = 7 is 36.6666666666667 %
```

```
1 LeaveOutPredictionFor1D(9)
```

```
Accuracy when age data is removed for k = 9 is 60.0 %
Erroneous when age data is removed for k = 9 is 40.0 %
```

```
1 LeaveOutPredictionFor1D(11)
```

```
Accuracy when age data is removed for k = 11 is 57.49999999999999 %
```

Erroneous when age data is removed for k = 11 is 42.500000000000001 %

```
1 naiveBaysesPredictionForTest1()
Naive bayses prediction answer for 2 A : ['W', 'W', 'W', 'W']

1 naiveBaysesPredictionForTest2()
Naive bayses prediction answer from testInput.txt fil 2 B : ['W', 'W', 'W', 'W']

1 #calculating accuracy for 2c
2 def CheckNaiveBaysesPredictionAnsWithOriginalAns():
3     l=naiveBaysesPredictionFor1C()
4     res=0
5     for i in range(120):
6         if l[i]==g2[i]:
7             res+=1
8     return res*100/120
9 print("accuracy for 2 C ",CheckNaiveBaysesPredictionAnsWithOriginalAns(),"%")
10 print("Error for 2 C without age ",100-CheckNaiveBaysesPredictionAnsWithOriginalAns(),"%")

accuracy for 2 C 69.16666666666667 %
Error for 2 C without age 30.83333333333333 %

1 #calculating accuracy for 2c
2 def CheckNaiveBaysesPredictionAnsWithOriginalAns():
3     l=naiveBaysesPredictionFor1C()
4     res=0
5     for i in range(120):
6         if l[i]==g2[i]:
7             res+=1
8     return res*100/120
9 print("accuracy for 2 D without age ",CheckNaiveBaysesPredictionAnsWithOriginalAns(),"%")
10 print("Error for 2 D without age ",100-CheckNaiveBaysesPredictionAnsWithOriginalAns(),%)

accuracy for 2 D without age 69.16666666666667 %
Error for 2 D without age 30.83333333333333 %
```

2 E) Compare the results of the two classifiers (i.e., the results form 1 c) and 1d) with the ones from 2 c) 2d) and discuss reasons why one might perform better than the other.

answer: we got accuracy for 1C and 1D : 63.83 and 70.83 respectively.

and we got accuracy for 2C and 2D : 70.83 and 69.83 respectively.

so with lesser data in KNN we got 7% increase in accuracy however for Gaussian Naïve Bayes Classification we got almost same result with 1% decline. So I can say that with more data Gaussian Naïve Bayes Classification perform better than KNN and it has lesser difference with less data as well.

