CMPT 423/820: Assignment 3 Question 6

- Solutions and Marking Guide

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`)
2. Create 2 (related) Bayesian network models, fitting their parameters to the data.
3. 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 [1]: import pandas as pd
        import numpy as np
        import math as math

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

Two Bayesian network structures

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

1. 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.

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!

In [2]:
```
# Fitting CPTs from data using Pandas

# for add-one smoothing
smooth1D = np.array([1]*2)
smooth2D = np.array([1]*4)
smooth3D = np.array([1]*8)



# P(A)
# Adding smooth1D implements add-one smoothing, also known as Beta(1,1
) prior
count_A = smooth1D + dataframe.groupby(['A'])['A'].count()
# a CPT can be construacted from a table of counts, if it is normalize
d
P_A = count_A/count_A.sum()



# P(B|A)
count_BA = smooth2D + dataframe.groupby(['B','A'])['B'].count()
# we sum along the 'B' axis to normalize
denom_BA = count_BA.groupby(['A']).sum()
P_B_A = count_BA/denom_BA



# P(C|A)
count_CA = smooth2D + dataframe.groupby(['C','A'])['C'].count()
# we sum along the 'C' axis to normalize
denom_CA = count_CA.groupby(['A']).sum()
P_C_A = count_CA/denom_CA



# P(D|BC) -- only in Model 1
count_DBC = smooth3D + dataframe.groupby(['D','B','C'])['D'].count()
```

```
# we sum along the 'D' axis to normalize
denom_BC = count_DBC.groupby(['B','C']).sum()
P_D_BC = count_DBC/denom_BC


# P(D|A) -- only in Model 2
count_DA = smooth2D + dataframe.groupby(['D','A'])['D'].count()
# we sum along the 'D' axis to normalize
denom_DA = count_DA.groupby(['A']).sum()
P_D_A = count_DA/denom_DA


###################################################
print('CPTs for Model One and Two')
print('P(A)')
print(P_A)
print()

print('P(B|A)')
print(P_B_A)
print()

print('P(C|A)')
print(P_C_A)
print()

print('CPTs for Model One only')

print('P(D|BC)')
print(P_D_BC)
print()

print('CPTs for Model Two only')

print('P(D|A)')
print(P_D_A)
print()
```

```
CPTs for Model One and Two
P(A)
A
0    0.807385
1    0.192615
Name: A, dtype: float64

P(B|A)
B  A
0  0    0.829630
   1    0.206186
1  0    0.170370
   1    0.793814
Name: B, dtype: float64

P(C|A)
C  A
0  0    0.934568
   1    0.597938
1  0    0.065432
   1    0.402062
Name: C, dtype: float64

CPTs for Model One only
P(D|BC)
B  C  D
0  0  0    0.948092
      1    0.051908
   1  0    0.186441
      1    0.813559
1  0  0    0.309091
      1    0.690909
   1  0    0.216216
      1    0.783784
Name: D, dtype: float64

CPTs for Model Two only
P(D|A)
D  A
0  0    0.792593
   1    0.371134
1  0    0.207407
   1    0.628866
Name: D, dtype: float64
```

Grading: 5 marks

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

Marking

Counting is not the problem, but keeping the counts straight and normalized is a bit tricky. I used Pandas dataframes, which I recommended, but figuring out how to do all the right summing is an exercise in skimming the Pandas documentation, and it's okay if it's done using Numpy or something else.

The numbers themselves are not really the point either. I used "add one smoothing" (also known as "Beta(1,1) priors") above, but that's not essential. I just wanted to show it could be done. It makes a small difference in the values, which is exactly what you'd expect.

- 3 marks: Correctness
  - The CPTs have to have the right number of values, e.g., P(A) has 2, P(B|A) has 4, etc
  - The CPTs need to sum to 1 in the right ways.
  - There have to be 5 CPTs in total.
- 2 marks: Neatness
  - The tables have at least minimal formatting; they can't simply be a list displayed to the console.

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

Let $\mathbf{X}$ 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 [3]:
```
# log likelihoods for the two models
# because working in probability space will usually cause underflow

loglik1 = 0
loglik2 = 0

# work through the data set, row by row
for row in dataframe.itertuples():
    # use the values on the row to index into the CPTs.
    pa = P_A[row.A]

    # I guess Pandas was not really meant for this
    # if you used a dictionary or a list, that's okay
    pb = P_B_A.xs(row.B, level='B')[row.A]
    pc = P_C_A.xs(row.C, level='C')[row.A]
    pd1 = P_D_BC.xs(row.D, level='D').xs(row.B, level='B')[row.C]
    pd2 = P_D_A.xs(row.D, level='D')[row.A]

    # now multiply probabilities (or add log probabilities)
    common = pa * pb * pc
    row_prob1 = common * pd1
    row_prob2 = common * pd2

    # two accumulators for the two models
    loglik1 += math.log(row_prob1, 10)
    loglik2 += math.log(row_prob2, 10)

print('Log-likelihoods:')
print('Model {}: {:.3f}'.format(1, loglik1))
print('Model {}: {:.3f}'.format(2, loglik2))
print('Difference: {:.3f}'.format(loglik1 - loglik2))
```

```
Log-likelihoods:
Model 1: -696.859
Model 2: -787.921
Difference: 91.062
```

In [4]:
```
# Since parts of the two models are the same
# we only need to keep track of what's different.

# loglikelihood differences for the two models
diff1 = 0
diff2 = 0

# work through the data set, row by row
for row in dataframe.itertuples():
    # only the model for Variable D is different
    pd1 = P_D_BC.xs(row.D, level='D').xs(row.B, level='B')[row.C]
    pd2 = P_D_A.xs(row.D, level='D')[row.A]

    # just accumulate the differences
    diff1 += math.log(pd1, 10)
    diff2 += math.log(pd2, 10)

print('Log-likelihood differences:')
print('Model {}: {:.3f}'.format(1, diff1))
print('Model {}: {:.3f}'.format(2, diff2))
print('Difference: {:.3f}'.format(diff1 - diff2))
```

```
Log-likelihood differences:
Model 1: -142.675
Model 2: -233.737
Difference: 91.062
```

Grading: 4 marks

Your calculations are correct.

Marking:

A correct solution means that the method was largely correct, if not exactly as above.
I showed in the second version that the only part that needs to be calculated is the factor involving D, since the two models were exactly the same for the other variables.

- 4 marks: Correctness. To be correct, the script has to:
  1. Loop through each row in the dataset
  2. Calculate the probability of the values on the row, using both models.
  3. Calculate the log-likelihood of the whole dataset with two models, because without the logs, underflow will occur.

Deduct a mark for each item not done.

Conclusion:

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

According to the log-likelihoods of the two models, Model 1 is better, because the log-likelihood is higher.

This is not too surprising given the name of the file. Model 1 is a diamond shape, whereas Model 2 has the structure of Naive Bayes. I generated synthetically, starting from a Bayesian network with the same diamond shape, with some artificially created CPTs, and randomly sampled values from the model. It's a good thing that Model 1 is the better fit, because that's what we started with. Had I started with Model 1, and generated data from it, it would have been the better model according to this analysis.

I'm a little disappointed in the speed of the calculations. There's probably a faster way to do these calculations with Pandas. Also, most of my effort to complete this question was figuring out which Pandas methods to call.

Grading: 1 mark

Your conclusion is appropriate.

Marking

The correct conclusion is that Model 1 is the better one. Do not give the mark if this conclusion is not the result of the effort. There's a lot of generosus marking here, but this is where you have to get the right answer.