UM-SJTU Joint Institute
Problem Solving with AI Techniques
(Ve593)

Project Two
Bayes Net

Ming Xingyu 517370910224

Date: 2020/10/11

Part 1. BN

In this part, we are asked to train a model based on Bayes Network with the data provided in *protein.csv*. I defined a function *partition* to randomly separate the train set and the test set with ratio 7:3. With the different train methods using the same data, we can have two different structures and we do inference on each of them. The results will be show in the following parts. Also, I trained model without any mandatory arcs, it gives me a model that leave *nuc* independent. Note that we need to use all the five parameters to train our model, I add the

1.1 $useLocalSearchWithTabuList() + useAprioriSmoothing() + LazyPropagation()$

Following the procedure, we first learn the structure, which is shown below.

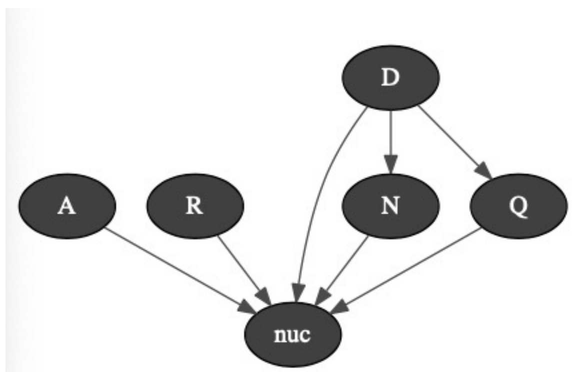

Figure 1: The structure learned by Tabulist.

Next, I learned the parameter and do inference, where I obtain the structure below.

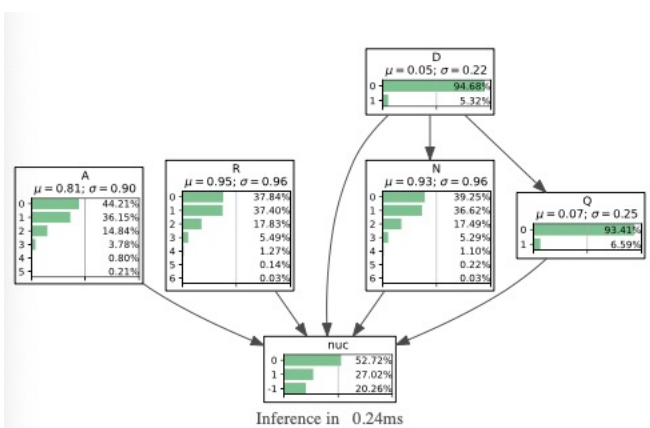

Figure 2: The structure learned by Tabulist with learned parameters.

Finally, we do inference on the test data and calculate the accuracy of our model.

$$acc_1 = 0.5276639344262295 = 52.8\%$$

1.2 $useGreedyHillClimbing()/useK2() + useAprioriSmoothing() + LazyPropagation()$

Similarly, we learned the structure of our BN as follows.

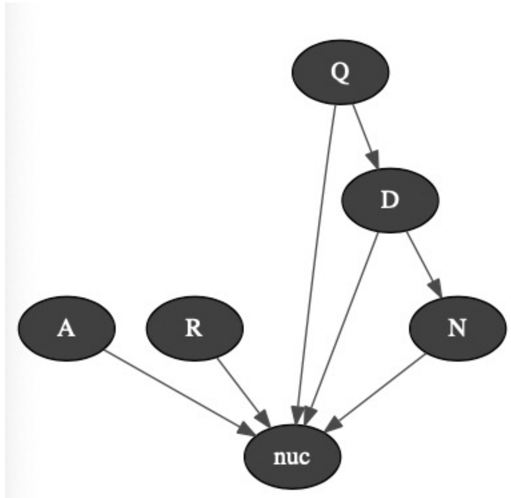

Figure 3: The structure learned by Hillclimbing and K2.

Next, I learned the parameter and do inference, where I obtain the structure below.

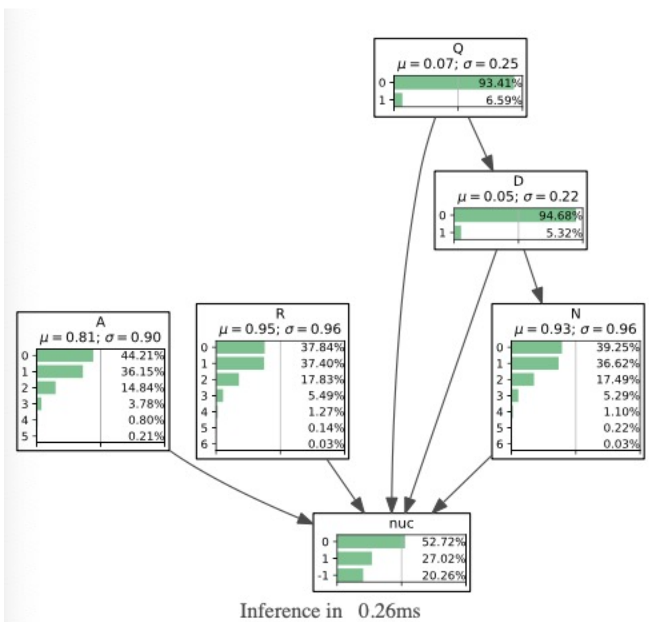

Figure 4: The structure learned by Hillclimbing and K2 with learned parameters.

Finally, we calculate the accuracy of our model.

$$acc_2 = 0.5276639344262295 = 52.8\%$$

1.3 Compare with no evidences

As I said before, since we are required to do inference based on the evidences, so we intuitively add the mandatory arcs in BN in the preceding parts. Removing all of these arcs, *nuc* is left independently, and we have the structure alone with the parameter as follows.

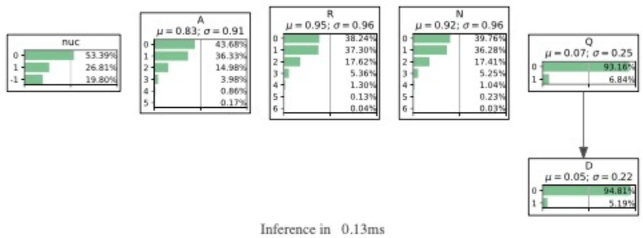

Figure 5: Structure that leave nuc alone.

Also, we do the inference on the same test data, which provide us with

$$acc = 0.5325884543761639 = 53.26\% > max\{acc_1, acc_2\}$$

Surprisingly, the the accuracy we obtained with complex structure of BN is less than only know the distribution of *nuc*. Hence, it can be conclude that predicting or diagnose this specific disease only depend on the evidence of A, R, N, Q, D can not be convincing, which also means to some extend, there is little connection between the features and nuc.

Part 2. DBN

2.1 1-order Markov

In this part, I trained the DBN with 1-order markov model for each of indices. The trained DBN is shown below. For convenience, at each time series, I used *Close, Open, High, Low, Volume* as the features. And I calculate the accuracy with

$$DBN(i,t) = P(R^i_{t+1} > 1|r_t) > 0.4$$

The above equation can be interpreted as, the close price at $t+1$ is larger than the close price at t ($R^i_{t+1} > 1$), with probability larger than 40%. Admittedly, this is not equivalent to the expression that $High_{t+1} > Close_t$, however, intuitively, it can guarantee the highest price must larger than the latest close price.

In this project, the way we do the discretize part is very subtle. I've done the discretize function differently for the prices and volume. The functions are shown below. The reason for this difference is that the result with the same discretizing strategy led to the $Volume$ independent. After several test, I finally decided to do the discretization in this way.

```python
def genbin(l):#discretize the price
    l_b=list(np.arange(0,math.ceil(max(l))+1,1))
    return list(pd.cut(l,bins=l_b,labels=False)),len(l_b)
def genbinv(v):#discretize the volume
    v_b=list(np.linspace(0,math.ceil(max(v)),2))
    return list(pd.cut(v,bins=v_b,labels=False)),len(v_b)
```

To make it a DBN, I have forbid the arcs from future to the past, with the help of the function **addForbiddenArc(future,past)**.

For each sector, the learned structure of it is shown in the following figures.

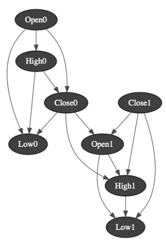

Figure 6: IDU

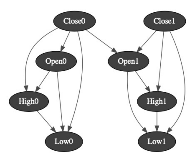

Figure 7: IHF

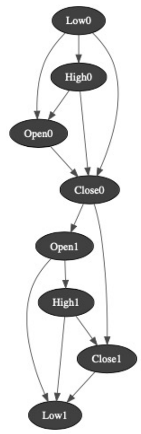

Figure 8: IYC

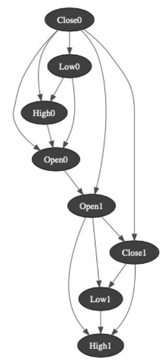

Figure 9: IYE

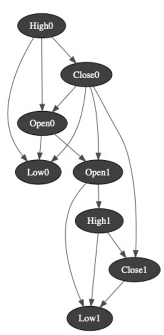

Figure 10: IYG

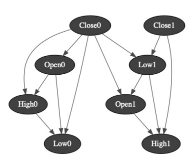

Figure 11: IYH

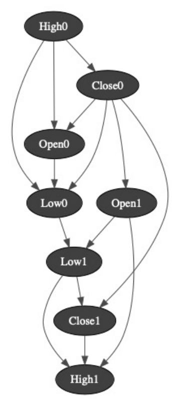

Figure 12: IYJ

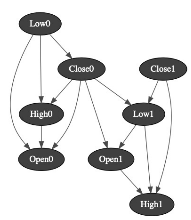

Figure 13: IYK

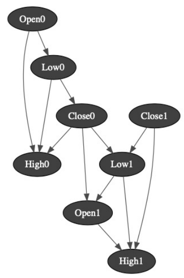

Figure 14: IYK

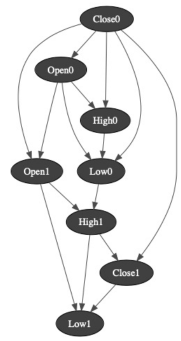

Figure 15: IYM

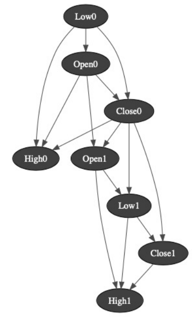

Figure 16: IYR

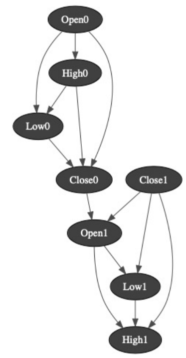

Figure 17: IYT

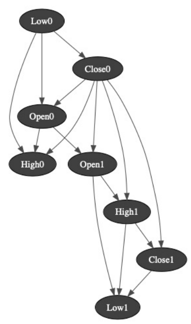

Figure 18: IYW

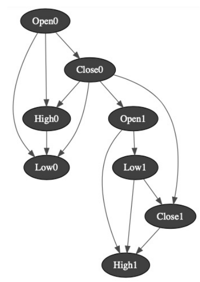

Figure 19: IYZ

And finally, I have calculate the accuracy of my model, which is shown as follows.

	Sector	accuracy	N
0	IDU	99	116
1	IHF	98	116
2	IYC	95	116
3	IYE	89	116
4	IYF	61	81
5	IYG	86	116
6	IYH	96	116
7	IYJ	88	116
8	IYK	89	116
9	IYM	88	116
10	IYR	63	77
11	IYT	90	116
12	IYW	95	116
13	IYZ	96	116

Table 1: The prediction acquired by 1-order Markov.

With the evaluation function provided in the project description, we can have

$$\frac{1}{N}\sum_{t\geq t_0}\sum_{i}\left[\text{DBN}(i,t) > \epsilon_{+}\text{and } \text{High}^{i}_{t+1} > \text{Close}^{i}_{i}\right] = 0.795$$

2.2 k-order Markov

To generalize our model, I design a function named **kmodel(filename,k)**. After several test with different input of order k, I found some of the structures learned have similar parts repeating themselves like 20, which means the order for this model is too high. Hence, I trained my model with different order and choose the best order.

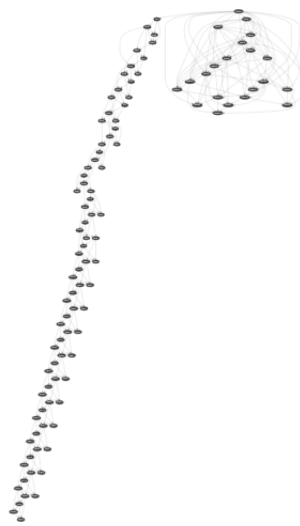

Figure 20: Structure with similar parts repeating.

I run the markov chain up to 20-order, and the results of the accuracy is shown in the following plot.

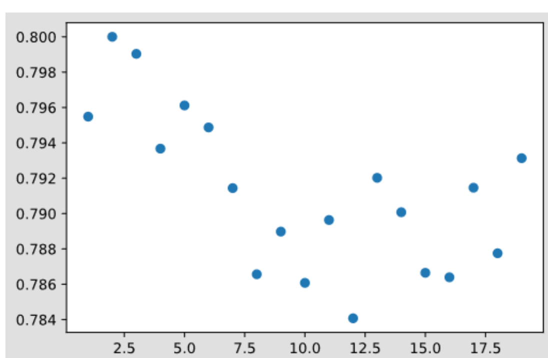

Figure 21: The result of evaluation of 1 to 20-order markov chain.

Apparently, the best choice of order is 2, given the strategy of my discretization. And the result is

$$\frac{1}{N}\sum_{t\geq t_0}\sum_{i}\left[\mathrm{DBN}(i,t)>\epsilon_{+}\text{and }\mathrm{High}^{i}_{t+1}>\mathrm{Close}^{i}_{i}\right]=0.8$$

Moreover, since the different orders do not provide us with a huge difference in the result, for convenience, the prediction work should be done by 2-order markov chain.

Appendix

BN

```
import csv
import random
import pyAgrum as gum
import pyAgrum.lib.notebook as gnb

def partition(filename):
    train=[['nuc', 'A', 'R', 'N', 'D', 'Q']]
    test=[['nuc', 'A', 'R', 'N', 'D', 'Q']]
    with open(filename+'.csv','r', encoding="utf-8") as csvfile:
        reader = csv.reader(csvfile)
        for line in reader:
            if line!=['nuc', 'A', 'R', 'N', 'D', 'Q']:
                t=random.random()
                if t <0.7:
                    train.append(line)
                else:
                    test.append(line)
    with open(filename+'_train.csv','w',encoding="utf-8") as csvfile:
        writer=csv.writer(csvfile)
        for line in train:
            writer.writerow(line)
    with open(filename+'_test.csv','w',encoding="utf-8") as csvfile:
        writer=csv.writer(csvfile)
        for line in test:
            writer.writerow(line)

def main():
    bn = gum.BayesNet('nuc_inf')
    #add variables to the network
```

```
va=gum.LabelizedVariable('nuc','a labelized variable',2)
va.addLabel('-1')
nuc = bn.add(va)
A = bn.add('A',6)
R,N = [bn.add(name,7) for name in ['R','N']]
D,Q = [bn.add(name,2) for name in ['D','Q']]
partition("protein")
learner = gum.BNLearner("protein_train.csv", bn)
#These arcs can be added or deleted
#learner.addMandatoryArc('A','nuc')
#learner.addMandatoryArc('R','nuc')
#learner.addMandatoryArc('Q','nuc')
#learner.addMandatoryArc('N','nuc')
#learner.addMandatoryArc('D','nuc')
learner.useLocalSearchWithTabuList()
bn0 = learner.learnBN()
gnb.showBN(bn0)
learner.useGreedyHillClimbing()
bn1 = learner.learnBN()
gnb.showBN(bn1)
learner.useK2([5,4,3,2,1,0])
bn2 = learner.learnBN()
gnb.showBN(bn2)
#We have 2 different BN structures according to the previous parts. Now, we do parameter learning
learner = gum.BNLearner("protein_train.csv", bn)
learner.setInitialDAG(bn0.dag())
learner.useAprioriSmoothing(1)
bn01 = learner.learnParameters()#first
gnb.showBN(bn01)
learner = gum.BNLearner("protein_train.csv", bn)
learner.setInitialDAG(bn2.dag())
learner.useAprioriSmoothing(1)
bn11 = learner.learnParameters()#second
gnb.showBN(bn11)
#first
ie1 = gum.LazyPropagation(bn01)
ie1.makeInference()
gnb.showInference(bn01,evs={})
```

```
    #second
    ie2 = gum.LazyPropagation(bn11)
    ie2.makeInference()
    gnb.showInference(bn11,evs={})
    with open('protein_test.csv','r', encoding="utf-8") as csvfile:
        reader = csv.reader(csvfile)
        count1=1
        count2=1
        acc1=0
        acc2=0
        for line in list(reader)[1:]:
            vnuc,vA,vR,vN,vD,vQ=[int(line[0]),int(line[1]),int(line[2]),int(line[3])
            #print(vnuc,vA,vR,vN,vD,vQ)
            ie2.eraseAllEvidence()
            ie1.eraseAllEvidence()
            ie1.setEvidence({'A':vA, 'R':vR,'N':vN, 'D': vD,'Q':vQ})
            ie2.setEvidence({'A':vA, 'R': vR,'N':vN, 'D': vD,'Q':vQ})
            ie1.makeInference()
            ie2.makeInference()
            ie2.addTarget(nuc)
            ie1.addTarget(nuc)
            if len(ie2.posterior(nuc).argmax())==1:#if we have one determined value of prob
                #print(ie2.posterior(nuc))
                #print(ie2.posterior(nuc).argmax()[0]['nuc'])
                if ie2.posterior(nuc).argmax()[0]['nuc']==2:#nuc=-1
                    if vnuc==-1:
                        acc2=acc2+1
                if ie2.posterior(nuc).argmax()[0]['nuc']==vnuc:
                    acc2=acc2+1
                count2=count2+1
            if len(ie1.posterior(nuc).argmax())==1:
                #print(ie1.posterior(nuc))
                #print(ie1.posterior(nuc).argmax()[0]['nuc'])
                if ie1.posterior(nuc).argmax()[0]['nuc']==2:
                    if vnuc==-1:
                        acc1=acc1+1
                if ie1.posterior(nuc).argmax()[0]['nuc']==vnuc:
```

```
                    acc1=acc1+1
                count1=count1+1
        acc2=acc2/count2
        acc1=acc1/count1
    print(acc2,acc1)

if __name__=='__main__':
    main()
```

DBN

```
import pyAgrum as gum
import numpy as np
import pandas as pd
import math
import pyAgrum.lib.notebook as gnb
import csv
import matplotlib.pyplot as plt

def genre(s):
    r=[]
    for i in range(1,len(s)):
        r.append(s[i]/s[i-1])
    return r

def genbin(l):#discretize the price
    l_b=list(np.arange(0,math.ceil(max(l))+1,1))
    return list(pd.cut(l,bins=l_b,labels=False)),len(l_b)

def genbinv(v):#discretize the volume
    v_b=list(np.linspace(0,math.ceil(max(v)),2))
    return list(pd.cut(v,bins=v_b,labels=False)),len(v_b)

def gentt(filename):#generate train data and test data return the number of variables
    df = pd.read_csv(filename+'.csv')
    df.dropna(axis=0, how='any', inplace=True)#drop the line with NAN in case there is missing data in the file
    Date=df['Date']
```

```
    index=list(Date).index('2015-11-13')#find the index of 2015-11-13, we need to slice the list later
        #generate returns and then discretized variables.
    Open,ob=genbin(genre(df['Open']))
    High,hb=genbin(genre(df['High']))
    Low,lb=genbin(genre(df['Low']))
    Close,cb=genbin(genre(df['Close']))
    Volume,vb=genbinv(genre(df['Volume']))
    train_Open=Open[:index]
    test_Open=Open[index-1:]
    train_High=High[:index]
    test_High=High[index-1:]
    train_Low=Low[:index]
    test_Low=Low[index-1:]
    train_Close=Close[:index]
    test_Close=Close[index-1:]
    train_Volume=Volume[:index]
    test_Volume=Volume[index-1:]
    train=pd.DataFrame()#The train data
    test=pd.DataFrame()#The test data
    train['Close0']=train_Close[0:-1]#at t-1
    train['Close1']=train_Close[1:]#at t
    train['Open0']=train_Open[0:-1]#at t-1
    train['Open1']=train_Open[1:]#at t
    train['High0']=train_High[0:-1]#at t-1
    train['High1']=train_High[1:]#at t
    train['Low0']=train_Low[0:-1]#at t-1
    train['Low1']=train_Low[1:]#at t
    train['Volume0']=train_Volume[0:-1]#at t-1
    train['Volume1']=train_Volume[1:]#at t
    test['Close0']=test_Close[0:-1]#at t-1
    test['Open0']=test_Open[0:-1]#at t-1
    test['High0']=test_High[0:-1]#at t-1
    test['Low0']=test_Low[0:-1]#at t-1
    test['Volume0']=test_Volume[0:-1]#at t-1
    #####Generate the boolean var for accuracy calculation
    true=[]
    h=list(df['High'])[index:]
    c=list(df['Close'])[index:]
```

```
    for i in range(len(h)-1):
        if c[i]<h[i+1]:
            true.append(1)
        else:
            true.append(0)
    test['true']=true
    train.set_index('Close0', inplace=True)
    train.to_csv(filename+'_train.csv')
    test.set_index('Close0', inplace=True)
    test.to_csv(filename+'_test.csv')
    return ob,hb,lb,cb,vb

def trainmodel(filename):
    ob,hb,lb,cb,vb=gentt(filename)
    #print(ob,hb,lb,cb)
    #build the model
    bn = gum.BayesNet(filename)
    Open0=bn.add('Open0',ob)
    High0=bn.add('High0',hb)
    Low0=bn.add('Low0',lb)
    Close0=bn.add('Close0',cb)
    Volume0=bn.add('Volume0',vb)
    Open1=bn.add('Open1',ob)
    High1=bn.add('High1',hb)
    Low1=bn.add('Low1',lb)
    Close1=bn.add('Close1',cb)
    Volume1=bn.add('Volume1',vb)
    learner = gum.BNLearner(filename+"_train.csv", bn)
    learner.addForbiddenArc('Open1','Open0')
    learner.addForbiddenArc('Open1','Close0')
    learner.addForbiddenArc('Open1','High0')
    learner.addForbiddenArc('Open1','Low0')
    learner.addForbiddenArc('Open1','Volume0')
    learner.addForbiddenArc('High1','Open0')
    learner.addForbiddenArc('High1','Close0')
    learner.addForbiddenArc('High1','High0')
    learner.addForbiddenArc('High1','Low0')
    learner.addForbiddenArc('High1','Volume0')
    learner.addForbiddenArc('Low1','Open0')
```

```
    learner.addForbiddenArc('Low1','Close0')
    learner.addForbiddenArc('Low1','High0')
    learner.addForbiddenArc('Low1','Low0')
    learner.addForbiddenArc('Low1','Volume0')
    learner.addForbiddenArc('Close1','Open0')
    learner.addForbiddenArc('Close1','Close0')
    learner.addForbiddenArc('Close1','High0')
    learner.addForbiddenArc('Close1','Low0')
    learner.addForbiddenArc('Close1','Volume0')
    learner.addForbiddenArc('Volume1','Open0')
    learner.addForbiddenArc('Volume1','Close0')
    learner.addForbiddenArc('Volume1','High0')
    learner.addForbiddenArc('Volume1','Low0')
    learner.addForbiddenArc('Volume1','Volume0')
    #learner.addMandatoryArc('Close0','Close1')
    learner.useLocalSearchWithTabuList()
    bn = learner.learnBN()
    gnb.showBN(bn)
    learner = gum.BNLearner(filename+"_train.csv", bn)
    learner.setInitialDAG(bn.dag())
    learner.useAprioriSmoothing(1)
    bn = learner.learnParameters()
    #gnb.showInference(bn,evs={})
    #do inference and calculate the accuracy
    ie = gum.LazyPropagation(bn)
    ie.makeInference()
    N=0.0
    acc=0
    with open(filename+'_test.csv','r', encoding="utf-8") as csvfile:
        reader = csv.reader(csvfile)
        for line in list(reader)[1:]:
            c,o,h,l,v,t=[line[0],line[1],line[2],line[3],line[4],line[5]]
            ie.eraseAllEvidence()
            ie.setEvidence({'Close0':c, 'Open0':o,'High0':h, 'Low0':
             ↪  l,'Volume0': v})
            ie.makeInference()
            prob=ie.posterior(Close1).tolist()
            if prob[0] < 0.6:
                N=N+1
```

```
                if t == '1':
                    acc=acc+1
    return acc,N

def genttk(filename,k):#generate the test and train set for k-order markov
    df = pd.read_csv(filename+'.csv')
    df.dropna(axis=0, how='any', inplace=True)#drop the line with NAN in case there is missing data in the file
    Date=df['Date']
    index=list(Date).index('2015-11-13')#find the index of 2015-11-13, we need to slice the list later
        #generate returns and then discretized variables.
    Open,ob=genbin(genre(df['Open']))
    High,hb=genbin(genre(df['High']))
    Low,lb=genbin(genre(df['Low']))
    Close,cb=genbin(genre(df['Close']))
    Volume,vb=genbinv(genre(df['Volume']))
    train_Open=Open[:index]
    test_Open=Open[index-1:]
    train_High=High[:index]
    test_High=High[index-1:]
    train_Low=Low[:index]
    test_Low=Low[index-1:]
    train_Close=Close[:index]
    test_Close=Close[index-1:]
    train_Volume=Volume[:index]
    test_Volume=Volume[index-1:]
    train=pd.DataFrame()#The train data
    test=pd.DataFrame()#The test data
    for i in range(k+1):#from 0 to k
        if i!=k:
            train['Close'+str(i)]=train_Close[i:-k+i]#at i
            train['Open'+str(i)]=train_Open[i:-k+i]#at i
            train['High'+str(i)]=train_High[i:-k+i]#at i
            train['Low'+str(i)]=train_Low[i:-k+i]#at i
            train['Volume'+str(i)]=train_Volume[i:-k+i]#at i
        if i==k:
            train['Close'+str(i)]=train_Close[i:]#at k
            train['Open'+str(i)]=train_Open[i:]#at k
```

```
            train['High'+str(i)]=train_High[i:]#at k
            train['Low'+str(i)]=train_Low[i:]#at k
            train['Volume'+str(i)]=train_Volume[i:]#at k
        if i!=k:
            test['Close'+str(i)]=test_Close[i:-k+i]#at i
            test['Open'+str(i)]=test_Open[i:-k+i]#at i
            test['High'+str(i)]=test_High[i:-k+i]#at i
            test['Low'+str(i)]=test_Low[i:-k+i]#at i
            test['Volume'+str(i)]=test_Volume[i:-k+i]#at i
        #####Generate the boolean var for accuracy calculation
    true=[]
    h=list(df['High'])[index+k-1:]
    c=list(df['Close'])[index+k-1:]
    for i in range(len(h)-1):
        if c[i]<h[i+1]:
            true.append(1)
        else:
            true.append(0)
    test['true']=true
    train.set_index('Close0', inplace=True)
    train.to_csv(filename+'_train'+str(k)+'.csv')
    test.set_index('Close0', inplace=True)
    test.to_csv(filename+'_test'+str(k)+'.csv')
    return ob,hb,lb,cb,vb

def kmodel(filename,k):#generate a k-order markov chain and calculate its accuracy
    ob,hb,lb,cb,vb=genttk(filename,k)
    bn = gum.BayesNet(filename)
    Open=[bn.add('Open'+str(i),ob) for i in range(k+1)]
    High=[bn.add('High'+str(i),hb) for i in range(k+1)]
    Low=[bn.add('Low'+str(i),lb) for i in range(k+1)]
    Close=[bn.add('Close'+str(i),cb) for i in range(k+1)]
    Volume=[bn.add('Volume'+str(i),vb) for i in range(k+1)]
    learner = gum.BNLearner(filename+'_train'+str(k)+'.csv', bn)
    for i in range(1,k+1):#i=future
        for j in range(i):#j=past
            learner.addForbiddenArc('Open'+str(i),'Open'+str(j))
            learner.addForbiddenArc('Open'+str(i),'Close'+str(j))
```

```
            learner.addForbiddenArc('Open'+str(i),'High'+str(j))
            learner.addForbiddenArc('Open'+str(i),'Low'+str(j))
            learner.addForbiddenArc('Open'+str(i),'Volume'+str(j))
            learner.addForbiddenArc('High'+str(i),'Open'+str(j))
            learner.addForbiddenArc('High'+str(i),'Close'+str(j))
            learner.addForbiddenArc('High'+str(i),'High'+str(j))
            learner.addForbiddenArc('High'+str(i),'Low'+str(j))
            learner.addForbiddenArc('High'+str(i),'Volume'+str(j))
            learner.addForbiddenArc('Low'+str(i),'Open'+str(j))
            learner.addForbiddenArc('Low'+str(i),'Close'+str(j))
            learner.addForbiddenArc('Low'+str(i),'High'+str(j))
            learner.addForbiddenArc('Low'+str(i),'Low'+str(j))
            learner.addForbiddenArc('Low'+str(i),'Volume'+str(j))
            learner.addForbiddenArc('Close'+str(i),'Open'+str(j))
            learner.addForbiddenArc('Close'+str(i),'Close'+str(j))
            learner.addForbiddenArc('Close'+str(i),'High'+str(j))
            learner.addForbiddenArc('Close'+str(i),'Low'+str(j))
            learner.addForbiddenArc('Close'+str(i),'Volume'+str(j))
            learner.addForbiddenArc('Volume'+str(i),'Open'+str(j))
            learner.addForbiddenArc('Volume'+str(i),'Close'+str(j))
            learner.addForbiddenArc('Volume'+str(i),'High'+str(j))
            learner.addForbiddenArc('Volume'+str(i),'Low'+str(j))
            learner.addForbiddenArc('Volume'+str(i),'Volume'+str(j))
    learner.useLocalSearchWithTabuList()
    bn = learner.learnBN()
    #gnb.showBN(bn)
    learner = gum.BNLearner(filename+'_train'+str(k)+'.csv', bn)
    learner.setInitialDAG(bn.dag())
    learner.useAprioriSmoothing(1)
    bn = learner.learnParameters()
    ie = gum.LazyPropagation(bn)
    ie.makeInference()
    N=0.0
    acc=0
    with open(filename+'_test'+str(k)+'.csv','r', encoding="utf-8") as csvfile:
        reader = csv.reader(csvfile)
        for line in list(reader)[1:]:
            t=line[-1]
```

```
            ie.eraseAllEvidence()
            for i in range(k):
                ie.setEvidence({'Close'+str(i):line[5*i],'Open'+str(i):line[5*i+1],'High'+str(i):line[5*i+2],'Low'+str(i): line[5*i+3],'Volume'+str(i): line[5*i+4]})
            ie.makeInference()
            prob=ie.posterior(Close[-1]).tolist()
            if prob[0] < 0.498:
                N=N+1
                if t == '1':
                    acc=acc+1
    #print(acc,N)
    return acc,N

def evaluate_k(k):
    filelist=['IDU','IHF','IYC','IYE','IYF','IYG','IYH','IYJ'
    ,'IYK','IYM','IYR','IYT','IYW','IYZ']
    acc_c=[]
    N_c=[]
    for file in filelist:
        #rint(file+":")
        a,n=kmodel(file,k)
        acc_c.append(a)
        N_c.append(n)
    #print(accuracy)
    ev=sum(acc_c)/sum(N_c)
    '''
    result=pd.DataFrame()
    result['Sector']=filelist
    result['accuracy']=acc_c
    result['N']=N_c
    result.to_csv('result.csv')
    #rint(result,ev)
    '''
    return ev

def main():
    ev=[]
    for k in range(1,20):
```

```python
        ev.append(evaluate_k(k))
    plt.scatter(list(range(1,20)),ev)

if __name__=='__main__':
    main()
```