APRIORI ALGORITHM

```
import csv


def getValuesFromCsv(path):
    f = open(path,'r')
    data = csv.reader(f)
    csvData={}
    for row in data:
        csvData[row[0]]=row[1].split()
    return csvData

database = getValuesFromCsv("datasetApriori.txt")
print(database)


def createL1(db):
    L1={}
    for tranid in db:
        tran = db[tranid]
        print(tran)
        for item in tran:
            #print(item)
            if item in L1:
                L1[item]+=1
                #print(item,L1[item])
            else:
                L1[item] = 1
    return L1

L1=createL1(database)
print(L1)

def compare(itemset1,itemset2,sizeafterjoin):
    matchCount = 0
    for item in itemset1:
        if item in itemset2:
            matchCount += 1
    return(matchCount >= (sizeafterjoin-2))
compare(['1','2'],['3','4'],2)


# In[5]:


def createNewitemset(itemset1,itemset2):
    for item in itemset1:
        if item not in itemset2:
            itemset2.append(item)
    itemset2.sort()
    print(itemset2)
    return ",".join(itemset2)
```

```
X=createNewitemset(['1','2'],['2','3'])



def createL(itemsetlist,db):
    L = {}
    for itemset1 in itemsetlist:
        itemset2 = itemset1.split(",")
        count = 0
        for tranid in db:
            tran=db[tranid]
            flag=True
            for item in itemset2:
                if item not in tran:
                    flag = False
                    continue
            if(flag):
                count+=1
        L[itemset1]= count
    return L
createL(['1,2','1,3','2,3'],database)


def join(c,db,k):
    print("C:",c,"k:",k)
    itemsetlist = [*c.keys()]
    print(itemsetlist)
    itemsetlist.sort()
    print("Sorted itemset list:",itemsetlist)
    newitemsetlist=[]
    length=len(itemsetlist)
    print(length)

    for i in range(0,length):
        startitemset=itemsetlist[i]
        startitemset1=startitemset.split(",")
        for j in range(i+1,length):
            nextitemset=itemsetlist[j]
            nextitemset1=nextitemset.split(",")
            if(compare(startitemset1,nextitemset1,k)):
                newitemset = createNewitemset(startitemset1,nextitemset1)
                if newitemset not in newitemsetlist:
                    newitemsetlist.append(newitemset)

    for itemset10 in itemsetlist:
        itemset100 = itemset10.split(',')
        for itemset20 in itemsetlist:
            itemset200 = itemset20.split(",")

            if(compare(itemset100,itemset200,k)):
                newitemset = createNewitemset(itemset100,itemset200)
```

```
            if newitemset not in newitemsetlist:
                newitemsetlist.append(newitemset)
    l=createL(newitemsetlist,db)
    return l
join({'1,3,2':2,'1,3,4':3},database,4)



def prune(l,minSup):
    keysToDelete=[]
    for key in l:
        if(l[key]<minSup):
            keysToDelete.append(key)
    for key in keysToDelete:
        del(l[key])
    return l

def apriori(data,L1,minSup):
    kTables={}
    k=2
    print("l1",L1)
    c = prune(L1,minSup)
    print("c1:",c)
    kTables[1] = c
    while(True):
        l = join(c,data,k)
        print("l"+str(k)+":",1)
        c=prune(l,minSup)
        print("c"+str(k)+":",c)
        if(len(c)==0):
            break
        kTables[k]=c
        k+=1
    print("\nFinal Answer:")
    print(kTables[k-1])
apriori(database,L1,2)
```

K-NEAREST NEIGHBOUR

```
import csv
#import time
import random
# import math
import operator


# def random():
#     sec = time.time()
#     print(sec)

# random()


def sqrt(data):
    return data ** 0.5

def loadData(filename , split , trainingSet=[] , testSet=[]):
    with open(filename , 'r') as csvFile:
        lines = csv.reader(csvFile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])

def getDistance(data1, data2, length):
    dis = 0
    for x in range(length):
        dis += (data1[x] - data2[x]) ** 2
    return sqrt(dis)

def getNeighbors(trainingSet, testData, k):
    distances = []
    length = len(testData)-1
    for x in range(len(trainingSet)):
        dist = getDistance(testData, trainingSet[x], length)
        distances.append((trainingSet[x], dist))

    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

def getResponse(neighbors):
    Votes = {}
```

```
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in Votes:
            Votes[response] += 1
        else:
            Votes[response] = 1
    sortedVotes = sorted(Votes.items(), key=operator.itemgetter(1), reverse=True)
    return sortedVotes[0][0]


# def getAccuracy(testSet, predictions):
#     correct = 0
#     for x in range(len(testSet)):
#         if testSet[x][-1] == predictions[x]:
#             correct += 1
#     return (correct/float(len(testSet))) * 100.0

def kNearest():
    trainingSet=[]
    testSet=[]
    split = 0.7
    loadData('iris.data', split, trainingSet, testSet)
    print ('Train set: ' , len(trainingSet))
    print ('Test set: ' ,  len(testSet))
    predictions=[]
    k = 3
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
        print('=> predicted result = ' , repr(result) , ', actual result = ' , repr(testSet[x][-1]))
#     accuracy = getAccuracy(testSet, predictions)
#     print('Accuracy: ', repr(accuracy) , '%')

kNearest()
```

LINEAR REGRESSION

```
import csv

def getValues(path):
    f = open(path,'r')
    data = csv.reader(f)
    csvData={}
    for row in data:
        csvData[row[0]] = row[1].split();
    return csvData

data = getValues("linear2.csv")
print(data)



def Mean(db, index):
    x_sum = 0
    for id in db:
        tran = db[id]

        x_sum += float(tran[index])

    return (x_sum/len(db))

X_mean = Mean(data, 0)
print(X_mean)
Y_mean = Mean(data, 1)
print(Y_mean)


def SigmaXx_Yy(db):
    num = 0
    for id in db:
        tran = db[id]
        xX = float(tran[0]) - X_mean
        yY = float(tran[1]) - Y_mean
        prod = xX * yY
        #print(prod)
        num += prod
    return num

Xx_Yy = SigmaXx_Yy(data)
print(Xx_Yy)



def SigmaXx2(db):
    prod = 0
    for id in db:
        tran = db[id]
        sub = float(tran[0]) - X_mean
```

```
        prod += sub * sub
        #print(prod)
    return prod

Xx2 = SigmaXx2(data)
print(Xx2)


b1 = Xx_Yy / Xx2
print(b1)
b0 = Y_mean - (b1 * X_mean)
print(b0)


def linear(q=None):
    if q:
        pt = [q , b0 + b1 * q]
        return pt
    return

linear(10)

import numpy as np
from matplotlib import pyplot as plt
x = np.linespace(0,30,100)
y = b0 + b1*3

plt.scatter(x,y,color="green",marker="^")
plt.scatter(x_test,predictions,color="d")
```

NAIVE BAYES

```
import csv


Data_List=[]
with open ("naiveBayes.csv",'r',errors='ignore') as df:
    df=csv.reader(df)

    for i in df:
        Data_List.append(i)

    '''n=len(Data_List) #length of dataset
    m=len(Data_List[0])#length of one row
    #print(m,n)
    laplace=1   #laplace is used if probabilty of any attribute is 0
    for i in range(1,n):
        Data_List[i-1]=Data_List[i]'''
    print(Data_List)

len(Data_List)


#calc probability
def prob(f,name):
    count=0
    for i in range(0,len(Data_List)):
        if Data_List[i][f] == name:
            count+=1
    return(count/len(Data_List))

prob(4,'Yes')



prob(0,'Sunny')


def prob_cond(feature,f,label):
    new_list=[]
    n=4
    for i in range(0,len(Data_List)):
        if Data_List[i][4] == label:
            new_list.append(Data_List[i])

    if not len(new_list)==0:

        n_events=0
        for i in range(len(new_list)):
            if new_list[i][f] == feature:
                n_events+=1
        prob=n_events/len(new_list)
```

```
        print(new_list)
        print(prob)
        return(prob)
    else:
        return(0)



prob_cond('Sunny',0,'Yes')


X=['Rainy','Mild','Normal','False']# test data




X=['Sunny','Hot','Normal','False']# test data



l = len(X)

prob_yes = prob(l,'Yes')
#p(no)
prob_no = 1-prob_yes

prob_x_given_yes = 1

prob_x_given_no = 1

for i in range(0,len(X)):

    prob_testfeature_given_yes = prob_cond(X[i],i,'Yes')
    prob_x_given_yes = prob_x_given_yes * prob_testfeature_given_yes

for i in range(0,len(X)):

    prob_testfeature_given_no = prob_cond(X[i],i,'No')
    prob_x_given_no = prob_x_given_no * prob_testfeature_given_no

prob_x_given_yes*= prob_yes
prob_x_given_no*=prob_no


p_today=prob(0,X[0])*prob(1,X[1])*prob(2,X[2])*prob(2,X[2])



if not p_today==0:
    prob_x_given_yes = prob_x_given_yes /p_today
    prob_x_given_no = prob_x_given_no /p_today
    print("Probability for yes is ",prob_x_given_yes)
```

```
    print("Probability for no is ",prob_x_given_no)
else:
    print("000")

if (prob_x_given_yes>prob_x_given_no):
    print("YES")
else:
    print("NO")
```

K-MEANS CLUSTERING

```
data = [1,2,3,4,5,11,12,13,14,25]


import random
def init(K,data):
    centroid=[]
    clusters=[]
    for i in range(K):
        random_element=random.choice(data)
        while random_element in centroid:
            random_element=random.choice(data)

        centroid.append(random_element)
#         print(centroid)

    for i in range(K):
        clusters.append([centroid[i]])

    return [centroid,clusters]


initial_list=init(2,data)
centroid=initial_list[0]
clusters=initial_list[1]
print(centroid)



def which(element,k):
    for i in range(k):
        if element in clusters[i]:
            return i
    return -1

def traversal(K):
    change=False
    for each in data:
        which_c=which(each,K)
        if each not in centroid:


            distance=[]
            for dis in centroid:
                distance.append(abs(dis-each))

            min_d=100000
            cent=0
            for i in range(len(distance)):
                if distance[i]<min_d:
                    cent=i
```

```
                min_d=distance[i]

            for i in range(len(clusters)):
                if each in clusters[i]:
                    clusters[i].remove(each)
                    print(clusters[i])
            if cent==which_c and which_c is not -1:
                return True
            clusters[cent].append(each)
            centroid[cent]=(each+centroid[cent])/2

    return change




# def main():
#     change=True
#     while change is True:
# traversal(3)

# print(clusters,centroid)

change =True
while change is True:
    K=2
    change=traversal(K)
    print(centroid)
    print(clusters)
    print("----------------------------")
```

DATA MINING LAB FILE

DONE BY

ASHEEQUE C M

ROLL NO- 16BCS008

DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF ENGINEERING AND TECHNOLOGY
JAMIA MILLIA ISLAMIA