Name: Jaykumar Dhirubhai Patel UTA id: 1001945949 Section: CSE 6363-005

```
#Data cleaning and formating
# This section Code is given by TA
import numpy as np
def clean_data(line):
    return line.replace('(', '').replace(')', '').replace(' ', '').strip().split(',')

def fetch_data(filename):
    with open(filename, 'r') as f:
        input_data = f.readlines()
        clean_input = list(map(clean_data, input_data))
        f.close()
    return clean_input


def readFile(dataset_path):
    input_data = fetch_data(dataset_path)
    input_np = np.array(input_data)
    return input_np

training_data = './set1.txt'
test_data = './test1.txt'
large_120_data = './set2.txt'
input_test_data='./testInput.txt'
set1 = readFile(training_data)

test1 = readFile(test_data)

set2 = readFile(large_120_data)

test2=readFile(input_test_data)

#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

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

```

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

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

- x,y,a[x= height, y= weight, a=age] > for 14 data points
- x2,v2,a2=[same as above] > for 120 data points
- tx,ty,ta=[same as above] > for given test data point

```
x=[]
y=[]
a=[]
g=[]
x2=[]
y2=[]
a2=[]
g2=[]
tx=[]
ty=[]
ta=[]
```

```
FileIOOprations = open("set1.csv","w")
FileIOOprations.write("height,weight,age,gender\n")
```

```
25
```

```
FileIOOprations2 = open("set2.csv","w")
FileIOOprations2.write("height,weight,age,gender\n")
```

```
25
```

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

```
18
```

```
for index in set1Data:
  FormatedList=index[3:].split(' ')
  FormatedList=FormatedList[0:-1]
  FinalString=(FormatedList[0][:-1])+','+(FormatedList[1][:-1])+','+(FormatedList[2][:-2])+','+FormatedList[3]+'\n'
  x.append(float(FormatedList[0][:-1]))
  y.append(float(FormatedList[1][:-1]))
  a.append(int(FormatedList[2][:-2]))
  g.append((FormatedList[3]))
  # print(FinalString)
  FileIOOprations.write(FinalString)

```

```
for index in set2Data:
  FormatedList=index[3:].split(' ')
  FormatedList=FormatedList[0:-1]
  FinalString=(FormatedList[0][:-1])+','+(FormatedList[1][:-1])+','+(FormatedList[2][:-2])+','+FormatedList[3]+'\n'
  x2.append(float(FormatedList[0][:-1]))
  y2.append(float(FormatedList[1][:-1]))
  a2.append(int(FormatedList[2][:-2]))
  g2.append((FormatedList[3]))
  FileIOOprations2.write(FinalString)
```

```
for i in range(len(setTestData)):
  FormatedList=setTestData[i][2:].split(' ')
  if i<len(setTestData)-1:
    FormatedList=FormatedList[0:-1]
  else:
    FormatedList=FormatedList
  FinalString=(FormatedList[0][:-1])+','+(FormatedList[1][:-1])+','+(FormatedList[2][:-1])+'\n'
  tx.append(float(FormatedList[0][:-1]))
  ty.append(float(FormatedList[1][:-1]))
  ta.append(int(FormatedList[2][:-1]))
  FileIOOprationsT.write(FinalString)

```

```
FileIOOprations.close()
```

```
FileIOOprations2.close()
```

```
FileIOOprationsT.close()
```

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

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

```
#Cartesian distance calculation for 14 data point
import math
def CartesianDistanceForTest1():
  ans=[]
  ans1=[]
  ans2=[]
  ans3=[]
  q0=queue[0]
  q1=queue[1]
  q2=queue[2]
  q3=queue[3]
  for i in range(len(x)):
    ans.append(math.sqrt((x[i]-q0[0])**2+(y[i]-q0[1])**2+(a[i]-q0[2])**2))
    ans1.append(math.sqrt((x[i]-q1[0])**2+(y[i]-q1[1])**2+(a[i]-q1[2])**2))
    ans2.append(math.sqrt((x[i]-q2[0])**2+(y[i]-q2[1])**2+(a[i]-q2[2])**2))
    ans3.append(math.sqrt((x[i]-q3[0])**2+(y[i]-q3[1])**2+(a[i]-q3[2])**2))
  return ans,ans1,ans2,ans3
#uncomment below line to see Cartesian distance
# CartesianDistanceForTest1()
```

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

```
def CartesianDistanceForTest2():
  masterAns=[]
  for i in range(len(test2)):
    ans=[]
    for j in range(len(x)):
      ans.append(dis(x[j],float(test2[i][0]),y[j],float(test2[i][1]),a[j],int(test2[i][2])))
    masterAns.append(ans)
  return masterAns
```

```
# Cartesian prediction for 14 data point
def CartesianPredictionForTest1(inpu):
  allans=CartesianDistanceForTest1()
  ans=allans[0]
  ans1=allans[1]
  ans2=allans[2]
  ans3=allans[3]
  c=math.ceil(inpu/2)
  l1=sorted(ans)[:inpu]
  l2=sorted(ans1)[:inpu]
  l3=sorted(ans2)[:inpu]
  l4=sorted(ans3)[:inpu]
  i1=[]
  i2=[]
  i3=[]
  i4=[]
  m1=0
  m2=0
  m3=0
  m4=0
  f1=0
  f2=0
  f3=0
  f4=0
  answer=[]
  for i in range(0,inpu):
    i1.append(ans.index(l1[i]))
    i2.append(ans1.index(l2[i]))
    i3.append(ans2.index(l3[i]))
    i4.append(ans3.index(l4[i]))
    if g[i1[i]] == 'M':
      m1+=1
    else:
      f1+=1
    if g[i2[i]] == 'M':
      m2+=1
    else:
      f2+=1
    if g[i3[i]] == 'M':
      m3+=1
    else:
      f3+=1
    if g[i4[i]] == 'M':
      m4+=1
    else:
      f4+=1
    if m1>=c:
      answer.append('M')
    else:
      answer.append('W')
    if m2>=c:
      answer.append('M')
    else:
      answer.append('W')
    if m3>=c:
      answer.append('M')
    else:
      answer.append('W')
    if m4>=c:
      answer.append('M')
    else:
      answer.append('W')
    return answer
```

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

```
  for i in range(len(allans)):
    sortedList.append(sorted(allans[i])[:inpu])
  l1=sorted(allans[0])[:inpu]
  # print(sortedList)
  answer=[]
  masterIndexAns=[]
  for i in range(len(allans)):
    ans=[]
    m,f=0,0
    for j in range(inpu):
      ans.append(allans[i].index(sortedList[i][j]))
      if g[ans[j]] == 'M':
        m+=1
      else:
        f+=1
    if m>=c:
      answer.append('M')
    else:
      answer.append('W')
    # print(answer,m,f)
    masterIndexAns.append(ans)
  return answer
  # print(masterIndexAns)
```

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

```
#Manhattan Dsiatance for 14 data point
import math
def ManhattanDistanceForTest1():
  ans=[]
  ans1=[]
  ans2=[]
  ans3=[]
  p = ([x[0],y[0],a[0]])
  q=[1.62065758929,59.376557437583,32]
  q1=[1.7793983848363, 72.071775670801, 36]
  q2=[1.7004576585974, 66.267508112786, 31]
  q3=[1.6591086215159, 61.751621901787, 29]
  for i in range(len(x)):
    t=[]
    t1=[]
    t2=[]
    t3=[]
    t = (abs(x[i]-q[0])+abs(y[i]-q[1])+abs(a[i]-q[2]))
    ans.append(t)
    t1=(abs(x[i]-q1[0])+abs(y[i]-q1[1])+abs(a[i]-q1[2]))
    ans1.append(t1)
    t2=(abs(x[i]-q2[0])+abs(y[i]-q2[1])+abs(a[i]-q2[2]))
    ans2.append(t2)
    t3=(abs(x[i]-q3[0])+abs(y[i]-q3[1])+abs(a[i]-q3[2]))
    ans3.append(t3)
  return ans,ans1,ans2,ans3
```

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

```
# manhattan prediction for 14 data point
def ManhattanPredictionForTest1(inpu):
  allans=ManhattanDistanceForTest1()
  ans=allans[0]
  ans1=allans[1]
  ans2=allans[2]
  ans3=allans[3]
  c=math.ceil(inpu/2)
  l1=sorted(ans)[:inpu]
  l2=sorted(ans1)[:inpu]
  l3=sorted(ans2)[:inpu]
  l4=sorted(ans3)[:inpu]
  i1=[]
  i2=[]
  i3=[]
  i4=[]
```

```
  m1=0
  m2=0
  m3=0
  m4=0
  f1=0
  f2=0
  f3=0
  f4=0
  answer=[]
  for i in range(0,inpu):
    i1.append(ans.index(l1[i]))
    i2.append(ans1.index(l2[i]))
    i3.append(ans2.index(l3[i]))
    i4.append(ans3.index(l4[i]))
    if g[i1[i]] == 'M':
      m1+=1
    else:
      f1+=1
    if g[i2[i]] == 'M':
      m2+=1
    else:
      f2+=1
    if g[i3[i]] == 'M':
      m3+=1
    else:
      f3+=1
    if g[i4[i]] == 'M':
      m4+=1
    else:
      f4+=1
  if m1>=c:
    answer.append('M')
  else:
    answer.append('W')
  if m2>=c:
    answer.append('M')
  else:
    answer.append('W')
  if m3>=c:
    answer.append('M')
  else:
    answer.append('W')
  if m4>=c:
    answer.append('M')
  else:
    answer.append('W')
  return answer
```

```
# manhattan prediction for 120 data point
def ManhattanPredictionForTest2(inpu):
  allans=[]
  allans=ManhattanDistanceForTest2()
  c=math.ceil(inpu/2)
  sortedList=[]
  for i in range(len(allans)):
    sortedList.append(sorted(allans[i])[:inpu])
  l1=sorted(allans[0])[:inpu]
  answer=[]
  masterIndexAns=[]
  for i in range(len(allans)):
    ans=[]
    m,f=0,0
    for j in range(inpu):
      ans.append(allans[i].index(sortedList[i][j]))
      if g[ans[j]] == 'M':
        m+=1
      else:
        f+=1
    if m>=c:
      answer.append('M')
    else:
      answer.append('W')
    masterIndexAns.append(ans)
  return answer
```

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

```
#Minkowski Distance
import math
def MinkowskiDistanceForTest1():
```

```
  ans=[]
  ans1=[]
  ans2=[]
  ans3=[]

  p = ([x[0],y[0],a[0]])
  q0=[1.62065758929,59.376557437583,32]
  q1=[1.7793983848363, 72.071775670801, 36]
  q2=[1.7004576585974, 66.267508112786, 31]
  q3=[1.6591086215159, 61.751621901787, 29]
  for i in range(len(x)):
    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))
    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))
    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))
    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))
  return ans,ans1,ans2,ans3


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

```
def MinkowskiPredictionForTest1(inpu):
  allans=MinkowskiDistanceForTest1()
  ans=allans[0]
  ans1=allans[1]
  ans2=allans[2]
  ans3=allans[3]
  c=math.ceil(inpu/2)
  l1=sorted(ans)[:inpu]
  l2=sorted(ans1)[:inpu]
  l3=sorted(ans2)[:inpu]
  l4=sorted(ans3)[:inpu]
  i1=[]
  i2=[]
  i3=[]
  i4=[]
  m1=0
  m2=0
  m3=0
  m4=0
  f1=0
  f2=0
  f3=0
  f4=0
  answer=[]
  for i in range(0,inpu):
    i1.append(ans.index(l1[i]))
    i2.append(ans1.index(l2[i]))
    i3.append(ans2.index(l3[i]))
    i4.append(ans3.index(l4[i]))
    if g[i1[i]] == 'M':
      m1+=1
    else:
      f1+=1
    if g[i2[i]] == 'M':
      m2+=1
    else:
      f2+=1
    if g[i3[i]] == 'M':
      m3+=1
    else:
      f3+=1
    if g[i4[i]] == 'M':
      m4+=1
    else:
      f4+=1
  if m1>=c:
    answer.append('M')
  else:
    answer.append('W')
  if m2>=c:
    answer.append('M')
  else:
    answer.append('W')
  if m3>=c:
    answer.append('M')
```

```
  else:
    answer.append('W')
  if m4>=c:
    answer.append('M')
  else:
    answer.append('W')
  return answer
```

```
def MinkowskiPredictionForTest2(inpu):
  allans=[]
  allans=MinkowskiDistanceForTest2()
  # print(allans[0])
  c=math.ceil(inpu/2)
  sortedList=[]
  for i in range(len(allans)):
    sortedList.append(sorted(allans[i])[:inpu])
  l1=sorted(allans[0])[:inpu]
  # print(sortedList)
  answer=[]
  masterIndexAns=[]
  for i in range(len(allans)):
    ans=[]
    m,f=0,0
    for j in range(inpu):
      ans.append(allans[i].index(sortedList[i][j]))
      if g[ans[j]] == 'M':
        m+=1
      else:
        f+=1
    if m>=c:
      answer.append('M')
    else:
      answer.append('W')
    # print(answer,m,f)
    masterIndexAns.append(ans)
  return answer
  # print(masterIndexAns)
```

** Question: **1** C**

```
# data formatting for as per leave-out condition
masterans=[]
masterGen=[]
masterTrainingDataX=[]
masterTrainingDataY=[]
masterTrainingDataA=[]
for i in range(len(x2)):
  testx=0
  testy=0
  testa=0
  trainx=[]
  trainy=[]
  traina=[]
  traing=[]
  ans=[]
  for j in range(len(x2)):
    if i==j:
      testx=x2[j]
      testy=y2[j]
      testa=a2[j]
    else:
      trainx.append(x2[j])
      trainy.append(y2[j])
      traina.append(a2[j])
      traing.append(g2[j])
  for k in range(len(x2)-1):
    temp=dis(trainx[k],testx,trainy[k],testy,traina[k],testa)
    ans.append(temp)
  masterTrainingDataX.append(trainx)
  masterTrainingDataY.append(trainy)
  masterTrainingDataA.append(traina)
  masterans.append(ans)
  masterGen.append(traing)
```

```
import math
def LeaveOutPrediction(inpu):
  sucess=0
  for i in range(0,len(masterGen)):
    tempArr=masterans[i]
    tempgen=masterGen[i]
```

```
    # print(len(tempArr),len(tempgen))
    for j in range(len(masterGen)-1):
      ans=tempArr
      c=math.ceil(inpu/2)
      l1=sorted(ans)[:inpu]
      i1=[]
      m=0
      f=0
      answerr=[]
      for k in range(inpu):
        i1.append(ans.index(l1[k]))
        if tempgen[i1[k]] == 'M':
          m+=1
        else:
          f+=1
    if m>=c and g2[i] == 'M':
      sucess+=1
      answerr.append('M')
    elif f>=c and g2[i] == 'W':
      sucess+=1
      answerr.append('W')
    # print(m,f,sucess,g2[i])
  print("Acurracy for k =",inpu,"is",(sucess/len(x2))*100,"%")
  print("Erroneous for k =",inpu,"is",(1-sucess/len(x2))*100,"%")
```

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 algorith and see what will be accuracy

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

```
#age removing and data fornmatting
masterAns=[]
masterGender=[]
for i in range(len(x2)):
  testx=0
  testy=0
  trainx=[]
  trainy=[]
  traing=[]
  ans=[]
  for j in range(len(x2)):
    if i==j:
      testx=x2[j]
      testy=y2[j]
    else:
      trainx.append(x2[j])
      trainy.append(y2[j])
      traing.append(g2[j])
  for k in range(len(x2)-1):
    temp=dist(trainx[k],testx,trainy[k],testy)
    ans.append(temp)
  masterAns.append(ans)
  masterGender.append(traing)
```

```
import math
def LeaveOutPredictionFor1D(inpu):
  sucess=0
  for i in range(0,len(masterGen)):
    tempArr=masterAns[i]
    tempgen=masterGender[i]
    # print(len(tempArr),len(tempgen))
    for j in range(len(masterGen)-1):
      ans=tempArr
      c=math.ceil(inpu/2)
      l1=sorted(ans)[:inpu]
      i1=[]
      m=0
      f=0
```

```
      answerr=[]
      for k in range(inpu):
        i1.append(ans.index(l1[k]))
        if tempgen[i1[k]] == 'M':
          m+=1
        else:
          f+=1
    if m>=c and g2[i] == 'M':
      sucess+=1
      answerr.append('M')
    elif f>=c and g2[i] == 'W':
      sucess+=1
      answerr.append('W')
    # print(m,f,sucess,g2[i])
  print("Acurracy when age data is removed for k =",inpu,"is",(sucess/len(x2))*100,"%")
  print("Erroneous when age data is removed for k =",inpu,"is",(1-sucess/len(x2))*100,"%")
```

Finished Quetion 1 Here..... Hurrayyyyyyy

```
#mean calculation
import math
def mean(arr):
  su=sum(arr)
  ave=su/len(arr)
  return ave
```

```
# standard devation calculation
import math
def sd(arr):
    me = mean(arr)
    var = sum((i - me)**2 for i in arr) / (len(arr)-1)
    sdev = pow(var,0.5)
    return sdev

```

```
#for M height
import math
def hForM():
  arrHM=[]
  for i in range(len(x)):
    if g[i] == 'M':
      arrHM.append(x[i])
  return arrHM
```

```
#for M weight
import math
def wForM():
  arrHM=[]
  for i in range(len(x)):
    if g[i] == 'M':
      arrHM.append(y[i])
  return arrHM
```

```
#for M age
import math
def aForM():
  arrHM=[]
  for i in range(len(x)):
    if g[i] == 'M':
      arrHM.append(a[i])
  return arrHM
```

```
#for W height
import math
def hForW():
  arrHW=[]
  for i in range(len(x)):
    if g[i] == 'W':
      arrHW.append(x[i])
  return arrHW
```

```
#for W weight
import math
def wForW():
  arrHW=[]
  for i in range(len(x)):
```

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

```
#for W age
import math
def aForW():
  arrHW=[]
  for i in range(len(x)):
    if g[i] == 'W':
      arrHW.append(a[i])
  return arrHW
```

```
meanHM=mean(hForM())
meanWM=mean(wForM())
meanAM=mean(aForM())
meanHW=mean(hForW())
meanWW=mean(wForW())
meanAW=mean(aForW())
sdHM=sd(hForM())
sdWM=sd(wForM())
sdAM=sd(aForM())
sdHW=sd(hForW())
sdWW=sd(wForW())
sdAW=sd(aForW())
# 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

```
# normal distrubution calculation for gussian naive baysen for Women
def naiveBayseW(h,w,a):
  ans=0
  con=1/math.sqrt(2*math.pi*(sdAW**2))
  e=-(((a - meanAW)**2)/(2*(sdAW**2)))
  con2=math.exp(e)
  aw=con*con2

  con=1/math.sqrt(2*math.pi*(sdHW**2))
  e=-(((h - meanHW)**2)/(2*(sdHW**2)))
  con2=math.exp(e)
  hw=con*con2

  con=1/math.sqrt(2*math.pi*(sdWW**2))
  e=-(((w- meanWW)**2)/(2*(sdWW**2)))
  con2=math.exp(e)
  ww=con*con2
  ans=ww*aw*hw
  return ans

```

```
# normal distrubution calculation for gussian naive baysen for Men
def naiveBayseM(h,w,a):
  ans=0
  con=1/math.sqrt(2*math.pi*(sdAM**2))
  e=-(((a - meanAM)**2)/(2*(sdAM**2)))
  con2=math.exp(e)
  am=con*con2
  con=1/math.sqrt(2*math.pi*(sdHM**2))
  e=-(((h - meanHM)**2)/(2*(sdHM**2)))
  con2=math.exp(e)
  hm=con*con2

  con2=math.exp(-(((w - meanWM)**2)/(2*(sdWM**2))))/math.sqrt(2*math.pi*(sdWM**2))
  ansm=am*hm*con2
  return ansm
```

```
#prediction based on prediction answer
def naiveBaysespredictionForTest1():
  ansArr=[]
```

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

```
##prediction based on prediction answer for diffrent test inputs from testInput.txt file
def naiveBaysespredictionForTest2():
  ansArr=[]
  for i in range(len(test2)):
    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]
      ansArr.append("M")
    else:
      ansArr.append("W")
  print("Naive bayses prediction answer from testInput.txt fil  2 B : ",ansArr)
```

Data formatting and mean , sd calucation for leave out

```
#for M Height for C
import math
def hmForTest2():
  arr=[]
  for i in range(len(x2)):
    ans=[]
    for j in range(len(masterTrainingDataX[0])):
      if masterGender[i][j] == 'M':
        ans.append(masterTrainingDataX[i][j])
    arr.append(ans)
  # print(ans)
  return arr
```

```
#for M weight for c
import math
def wmForTest2():
  arr=[]
  for i in range(len(x2)):
    ans=[]
    for j in range(len(masterTrainingDataX[0])):
      if masterGender[i][j] == 'M':
        ans.append(masterTrainingDataY[i][j])
    arr.append(ans)
  # print(ans)
  return arr
```

```
#for M weight for C
import math
def amForTest2():
  arr=[]
  for i in range(len(x2)):
    ans=[]
    for j in range(len(masterTrainingDataX[0])):
      if masterGender[i][j] == 'M':
        ans.append(masterTrainingDataA[i][j])
    arr.append(ans)
  # print(ans)
  return arr
```

```
#for W height for C
import math
def hwForTest2():
  arr=[]
  for i in range(len(x2)):
    ans=[]
    for j in range(len(masterTrainingDataX[0])):
      if masterGender[i][j] == 'W':
        ans.append(masterTrainingDataX[i][j])
    arr.append(ans)
  # print(ans)
  return arr
```

```
#for W weight for C
import math
def wwForTest2():
  arr=[]
  for i in range(len(x2)):
```

```
    ans=[]
    for j in range(len(masterTrainingDataX[0])):
      if masterGender[i][j] == 'W':
        ans.append(masterTrainingDataY[i][j])
    arr.append(ans)
  # print(ans)
  return arr
```

```
#for W age for C
import math
def awForTest2():
  arr=[]
  for i in range(len(x2)):
    ans=[]
    for j in range(len(masterTrainingDataX[0])):
      if masterGender[i][j] == 'W':
        ans.append(masterTrainingDataA[i][j])
    arr.append(ans)
  # print(ans)
  return arr
```

```
#MEAN CALCULATION FOR 2 C
import math
def meanTest2(arr):
  ans=[]
  k=0
  for i in range(len(arr)):
    su=sum(arr[i])
    ave = su/len(arr[i])
    k+=1
    ans.append(ave)
  return ans
```

```
#SD CALCULATION FOR 2 C
import math
def sdTest2(arr):
  ans=[]
  k=0
  for i in range(len(arr)):
    me = mean(arr[i])
    var = sum((i - me)**2 for i in arr[i]) / (len(arr[i])-1)
    sdev = pow(var,0.5)
    k+=1
    ans.append(sdev)
  return ans
```

```
meanHMTest2=[]
meanAMTest2=[]
meanWMTest2=[]
meanHWTest2=[]
meanWWTest2=[]
meanAWTest2=[]
sdHMTest2=[]
sdWMTest2=[]
sdAMTest2=[]
sdHWTest2=[]
sdWWTest2=[]
sdAWTest2=[]

meanHMTest2=meanTest2(hmForTest2())
meanWMTest2=meanTest2(wmForTest2())
meanAMTest2=meanTest2(amForTest2())
meanHWTest2=meanTest2(hwForTest2())
meanWWTest2=meanTest2(wwForTest2())
meanAWTest2=meanTest2(awForTest2())
sdHMTest2=sdTest2(hmForTest2())
sdWMTest2=sdTest2(wmForTest2())
sdAMTest2=sdTest2(amForTest2())
sdHWTest2=sdTest2(hwForTest2())
sdWWTest2=sdTest2(wwForTest2())
sdAWTest2=sdTest2(awForTest2())

#ALL NESSASARY VARIBLES FOR CALCULATION(EG. meanHMTest2= mean for height label for men for 2C )
```

```
#Gussian naive baysen calculation for 2C for Women
def naiveBayseWTest2():
  meanAW=meanAWTest2
```

```
  meanHW=meanHWTest2
  meanWW=meanWWTest2
  sdAW=sdAWTest2
  sdHW=sdHWTest2
  sdWW=sdWWTest2
  ans=[]
  for i in range(len(sdAWTest2)):
    con=1/math.sqrt(2*math.pi*(float(sdAW[i])**2))
    e=-(((a2[i] - meanAW[i])**2)/(2*(sdAW[i]**2)))
    con2=math.exp(e)
    aw=con*con2

    con=1/math.sqrt(2*math.pi*(sdHW[i]**2))
    e=-(((x2[i] - meanHW[i])**2)/(2*(sdHW[i]**2)))
    con2=math.exp(e)
    hw=con*con2

    con=1/math.sqrt(2*math.pi*(sdWW[i]**2))
    e=-(((y2[i]- meanWW[i])**2)/(2*(sdWW[i]**2)))
    con2=math.exp(e)
    ww=con*con2
    ans.append(ww*aw*hw)
  return ans
```

```
#Gussian naive baysen calculation for 2C for Men
def naiveBayseMTest2():
  meanAM=meanAMTest2
  meanHM=meanHMTest2
  meanWM=meanWMTest2
  sdAM=sdAMTest2
  sdHM=sdHMTest2
  sdWM=sdWMTest2
  ans=[]
  for i in range(len(sdAMTest2)):
    # print(i)
    con=1/math.sqrt(2*math.pi*(float(sdAM[i])**2))
    e=-(((a2[i] - meanAM[i])**2)/(2*(sdAM[i]**2)))
    con2=math.exp(e)
    aw=con*con2

    con=1/math.sqrt(2*math.pi*(sdHM[i]**2))
    e=-(((x2[i] - meanHM[i])**2)/(2*(sdHM[i]**2)))
    con2=math.exp(e)
    hw=con*con2

    con=1/math.sqrt(2*math.pi*(sdWM[i]**2))
    e=-(((y2[i]- meanWM[i])**2)/(2*(sdWM[i]**2)))
    con2=math.exp(e)
    ww=con*con2
    ans.append(ww*aw*hw)
  return ans

```

```
#prediction ans for leave out array for 2 C
def naiveBaysespredictionFor1C():
  ansArr=[]
  for i in range(len(x2)):
    if naiveBayseMTest2()[i] > naiveBayseWTest2()[i]:
      ansArr.append("M")
    else:
      ansArr.append("W")
  # print("Naive bayses prediction answer for 1 A : ",ansArr)
  return ansArr
```

Without age

```
#Gussian naive baysen calculation for 2C for Women
def naiveBayseWTest2():
  meanAW=meanAWTest2
  meanHW=meanHWTest2
  meanWW=meanWWTest2
  sdAW=sdAWTest2
  sdHW=sdHWTest2
  sdWW=sdWWTest2
  ans=[]
  for i in range(len(sdAWTest2)):

    con=1/math.sqrt(2*math.pi*(sdHW[i]**2))
    e=-(((x2[i] - meanHW[i])**2)/(2*(sdHW[i]**2)))
    con2=math.exp(e)
```

```
    hw=con*con2

    con=1/math.sqrt(2*math.pi*(sdWW[i]**2))
    e=-(((y2[i]- meanWW[i])**2)/(2*(sdWW[i]**2)))
    con2=math.exp(e)
    ww=con*con2
    ans.append(ww*hw)
  return ans
```

```
#Gussian naive baysen calculation for 2C for Men
def naiveBayseMTest2():
  meanAM=meanAMTest2
  meanHM=meanHMTest2
  meanWM=meanWMTest2
  sdAM=sdAMTest2
  sdHM=sdHMTest2
  sdWM=sdWMTest2
  ans=[]
  for i in range(len(sdAMTest2)):

    con=1/math.sqrt(2*math.pi*(sdHM[i]**2))
    e=-(((x2[i] - meanHM[i])**2)/(2*(sdHM[i]**2)))
    con2=math.exp(e)
    hw=con*con2

    con=1/math.sqrt(2*math.pi*(sdWM[i]**2))
    e=-(((y2[i]- meanWM[i])**2)/(2*(sdWM[i]**2)))
    con2=math.exp(e)
    ww=con*con2
    ans.append(ww*hw)
  return ans

```

```
#prediction ans for leave out array for 2 C
def naiveBaysespredictionFor1C():
  ansArr=[]
  for i in range(len(x2)):
    if naiveBayseMTest2()[i] > naiveBayseWTest2()[i]:
      ansArr.append("M")
    else:
      ansArr.append("W")
  # print("Naive bayses prediction answer for 1 A : ",ansArr)
  return ansArr
```

```
print("For K=1, metric = euclidean --> ",CartesianPredictionForTest1(1))
print("For K=1, metric = Manhattan --> ",ManhattanPredictionForTest1(1))
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']
```

```
print("For K=3, metric = euclidean --> ",CartesianPredictionForTest1(3))
print("For K=3, metric = Manhattan --> ",ManhattanPredictionForTest1(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']
```

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

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

```
#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
inp=int(input("Enter any K value to all function's result with that K value: "))

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

```

```
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

```
LeaveOutPrediction(1)
```

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

```
LeaveOutPrediction(3)
```

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

```
LeaveOutPrediction(7)
```

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

best answer for k=9

```
LeaveOutPrediction(9)
```

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

```
LeaveOutPrediction(11)
```

```
Acurracy for k = 11 is 59.166666666666664 %
Erroneous for k = 11 is 40.833333333333336 %
```

answer 1 D

Acurrcy is inscresed with lesser data in knn algorith. and for k=3 I got best answer when age data is missing which is 7% more acurracy than age is present for previous quetion.

```
LeaveOutPredictionFor1D(1)
```

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

```
LeaveOutPredictionFor1D(3)
```

```
Acurracy when age data is removed for k = 3 is 70.83333333333334 %
Erroneous when age data is removed for k = 3 is 29.166666666666664 %
```

```
LeaveOutPredictionFor1D(7)
```

```
Acurracy when age data is removed for k = 7 is 63.33333333333333 %
Erroneous when age data is removed for k = 7 is 36.66666666666667 %
```

```
LeaveOutPredictionFor1D(9)
```

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

```
LeaveOutPredictionFor1D(11)
```

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

```
Erroneous when age data is removed for k = 11 is 42.50000000000001 %
```

```
naiveBaysespredictionForTest1()
```

```
Naive bayses prediction answer for 2 A :  ['W', 'W', 'W', 'W']
```

```
naiveBaysespredictionForTest2()
```

```
Naive bayses prediction answer from testInput.txt fil  2 B :  ['W', 'W', 'W', 'W']
```

```
#calculating acurracy for 2c
def CheckNaiveBaysesPredictionAnsWithOriginalAns():
  l=naiveBaysespredictionFor1C()
  res=0
  for i in range(120):
    if l[i]==g2[i]:
      res+=1
  return res*100/120
print("acurracy for 2 C ",CheckNaiveBaysesPredictionAnsWithOriginalAns(),"%")
print("Error for 2 C without age ",100-CheckNaiveBaysesPredictionAnsWithOriginalAns(),"%")
```

```
acurracy for 2 C  69.16666666666667 %
Error for 2 C without age  30.83333333333333 %
```

```
#calculating acurracy for 2c
def CheckNaiveBaysesPredictionAnsWithOriginalAns():
  l=naiveBaysespredictionFor1C()
  res=0
  for i in range(120):
    if l[i]==g2[i]:
      res+=1
  return res*100/120
print("acurracy for 2 D without age ",CheckNaiveBaysesPredictionAnsWithOriginalAns(),"%")
print("Error for 2 D without age ",100-CheckNaiveBaysesPredictionAnsWithOriginalAns(),"%")
```

```
acurracy 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% increse in accuracy however for Gaussian Na¨ive Bayes Classification we got almost same result with 1% decline. So I can say that with more data Gaussian Na¨ive Bayes Classification perform better than KNN and it has lesser diffrance with less data as well.

Colab paid products - Cancel contracts here