Exp 7 : Naive Bayes

```
def table(s,r):#cal count of selected tuples with yes and no
 l1=[]
 c=-1
 for x in r:
  c +=1
  if x==s:
   l1.append(st[c])
 pj = l1.count("yes")
 nj = l1.count("no")
 print(s,"for yes: ",pj,s,"for no : ",nj)
 return pj,nj
color=['red','red','red','yellow','yellow','yellow','yellow','yellow','red','red']
typ =['sports','sports','sports','sports','sports','SUV','SUV','SUV','SUV','sports',]
origin=['Domestic','Domestic','Domestic','Domestic','Imported','Imported','Imported','Domestic','Imported','Imported'
]
st=['yes','no','yes','no','yes','no','yes','no','no','yes']
ty=st.count('yes')#no of yes tuples
tn=st.count('no')#no of no tuples
py=ty/len(st) #p(yes/total no of tuples)
pn=tn/len(st) #p(no/total no of tuples)
print('yes/total no of tuples :',py,'| no/total no of tuples :',pn)
y,n=table("red",color) # X = color:red | type: SUV | origin :domestic
y1,n1=table('SUV',typ)
y2,n2= table("Domestic",origin)
pyx = (y*y1*y2*py)/(ty*ty*ty) #p(X/yes)
pnx = (n*n1*n2*pn)/(tn*tn*tn)#p(X/no)
print('for the tuple X = (color=red , type =SUV, origin = Domestic)')
print('p(x/yes) = ',pyx,' p(x/no) = ',pnx)
if pyx > pnx:
 print("yes has highest probability")
else:
 print("no has highest probability")
```

Exp 8: K-mean

```
#implementing k-mean algorithm
# k =int(input("no of cluster: "))
#enter length of list1
x = int(input("enter length : "))
dataset = [0] * x
for i in range(x):
 dataset[i]= int(input("enter dataset"))
list1 = dataset
m=list1
print("DATASET: ",m)
n= int(len(m))
# randomly selecting mean
m1 = list1[0]
m2= list1[n-1]
print("mean m1 :",m1)
print("mean m2 :",m2)
#first iteration
```

```
iteration = 1
p=[0]*x #declaring array
q=[0]*x
for i in range(n ):
 g = abs(m1-m[i])
 h = abs(m2-m[i])
 if g<h :
  p[i]=m[i]
 else:
  q[i]=m[i]
print("CLUSTER 1 p: ",p)
print("CLUSTER 2 q: ",q)
print("ITERATION NO : ",iteration)
#removing zero from clusters
q=list(filter(lambda num: num != 0, q))
p=list(filter(lambda num: num != 0, p))
print(p,q)
```

Exp 9: Apriori

```
data = [
        ['T100',['I1','I2','I5']],
        ['T200',['I2','I4']],
        ['T300',['I2','I3']],
        ['T400',['I1','I2','I4']],
        ['T500',['I1','I3']],
        ['T600',['I2','I3']],
        ['T700',['I1','I3']],
        ['T800',['I1','I2','I3','I5']],
        ['T900',['I1','I2','I3']]
        ]
init = []
for i in data:
    for q in i[1]:
        if(q not in init):
            init.append(q)
init = sorted(init)
print(init)

sp = 0.4
s = int(sp*len(init))
s

from collections import Counter
c = Counter()
for i in init:
    for d in data:
        if(i in d[1]):
            c[i]+=1
print("C1:")
for i in c:
    print(str([i])+": "+str(c[i]))
print()
l = Counter()
```

```
for i in c:
    if(c[i] >= s):
        l[frozenset([i])]+=c[i]
print("L1:")
for i in l:
    print(str(list(i))+": "+str(l[i]))
print()
pl = l
pos = 1
for count in range (2,1000):
    nc = set()
    temp = list(l)
    for i in range(0,len(temp)):
        for j in range(i+1,len(temp)):
            t = temp[i].union(temp[j])
            if(len(t) == count):
                nc.add(temp[i].union(temp[j]))
    nc = list(nc)
    c = Counter()
    for i in nc:
        c[i] = 0
        for q in data:
            temp = set(q[1])
            if(i.issubset(temp)):
                c[i]+=1
    print("C"+str(count)+":")
    for i in c:
        print(str(list(i))+": "+str(c[i]))
    print()
    l = Counter()
    for i in c:
        if(c[i] >= s):
            l[i]+=c[i]
    print("L"+str(count)+":")
    for i in l:
        print(str(list(i))+": "+str(l[i]))
    print()
    if(len(l) == 0):
        break
    pl = l
    pos = count
print("Result: ")
print("L"+str(pos)+":")
for i in pl:
    print(str(list(i))+": "+str(pl[i]))
print()

from itertools import combinations
for l in pl:
    c = [frozenset(q) for q in combinations(l,len(l)-1)]
    mmax = 0
    for a in c:
        b = l-a
        ab = l
        sab = 0
```

```
        sa = 0
        sb = 0
        for q in data:
            temp = set(q[1])
            if(a.issubset(temp)):
                sa+=1
            if(b.issubset(temp)):
                sb+=1
            if(ab.issubset(temp)):
                sab+=1
        temp = sab/sa*100
        if(temp > mmax):
            mmax = temp
        temp = sab/sb*100
        if(temp > mmax):
            mmax = temp
        print(str(list(a))+" -> "+str(list(b))+" = "+str(sab/sa*100)+"%")
        print(str(list(b))+" -> "+str(list(a))+" = "+str(sab/sb*100)+"%")
    curr = 1
    print("choosing:", end=' ')
    for a in c:
        b = l-a
        ab = l
        sab = 0
        sa = 0
        sb = 0
        for q in data:
            temp = set(q[1])
            if(a.issubset(temp)):
                sa+=1
            if(b.issubset(temp)):
                sb+=1
            if(ab.issubset(temp)):
                sab+=1
        temp = sab/sa*100
        if(temp == mmax):
            print(curr, end = ' ')
        curr += 1
        temp = sab/sb*100
        if(temp == mmax):
            print(curr, end = ' ')
        curr += 1
    print()
    print()
```

Exp 10: Page rank

```
import java.util.*;
import java.io.*;
public class PageRank {
public int path[][] = new int[10][10];
public double pagerank[] = new double[10];
public void calc(double totalNodes) {
double InitialPageRank;
double OutgoingLinks = 0;
```

```
double DampingFactor = 0.85;
double TempPageRank[] = new double[10];
int ExternalNodeNumber;
int InternalNodeNumber;
int k = 1; // For Traversing
int ITERATION_STEP = 1;
InitialPageRank = 1 / totalNodes;
System.out.printf(&quot; Total Number of Nodes :&quot; + totalNodes + &quot;\t Initial PageRank of All Nodes
:&quot; + InitialPageRank + &quot;\n&quot;);
// 0th ITERATION _ OR _ INITIALIZATION PHASE //
for (k = 1; k &lt;= totalNodes; k++) {
this.pagerank[k] = InitialPageRank;
}
System.out.printf(&quot;\n Initial PageRank Values , 0th Step \n&quot;);
for (k = 1; k &lt;= totalNodes; k++) {
System.out.printf(&quot; Page Rank of &quot; + k + &quot; is :\t&quot; + this.pagerank[k] + &quot;\n&quot;);
}
while (ITERATION_STEP &lt;= 2) // Iterations
{
// Store the PageRank for All Nodes in Temporary Array
for (k = 1; k &lt;= totalNodes; k++) {


TempPageRank[k] = this.pagerank[k];
this.pagerank[k] = 0;
}
for (InternalNodeNumber = 1; InternalNodeNumber &lt;= totalNodes; InternalNodeNumber++) {
for (ExternalNodeNumber=1;
ExternalNodeNumber &lt;= totalNodes;
ExternalNodeNumber++) {
if (this.path[ExternalNodeNumber][InternalNodeNumber] == 1) {
k = 1;
OutgoingLinks = 0; // Count the Number of Outgoing Links for each ExternalNodeNumber
while (k &lt;= totalNodes) {
if (this.path[ExternalNodeNumber][k] == 1) {
OutgoingLinks = OutgoingLinks + 1; // Counter for Outgoing Links
}
k = k + 1;
}
// Calculate PageRank
this.pagerank[InternalNodeNumber] += TempPageRank[ExternalNodeNumber] * (1 /
OutgoingLinks);
}
}
}
System.out.printf(&quot;\n After &quot; + ITERATION_STEP + &quot;th Step \n&quot;);
for (k = 1; k &lt;= totalNodes; k++)
System.out.printf(&quot; Page Rank of &quot; + k + &quot; is :\t&quot; + this.pagerank[k] + &quot;\n&quot;);
ITERATION_STEP = ITERATION_STEP + 1;
}
// Add the Damping Factor to PageRank
for (k = 1; k &lt;= totalNodes; k++) {
this.pagerank[k] = (1 - DampingFactor) + DampingFactor * this.pagerank[k];
}
// Display PageRank
```

System.out.printf("\n Final Page Rank : \n");

for (k = 1; k <= totalNodes; k++) {
System.out.printf(" Page Rank of " + k + " is :\t" + this.pagerank[k] + "\n");
}
}
public static void main(String args[]) {
int nodes, i, j, cost;
Scanner in = new Scanner(System.in);
System.out.println("Enter the Number of WebPages \n");
nodes = in .nextInt();
PageRank p = new PageRank();
System.out.println("Enter the Adjacency Matrix with 1->PATH & 0->NO PATH Between two WebPages: \n");
for (i = 1; i <= nodes; i++)
for (j = 1; j <= nodes; j++) {
p.path[i][j] = in .nextInt();
if (j == i)
p.path[i][j] = 0;
}
p.calc(nodes);
}
}