In [ ]:

```
#LAB CYCLE 3

Write a program to learn a naïve Bayes classifier and use it to predict
data. Laplacian smoothing should be used. The learned classifier should
instances and the accuracy of prediction for the test instances should b
single program should train the classifier on the training set as well a
Data Set Description:
The task is to predict whether a citizen is happy to live in a city base
of the city as rated by the citizens in a scale of 1-5 during a survey.
Attribute Information:
D = decision/class attribute (D) with values 0 (unhappy) and 1 (happy) (
X1 = the availability of information about the city services (Column 2 o
X2 = the cost of housing
X3 = the overall quality of public schools
X4 = your trust in the local police
X5 = the maintenance of streets and sidewalks
X6 = the availability of social community events
```

In [6]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [9]:

```
dataset = pd.read_csv('data3.csv')
dataset
```

Out[9]:

| | D | X1 | X2 | X3 | X4 | X5 | X6 |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 3 | 3 | 3 | 4 | 2 | 4 |
| **1** | 0 | 3 | 2 | 3 | 5 | 4 | 3 |
| **2** | 1 | 5 | 3 | 3 | 3 | 3 | 5 |
| **3** | 0 | 5 | 4 | 3 | 3 | 3 | 5 |
| **4** | 0 | 5 | 4 | 3 | 3 | 3 | 5 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **124** | 1 | 5 | 2 | 4 | 4 | 2 | 3 |
| **125** | 0 | 5 | 3 | 3 | 4 | 4 | 5 |
| **126** | 0 | 5 | 3 | 3 | 4 | 4 | 4 |
| **127** | 0 | 3 | 2 | 3 | 3 | 5 | 4 |
| **128** | 0 | 4 | 1 | 3 | 3 | 3 | 4 |

129 rows × 7 columns

In [10]:

```
dtotal=dataset.shape[0]
dtotal
```

Out[10]:

```
129
```

In [11]:

```
fzero=dataset['D'][dataset['D']==0].count()
fzero
```

Out[11]:
```
59
```

In [15]:
```
fone=dataset['D'][dataset['D']==1].count()
fone
```

Out[15]:
```
70
```

In [13]:
```
pzero=fzero/dtotal
pzero
```

Out[13]:
```
0.5426356589147286
```

In [16]:
```
pone=fone/dtotal
pone
```

Out[16]:
```
0.5426356589147286
```

In [17]:
```
dzero=np.zeros((5,6))
done=np.zeros((5,6))
```

In [19]:
```
for i in range(0,6):
    for j in range(0,5):
        dzero[j][i]=dataset['X'+str(i+1)][dataset['D']==0][dataset['X'+s
dzero
```

Out[19]:
```
array([[ 1., 15.,  4.,  0.,  5.,  1.],
       [ 0., 14.,  7.,  5.,  9.,  1.],
       [16., 18., 34., 24., 15., 16.],
       [25., 10.,  9., 21., 22., 24.],
       [17.,  2.,  5.,  9.,  8., 17.]])
```

In [20]:
```
for i in range(0,6):
    for j in range(0,5):
        done[j][i]=dataset['X'+str(i+1)][dataset['D']==1][dataset['X'+st
done
```

Out[20]:
```
array([[ 0., 13.,  2.,  1.,  1.,  0.],
       [ 0., 23.,  9.,  1.,  8.,  0.],
       [ 7., 22., 25., 24., 11.,  5.],
       [18.,  8., 24., 30., 30., 28.],
       [45.,  4., 10., 14., 20., 37.]])
```

In [21]:
```
zeroprobzero=np.zeros((6))
oneprobzero=np.zeros((6))
```

In [28]:
```
for i in range(0,5):
    for j in range(0,6):
        if(dzero[i][j]==0):
            zeroprobzero[j] = zeroprobzero[j]+ 1

zeroprobzero
```

Out[28]:
```
array([5., 0., 0., 5., 0., 0.])
```

In [29]:
```
for i in range(0,5):
    for j in range(0,6):
        if(dzero[i][j]==0):
            oneprobzero[j] = oneprobzero[j]+ 1

oneprobzero
```

Out[29]:
```
array([1., 0., 0., 1., 0., 0.])
```

In [30]:
```
dzeroprob=np.zeros((5,6))
doneprob=np.zeros((5,6))
done
```

Out[30]:
```
array([[ 0., 13.,  2.,  1.,  1.,  0.],
       [ 0., 23.,  9.,  1.,  8.,  0.],
       [ 7., 22., 25., 24., 11.,  5.],
       [18.,  8., 24., 30., 30., 28.],
       [45.,  4., 10., 14., 20., 37.]])
```

In [32]:
```
for n in range(0,6):
    if(zeroprobzero[n]>0):
        for j in range(0,5):
            dzeroprob[j][n]=(dzero[j][n]+1)/(fzero+5)
    else:
        for j in range(0,5):
            dzeroprob[j][n]=dzero[j][n]/fzero
dzeroprob
```

Out[32]:
```
array([[0.02666667, 0.21428571, 0.05714286, 0.01333333, 0.07142857,
        0.01428571],
       [0.01333333, 0.2       , 0.1       , 0.08      , 0.12857143,
        0.01428571],
       [0.22666667, 0.25714286, 0.48571429, 0.33333333, 0.21428571,
        0.22857143],
       [0.34666667, 0.14285714, 0.12857143, 0.29333333, 0.31428571,
        0.34285714],
       [0.24      , 0.02857143, 0.07142857, 0.13333333, 0.11428571,
        0.24285714]])
```

In [33]:
```
for n in range(0,6):
    if(oneprobzero[n]>0):
        for j in range(0,5):
            doneprob[j][n]=(done[j][n]+1)/(fone+5)
    else:
        for j in range(0,5):
            doneprob[j][n]=done[j][n]/fone
doneprob
```

Out[33]:
```
array([[0.01333333, 0.18571429, 0.02857143, 0.02666667, 0.01428571,
        0.        ],
       [0.01333333, 0.32857143, 0.12857143, 0.02666667, 0.11428571,
        0.        ],
       [0.10666667, 0.31428571, 0.35714286, 0.33333333, 0.15714286,
        0.07142857],
       [0.25333333, 0.11428571, 0.34285714, 0.41333333, 0.42857143,
        0.4       ],
       [0.61333333, 0.05714286, 0.14285714, 0.2       , 0.28571429,
        0.52857143]])
```

In [35]:
```
testset=pd.read_csv('test3.csv')
testset
```

Out[35]:

| | D | X1 | X2 | X3 | X4 | X5 | X6 |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 5 | 1 | 4 | 4 | 4 | 5 |
| **1** | 0 | 5 | 2 | 2 | 4 | 4 | 5 |
| **2** | 0 | 5 | 3 | 5 | 4 | 5 | 5 |
| **3** | 1 | 3 | 4 | 4 | 5 | 1 | 3 |
| **4** | 1 | 5 | 1 | 5 | 5 | 5 | 5 |
| **5** | 1 | 4 | 3 | 3 | 4 | 4 | 4 |
| **6** | 1 | 5 | 5 | 1 | 1 | 5 | 1 |
| **7** | 0 | 4 | 4 | 4 | 4 | 1 | 3 |
| **8** | 1 | 5 | 2 | 3 | 4 | 4 | 3 |
| **9** | 0 | 5 | 3 | 3 | 1 | 3 | 5 |
| **10** | 1 | 5 | 2 | 3 | 4 | 2 | 5 |
| **11** | 1 | 5 | 3 | 3 | 4 | 4 | 5 |
| **12** | 0 | 4 | 3 | 3 | 4 | 4 | 5 |
| **13** | 0 | 5 | 3 | 2 | 5 | 5 | 5 |

In [36]:
```
ttotal=testset.shape[0]
ttotal
```

Out[36]:
```
14
```

In [39]:
```
tp=0
tn=0
fp=0
fn=0
for n in range(0,ttotal):
    a=1
    b=1
    for i in range(1,6):
        k=testset.at[n,'X'+str(i)]
        a=a*dzeroprob[k-1][i-1]
        if i==5:
            break
    a=a*pzero
    b=b*pone
    #print a
    #print b
    if(a>b):
        predict=0
    else:
        predict=1
     #print(d)
    d=testset.at[n,'D']
    #print(d)
    if(d==1 and predict==1):
        tp=tp+1
    elif(d==1 and predict==0):
        tn=tn+1
```

```
    elif(d==0 and predict==1):
        fp=fp+1
    else:
        fn=fn+1


#print("tp = ",tp)
#print("tn = ",tn)
#print("fp = ",fp)
#print("fn = ",fn)
```

In [40]:
```
confusion=np.array([[tp,fp],[fn,tn]])
confusion
```

Out[40]:
```
array([[7, 7],
       [0, 0]])
```

In [41]:
```
correctness=(tp+tn)/(tp+tn+fp+fn)
print("correctness = ",correctness)
```

```
correctness =  0.5
```

In [42]:
```
errorness=(fp+fn)/(tp+tn+fp+fn)
print("errorness=",errorness)
```

```
errorness= 0.5
```

In [43]:
```
prediction=tp/(tp+fp)
print("prediction=",prediction)
```

```
prediction= 0.5
```

In [44]:
```
recall=tp/(tp+fn)
print("recall=",recall)
```

```
recall= 1.0
```

In [ ]:
```

```