A3Q6

March 16, 2020

1 CMPT 423/820: Assignment 3 Question 6

- Seyedeh Mina Mousavifar
- sem311

In this Notebook, you'll use Python to explore Structure Learning in Bayesian Networks. The basic approach will be: 1. Load a categorical dataset (`diamond.csv`) 3. Create 2 (related) Bayesian network models, fitting their parameters to the data. 4. Compare the 2 structures, using log-likelihood.

The data has 5 columns, with the first column being an index. The first row gives the column names, A, B, C, D. There are too many rows and columns to calculated this data by hand, so we will need to use computational tools.

```
In [82]: import pandas as pd

         dataframe = pd.read_csv('diamond.csv', index_col=0)
```

1.1 Two Bayesian network structures

We will look at two Bayesian network structures out of the very many that are possible for this dataset. 2. Model M_1.

$$P_1(ABCD) = P(A)P(B|A)P(C|A)P(D|BC)$$

2. Model M_2.

$$P_2(ABCD) = P(A)P(B|A)P(C|A)P(D|A)$$

You will notice that the JPD is factorized, and that there are no graphs given. You can reconstruct the graph by reading the families.

1.1.1 Step 1. Calculate the Conditional Probability tables for both models.

In particular: * $P(A)$. (used in both models) * $P(B|A)$. (used in both models) * $P(C|A)$. (used in both models) * $P(D|BC)$. (used only in model M_1) * $P(D|A)$. (used only in model M_2)

Calculate and display these tables neatly. I recommend a Pandas dataframe for each table. It will make step 2 easier!

P(A)

```
In [83]: # count number of 1s and 0s and divide it by total number of records
         P_A = dataframe.groupby(['A']).size().reset_index(name='P(A)')
         total = dataframe.shape[0]
         P_A['P(A)'] = P_A['P(A)']/total

         P_A
```

```
Out[83]:    A   P(A)
         0  0  0.808
         1  1  0.192
```

$P(B|A)$

$$P(B|A) = \frac{P(A,B)}{P(A)}$$

```
In [84]: # count number of 1s and 0s for (A,B) and divide it by total number of records
         P_BandA = dataframe.groupby(['A','B']).size().reset_index(name='P(A,B)')
         P_BandA['P(A,B)'] = P_BandA['P(A,B)']/total

         # P(A,B)/P(A)
         P_BgivenA = pd.merge(P_BandA, P_A, how='inner', on=['A'])
         P_BgivenA['P(B|A)'] = P_BgivenA['P(A,B)']/P_BgivenA['P(A)']
         # remove unneccesary columns
         P_BgivenA = P_BgivenA.drop(['P(A,B)','P(A)'],1)
         P_BgivenA
```

```
Out[84]:    A  B    P(B|A)
         0  0  0  0.830446
         1  0  1  0.169554
         2  1  0  0.203125
         3  1  1  0.796875
```

$P(C|A)$

$$P(C|A) = \frac{P(A,C)}{P(A)}$$

```
In [85]: # count number of 1s and 0s for (A,C) and divide it by total number of records
         P_CandA = dataframe.groupby(['A','C']).size().reset_index(name='P(A,C)')
         P_CandA['P(A,C)'] = P_CandA['P(A,C)']/total

         # P(A,C)/P(A)
         P_CgivenA = pd.merge(P_CandA, P_A, how='inner', on=['A'])
         P_CgivenA['P(C|A)'] = P_CgivenA['P(A,C)']/P_CgivenA['P(A)']
         # remove unneccesary columns
         P_CgivenA = P_CgivenA.drop(['P(A,C)','P(A)'],1)
         P_CgivenA
```

```
Out[85]:    A  C    P(C|A)
         0  0  0  0.935644
         1  0  1  0.064356
         2  1  0  0.598958
         3  1  1  0.401042
```

$P(D|B,C)$

$$P(D|B,C) = \frac{P(B,C,D)}{P(B,C)}$$

```
In [86]: # count number of 1s and 0s for (B,C,D) and divide it by total number of records
         P_BandCandD = dataframe.groupby(['B','C','D']).size().reset_index(name='P(B,C,D)')
         P_BandCandD['P(B,C,D)'] = P_BandCandD['P(B,C,D)']/total

         # P(B,C)
         P_BandC = dataframe.groupby(['B','C']).size().reset_index(name='P(B,C)')
         P_BandC['P(B,C)'] = P_BandC['P(B,C)']/total

         # P(B,C,D)/P(B,C)
         P_DgivenBC = pd.merge(P_BandCandD, P_BandC, how='inner', on=['B','C'])
         P_DgivenBC['P(D|B,C)'] = P_DgivenBC['P(B,C,D)']/P_DgivenBC['P(B,C)']
         # remove unneccesary columns
         P_DgivenBC = P_DgivenBC.drop(['P(B,C,D)','P(B,C)'],1)
         P_DgivenBC
```

```
Out[86]:    B  C  D  P(D|B,C)
         0  0  0  0  0.949464
         1  0  0  1  0.050536
         2  0  1  0  0.175439
         3  0  1  1  0.824561
         4  1  0  0  0.307339
         5  1  0  1  0.692661
         6  1  1  0  0.208333
         7  1  1  1  0.791667
```

$P(D|A)$

$$P(D|A) = \frac{P(A,D)}{P(A)}$$

```
In [87]: # count number of 1s and 0s for (A,D) and divide it by total number of records
         P_DandA = dataframe.groupby(['A','D']).size().reset_index(name='P(A,D)')
         P_DandA['P(A,D)'] = P_DandA['P(A,D)']/total

         # P(A,D)/P(A)
         P_DgivenA = pd.merge(P_DandA, P_A, how='inner', on=['A'])
         P_DgivenA['P(D|A)'] = P_DgivenA['P(A,D)']/P_DgivenA['P(A)']
         # remove unneccesary columns
         P_DgivenA = P_DgivenA.drop(['P(A,D)','P(A)'],1)
         P_DgivenA
```

```
Out[87]:    A  D    P(D|A)
         0  0  0  0.793317
         1  0  1  0.206683
         2  1  0  0.369792
         3  1  1  0.630208
```

Grading: 5 marks Your notebook calculates the 5 tables correctly, and presents them neatly.

1.1.2 Step 2. Compare the two models using the likelihood of the data

Let **X** represent the dataset. We want to compare the following two quantities: * $P(\mathbf{X}|M_1)$ that is, the likelihood of the data using M_1. * $P(\mathbf{X}|M_2)$ that is, the likelihood of the data using the other model.

Let a, b, c, d be the values for the variables on one of the rows. In the first model:

$$P(a, b, c, d|M_1) = P(a)P(b|a)P(c|a)P(d|bc)$$

In other words, we are using the tables for M_1 in this factorization. TO finish this calculation off, we have to look up one number from each table, and mutiply them all together. These are the tables we calculated in Step 1.

This is the likelihood of a single row of the data. To calculate the likelihood of the whole data set, we assume each row is independent:

$$P(\mathbf{X}|M_1) = \prod_i P_1(a_i, b_i, c_i, d_i|M_1).$$

In other words we multiply the likelihoods of all the rows together to get the likelihood of the dataset using Model M_1.

Keep in mind this calculation is for the first model, M_1; a slightly different calculation is needed for the second model M_2, because the family for D is different.

When you have calculated the likelihood for the data in **both models**, we can say that the preferred model is the one where the likelihood is highest.

Here's the task: Compare the likelihood of the data in both models, and decide which one fits the data better.

```
In [88]: # creating full data for each combination of a,b,c,d
         info = pd.merge(P_BgivenA, P_A, how='inner', on=['A'])
         info = pd.merge(info, P_CgivenA, how='inner', on=['A'])
         info = pd.merge(info, P_DgivenA, how='inner', on=['A'])
         info = pd.merge(info, P_DgivenBC, how='inner', on=['B','C','D'])

         info = info[['A', 'B', 'C', 'D',
                      'P(A)', 'P(B|A)', 'P(C|A)', 'P(D|A)', 'P(D|B,C)']]
```

M1

$$P(a, b, c, d|M_1) = P(a)P(b|a)P(c|a)P(d|bc)$$

```
In [89]: import math
         import numpy as np
```

```
        # P M1
        info['P(a,b,c,d|M_1)'] = info['P(A)']*info['P(B|A)']*info['P(C|A)']*info['P(D|B,C)']

        # log for prevention of underflow for small numbers in likelihood
        info['M1-log'] = np.log2(info['P(a,b,c,d|M_1)'])
```

M2

$$P(a, b, c, d|M_2) = P(a)P(b|a)P(c|a)P(d|a)$$

```
In [90]: # P M2
         info['P(a,b,c,d|M_2)'] = info['P(A)']*info['P(B|A)']*info['P(C|A)']*info['P(D|A)']

         # log for prevention of underflow for small numbers in likelihood
         info['M2-log'] = np.log2(info['P(a,b,c,d|M_2)'])
```

Likelihood

$$P(\mathbf{X}|M) = \prod_i P(a_i, b_i, c_i, d_i|M)$$
$$P(\mathbf{X}|M) = \sum_i P(a_i, b_i, c_i, d_i|M)$$

```
In [92]: # object for saving data
         result = {'M1':{}, 'M2':{}}

         # M1
         result['M1']['likelihood'] = np.prod(info['P(a,b,c,d|M_1)'])
         result['M1']['log-likelihood'] = info['M1-log'].sum()

         # M2
         result['M2']['likelihood'] = np.prod(info['P(a,b,c,d|M_2)'])
         result['M2']['log-likelihood'] = info['M2-log'].sum()

In [93]: # printing result in tabular format
         print('\033[1m' + 'Likelihood' + '\033[0m')
         print('{:<25} {:<25} {:<25}'.format('Model',
                                             'likelihood',
                                             'log likelihood'))

         for model, item in result.items():
             print("{:<25} {:<25} {:<25}".format(model,
                                                 item['likelihood'],
                                                 item['log-likelihood']))
```

```
Likelihood
Model                     likelihood                log likelihood
M1                        2.353342834066061e-28     -91.77927514972284
M2                        8.447394304170701e-27     -86.61355216780413
```

Grading: 4 marks Your calculations are correct.

1.1.3 Conclusion:

Which of the two model is the better fit for the data?

> The model with a higher likelihood better represents the data. M2 has higher likelihood and log-likelihood than M1, so it is the better fit for the data.

Grading: 1 mark Your conclusion is appropriate.