

P1 Discrete Variables

Will Koehrsen wjk68

March 5, 2018

Contents

1 Discrete Variables Bayesian Inference Using MCMC

1.0.1 Bayesian Belief Network with Discrete Variables

For a simple example, I created a belief network representing the probability of getting a raise based on several variables. All of the variables in the network are discrete and the entire joint probability can be specified using priors and conditional probability tables.

The entire network is shown below:

The top level variables show the prior probabilities in the node. The other probabilities we need to specify are the conditionals for the mid-level nodes (Boss's Mood and Profits) and the final node (Raise). All of these variables depend on two parent variables, with the mid level nodes having 6 entries in the conditional probability table (representing all possible states of the parents) and the Raise node having four entries in the conditional probability table. The tables are presented below:

Conditional Probability of Boss's Mood

Weather	Personal Performance	P(Mood = Positive)
0	0	0.15
0	1	0.55
1	0	0.45
1	1	0.85
2	0	0.35
2	1	0.75

Conditional Probability of Company Performance

Company Revenue	Personal Performance	P(Company Performance = Profit)
0	0	0.05
0	1	0.10
1	0	0.75

Company Revenue	Personal Performance	P(Company Performance = Profit)
1	1	0.95
2	0	0.20
2	1	0.40

Conditional Probability of Raise

Boss's Mood	Company Performance	P(Raise = True)
0	0	0.05
0	1	0.65
1	0	0.40
1	1	0.90

With the prior probabilities and the conditional probabilities, we can compute the entire joint probability of the network. We can also use Markov Chain Monte Carlo to approximate the posterior probabilities of any variables in the network with or without observations.

1.1 A. Discrete Bayesian Network in PyMC3

The first approach to solve this network uses a Markov Chain Monte Carlo algorithm to find approximate probabilities. MCMC works by sampling from the posterior distribution to find an approximation of the posterior and the accuracy increases as the number of samples increases. PyMC3 implements Bayesian Reasoning by constructing a graph of the network and then evaluating the graph during the sampling process. In this problem we have only two kinds of variables: binary (taking on two values), and discrete (in this case taking on 3 values). We can build up the graph and then query it to find the approximate probabilities without and then with evidence (observations).

```
In [20]: import numpy as np
import pandas as pd

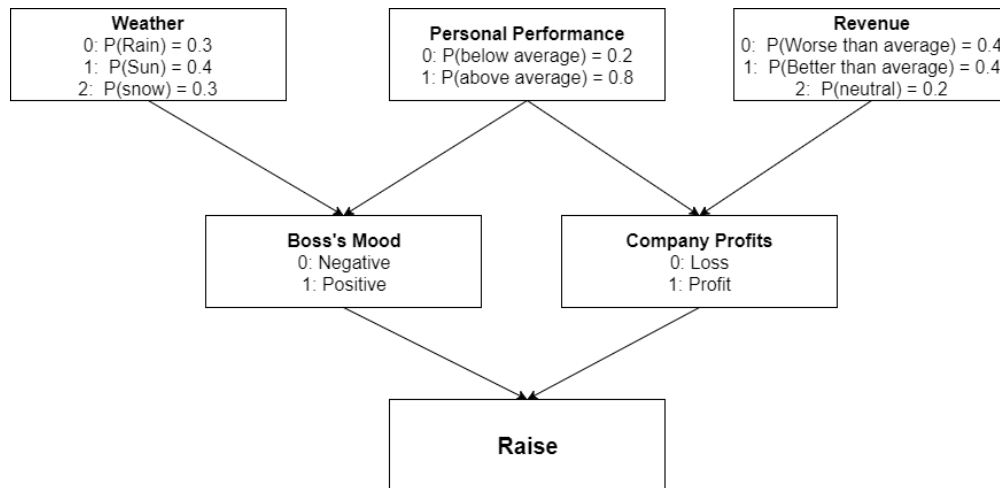
import pymc3 as pm

import matplotlib.pyplot as plt
%matplotlib inline

import theano.tensor as tt
import matplotlib

import seaborn as sns

from IPython.core.pylabtools import figsize
```



image

1.1.1 Construct the Model Graph

First, we make the network in pymc3. We specify the prior probabilities and the conditional probability tables. In pymc3, the conditional probabilities are represented using switch statements.

```
In [3]: N_SAMPLES = 1000
        N_CHAINS = 2
```

```
with pm.Model() as raise_model:
```

```
# Weather is a categorical variable with 3 states
weather = pm.Categorical('weather', np.array([0.3, 0.4, 0.3]))
```

```
# Personal Performance is a Bernoulli Variable with 2 states
pp = pm.Bernoulli('pp', 0.8)
```

```
# Company Revenue is a categorical variable with 3 states
revenue = pm.Categorical('revenue', np.array([0.4, 0.4, 0.2]))
```

```
# Boss's Mood Conditional Probability Definition
mood = pm.Bernoulli('mood', tt.switch(tt.eq(weather, 0), tt.switch(tt.eq(pp,
                                                                    tt.switch(tt.eq(weather, 1), tt.switch(
                                                                    tt.switch(
```

```
# Company Performance Conditional Probability Definition
cp = pm.Bernoulli('cp', tt.switch(tt.eq(revenue, 0), tt.switch(tt.eq(pp, 0),
                                                                tt.switch(tt.eq(revenue, 1), tt.switch(tt.
                                                                tt.switch(tt.eq(pp,
```

```

# Raise is a Bernoulli Variable with 2 states
raise_ = pm.Bernoulli('raise_', tt.switch(tt.eq(mood, 0), tt.switch(tt.eq(cp
                                     tt.switch(tt.eq(cp

```

1.1.2 Sample from the Posterior Using the Gibbs Metropolis Algorithm

After the graph is built, we can draw samples from the posterior distribution using a Markov Chain Monte Carlo approach called Gibbs Metropolis. At each step, the algorithm draws a new sample from the posterior based on the provided probabilities. This is a Markov Chain because the next state depends only a bounded subset of past states. MCMC can be compared to a random walk: from a given state, there are a limited number of random steps available to move. In this case, we are sampling with no observations, but if we have observations, we can compare the samples to the observations using rejection sampling. If the sample does not agree with the evidence, it is rejected, otherwise, the sample is accepted and becomes the current state. The model converges as the number of steps increases.

```

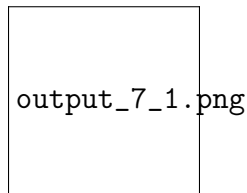
In [4]: with raise_model:
        # Sample and plot traces
        raise_trace = pm.sample(draws=N_SAMPLES, chains=N_CHAINS)
        pm.traceplot(raise_trace)

```

```

Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>CategoricalGibbsMetropolis: [revenue, weather]
>BinaryGibbsMetropolis: [pp, mood, cp, raise_]

```



1.2 Evaluate the Trace to Find Probabilities

Using the sample trace, we can find the probability of different nodes in the network. In this case, we are taking the mean of the trace variables to represent the most likely value for a parameter.

```

In [5]: def query_model(trace, obs={}, compare=False):
        if 'weather' in obs.keys():

```

```

f_rain = obs['weather'].get('rain', 0)
f_sun = obs['weather'].get('sun', 0)
f_snow = obs['weather'].get('snow', 0)

else:
    f_rain = np.mean(trace['weather'] < 0.5)
    f_sun = np.mean((trace['weather'] >= 0.5) & (trace['weather'] <= 1.5))
    f_snow = np.mean(trace['weather'] > 1.5)

print('Approximate Probabilities from MCMC:')
print("""\nWeather: {:.2f}, {:.2f}, {:.2f}""".format(
    f_rain, f_sun, f_snow))

if 'pp' in obs.keys():
    f_pp_above = obs['pp'].get('pp')

else:
    f_pp_above = np.mean(trace['pp'] >= 0.5)

f_pp_below = 1 - f_pp_above
print("""Personal Performance: {:.2f}, {:.2f}""".format(
    f_pp_below, f_pp_above))

if 'revenue' in obs.keys():
    f_revenue_below = obs['revenue'].get('below', 0)
    f_revenue_above = obs['revenue'].get('above', 0)
    f_revenue_neutral = obs['revenue'].get('neutral', 0)

else:
    f_revenue_below = np.mean(trace['revenue'] < 0.5)
    f_revenue_above = np.mean((trace['revenue'] >= 0.5) & (trace['revenue'] < 1.5))
    f_revenue_neutral = np.mean(trace['revenue'] > 1.5)
print("""Revenue: {:.2f}, {:.2f}, {:.2f}""".format(
    f_revenue_below, f_revenue_above, f_revenue_neutral))

if 'mood' in obs.keys():
    f_mood_positive = obs['mood'].get('mood')

else:
    f_mood_positive = np.mean(trace['mood'] >= 0.5)

f_mood_negative = 1 - f_mood_positive
print("""Boss's mood: {:.2f}, {:.2f}""".format(
    f_mood_negative, f_mood_positive))

if 'cp' in obs.keys():

```

```

        f_profits = obs['cp'].get('cp')
    else:
        f_profits = np.mean(trace['cp'] >= 0.5)

    f_loss = 1 - f_profits

    print("""Profits: {:.2f}, {:.2f}""".format(
        f_loss, f_profits))

    if 'raise' in obs.keys():
        f_raise = obs['raise'].get('raise')

    else:
        f_raise = np.mean(trace['raise_'] >= 0.5)

    print("""\nProbability of a raise: {:.4f}""".format(f_raise))

    if compare:
        return [f_rain, f_sun, f_snow, f_pp_above, f_revenue_below,
                f_revenue_above, f_revenue_neutral, f_mood_positive,
                f_profits, f_raise]

```

1.2.1 Approximate Probabilities with No Evidence

The first query we will make finds the approximate probability of a raise with no evidence.

```
In [6]: query_model(raise_trace)
```

Approximate Probabilities from MCMC:

```

Weather: 0.30, 0.40, 0.30.
Personal Performance: 0.21, 0.79
Revenue: 0.39, 0.40, 0.21.
Boss's mood: 0.36, 0.64.
Profits: 0.54, 0.46.

```

```
Probability of a raise: 0.5140.
```

We can see based only on the priors and conditional probabilities, with no observations, there is a slightly over 50% chance of a raise.

1.3 Model Accounting for Observations

If we do have observations, such as the weather, personal performance, or revenue, we can feed those into the model to update our estimate. This is the essence of Bayesian Infer-

ence: we have prior beliefs, that we subsequently update as we gather more information. Bayes Rule is expressed below:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} = \frac{\text{likelihood} * \text{prior}}{\text{normalization}}$$

```
In [7]: def model_with_evidence(weather_obs=None, pp_obs=None, revenue_obs=None,
                                mood_obs=None, cp_obs=None, raise_obs=None):

    with pm.Model() as raise_model:

        # Weather is a categorical variable with 3 states
        weather = pm.Categorical('weather', np.array([0.3, 0.4, 0.3]),
                                observed = weather_obs)

        # Personal Performance is a Bernoulli Variable with 2 states
        pp = pm.Bernoulli('pp', 0.8, observed = pp_obs)

        # Company Revenue is a categorical variable with 3 states
        revenue = pm.Categorical('revenue', np.array([0.4, 0.4, 0.2]),
                                observed = revenue_obs)

        # Boss's Mood Conditional Probability Definition
        mood = pm.Bernoulli('mood', tt.switch(tt.eq(weather, 0), tt.switch(
                                                    tt.switch(tt.eq(weather, 1)

                                observed = mood_obs)

        # Company Performance Conditional Probability Definition
        cp = pm.Bernoulli('cp', tt.switch(tt.eq(revenue, 0), tt.switch(tt
                                                    tt.switch(tt.eq(revenue, 1), tt
                                                    tt.switch

                                observed = cp_obs)

        # Raise is a Bernoulli Variable with 2 states
        raise_ = pm.Bernoulli('raise_', tt.switch(tt.eq(mood, 0), tt.swit
                                                    tt.swit

                                observed = raise_obs)

        # Sample and plot traces
        raise_trace = pm.sample(draws=N_SAMPLES, chains=N_CHAINS)

    return (raise_trace)
```

1.3.1 Approximate Probabilities with Evidence

Now, we can query the same model but passing in observations of the variables. We will start off with the single observation that company revenue was above average.

```
In [8]: above_revenue_trace = model_with_evidence(revenue_obs=1)
```

```
Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>CategoricalGibbsMetropolis: [weather]
>BinaryGibbsMetropolis: [pp, mood, cp, raise_]
```

```
In [9]: query_model(above_revenue_trace, obs={'revenue': {'above':1}})
```

Approximate Probabilities from MCMC:

```
Weather: 0.30, 0.40, 0.29.
Personal Performance: 0.21, 0.79
Revenue: 0.00, 1.00, 0.00.
Boss's mood: 0.36, 0.64.
Profits: 0.10, 0.90.
```

Probability of a raise: 0.7475.

Based on the MCMC process, we can see that with this piece of information, the posterior probability of a raise is greater than that without any observations. Our estimate has been updated to be more accurate with the evidence. Next, we can incorporate even more observations to find the posterior probability for a raise. The good part about the model is it returns the posterior probabilities of all variables in the model, and not only the final raise. Therefore, we can see how observations change the approximate probabilities of any variable in the network.

```
In [10]: good_mood_above_performance = model_with_evidence(mood_obs=1, pp_obs=1)
```

```
Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>CategoricalGibbsMetropolis: [revenue, weather]
>BinaryGibbsMetropolis: [cp, raise_]
The number of effective samples is smaller than 25% for some parameters.
```

```
In [11]: query_model(good_mood_above_performance, obs = {'mood': {'mood': 1}, 'pp': {'pp': 1}})
```


Approximate Probabilities from MCMC:

Weather: 0.23, 0.46, 0.31.
Personal Performance: 0.00, 1.00
Revenue: 0.39, 0.41, 0.20.
Boss's mood: 0.00, 1.00.
Profits: 0.50, 0.50.

Probability of a raise: 0.6655.

2 B. Exact Evaluation of the Joint Probability

The other method to evaluate the network is by explicitly using the priors and conditional probability tables. This will allow us to find the exact posterior probability. The reason we can do exact evaluation for this problem is because we are only using binary and discrete variables. With continuous variables, evaluating posteriors from the joint is nearly impossible by hand and computationally intractable for large network.

We will use Bayes' Rule to find the probability of a raise both without and then with observations. After we instantiate the model to evaluate the joint exactly, we can compare the results to the approximate values from MCMC.

```
In [12]: def joint_probability(weather_obs = None, pp_obs=None, revenue_obs=None,
                                mood_obs = None, cp_obs = None, raise_obs = None,
                                compare=False):

    if weather_obs==0:
        p_w_0 = 1
        p_w_1 = 0
        p_w_2 = 0
    elif weather_obs==1:
        p_w_0 = 0
        p_w_1 = 1
        p_w_2 = 0
    elif weather_obs==2:
        p_w_0 = 0
        p_w_1 = 0
        p_w_2 = 1

    else:
        p_w_0 = 0.3
        p_w_1 = 0.4
        p_w_2 = 0.3
```

```

print('\nExact Probabilities from the joint:')
print("""\nWeather {:.2f}, {:.2f}, {:.2f}""".format(
    p_w_0, p_w_1, p_w_2))

if pp_obs==0:
    p_pp_0=1
    p_pp_1=0
elif pp_obs==1:
    p_pp_0=0
    p_pp_1=1
else:
    p_pp_0 = 0.2
    p_pp_1 = 0.8

print("""Personal Performance: {:.2f}, {:.2f}""".format(
    p_pp_0, p_pp_1))

if revenue_obs==0:
    p_r_0 = 1
    p_r_1 = 0
    p_r_2 = 0
elif revenue_obs==1:
    p_r_0 = 0
    p_r_1 = 1
    p_r_2 = 0
elif revenue_obs==2:
    p_r_0 = 0
    p_r_1 = 0
    p_r_2 = 1
else:
    p_r_0 = 0.4
    p_r_1 = 0.4
    p_r_2 = 0.2

print("""Revenue: {:.2f}, {:.2f}, {:.2f}""".format(
    p_r_0, p_r_1, p_r_2))

if mood_obs==0:
    p_m_0=1
    p_m_1=0
elif mood_obs==1:
    p_m_0=0
    p_m_1=1

```

```

else:
    p_m_1 = (0.15 * p_w_0 * p_pp_0 +
             0.55 * p_w_0 * p_pp_1 +
             0.45 * p_w_1 * p_pp_0 +
             0.85 * p_w_1 * p_pp_1 +
             0.35 * p_w_2 * p_pp_0 +
             0.75 * p_w_2 * p_pp_1)
    p_m_0 = 1 - p_m_1

print("""Boss's mood: {:.2f}, {:.2f}""".format(
    p_m_0, p_m_1))

if cp_obs==0:
    p_cp_0=1
    p_cp_1=0
elif cp_obs==1:
    p_cp_0=0
    p_cp_1=1

else:
    p_cp_1 = (0.05 * p_r_0 * p_pp_0 +
             0.10 * p_r_0 * p_pp_1 +
             0.75 * p_r_1 * p_pp_0 +
             0.95 * p_r_1 * p_pp_1 +
             0.20 * p_r_2 * p_pp_0 +
             0.40 * p_r_2 * p_pp_1)
    p_cp_0 = 1 - p_cp_1

print("""Profits: {:.2f}, {:.2f}""".format(
    p_cp_0, p_cp_1))

if raise_obs==0:
    p_r=0
elif raise_obs==1:
    p_r=1

else:
    p_r = (0.05 * p_m_0 * p_cp_0 +
          0.65 * p_m_0 * p_cp_1 +
          0.40 * p_m_1 * p_cp_0 +
          0.90 * p_m_1 * p_cp_1)

print('\nProbabability of a raise: {:.4f}.'.format(p_r))

```

```

    if compare:
        return [p_w_0, p_w_1, p_w_2, p_pp_1, p_r_0, p_r_1, p_r_2,
                p_m_1, p_cp_1, p_r]

```

2.0.1 Exact Posterior Probability with No Evidence

```
In [13]: joint_probability()
```

Exact Probabilities from the joint:

```

Weather 0.30, 0.40, 0.30.
Personal Performance: 0.20, 0.80.
Revenue: 0.40, 0.40, 0.20.
Boss's mood: 0.35, 0.65.
Profits: 0.53, 0.47.

```

Probabability of a raise: 0.5300.

2.0.2 Exact Posterior Probability with Evidence

```
In [14]: joint_probability(pp_obs=1)
```

Exact Probabilities from the joint:

```

Weather 0.30, 0.40, 0.30.
Personal Performance: 0.00, 1.00.
Revenue: 0.40, 0.40, 0.20.
Boss's mood: 0.27, 0.73.
Profits: 0.50, 0.50.

```

Probabability of a raise: 0.5690.

```
In [15]: joint_probability(pp_obs=1, revenue_obs=1)
```

Exact Probabilities from the joint:

```

Weather 0.30, 0.40, 0.30.
Personal Performance: 0.00, 1.00.
Revenue: 0.00, 1.00, 0.00.

```

Boss's mood: 0.27, 0.73.
Profits: 0.05, 0.95.

Probabability of a raise: 0.8062.

```
In [16]: joint_probability(weather_obs=0, pp_obs=1, revenue_obs=1)
```

Exact Probabilities from the joint:

Weather 1.00, 0.00, 0.00.
Personal Performance: 0.00, 1.00.
Revenue: 0.00, 1.00, 0.00.
Boss's mood: 0.45, 0.55.
Profits: 0.05, 0.95.

Probabability of a raise: 0.7602.

```
In [17]: joint_probability(revenue_obs=1)
```

Exact Probabilities from the joint:

Weather 0.30, 0.40, 0.30.
Personal Performance: 0.20, 0.80.
Revenue: 0.00, 1.00, 0.00.
Boss's mood: 0.35, 0.65.
Profits: 0.09, 0.91.

Probabability of a raise: 0.7644.

As in the approximate MCMC sampling, we can see that evidence changes the posterior probabilities. In this model however, we are not using the evidence to update all of the probabilities, only the probability of a raise. We can find the posterior for any variable in the network, but it is tedious by hand.

3 Comparison of Approximate and Exact

The final step is to compare the approximate MCMC method and exact evaluation from the joint probability network. Ideally these should be in close agreement.

```

In [18]: def compare(weather_obs = None, pp_obs=None, revenue_obs=None,
                    mood_obs = None, cp_obs = None, raise_obs = None):

    approximate_trace = model_with_evidence(weather_obs, pp_obs, revenue_obs,
                                            mood_obs, cp_obs, raise_obs)

    obs_dict = {}

    if weather_obs==0:
        obs_dict['weather'] = {'rain': 1}
    elif weather_obs==1:
        obs_dict['weather'] = {'sun': 1}
    elif weather_obs==2:
        obs_dict['weather'] = {'snow': 1}

    if pp_obs==0:
        obs_dict['pp'] = {'pp': 0}
    elif pp_obs==1:
        obs_dict['pp'] = {'pp': 1}

    if revenue_obs==0:
        obs_dict['revenue'] = {'below': 1}
    elif revenue_obs==1:
        obs_dict['revenue'] = {'above': 1}
    elif revenue_obs==2:
        obs_dict['revenue'] = {'neutral': 1}

    if mood_obs==0:
        obs_dict['mood'] = {'mood': 0}
    elif mood_obs==1:
        obs_dict['mood'] = {'mood': 1}

    if cp_obs==0:
        obs_dict['cp'] = {'cp': 0}
    elif cp_obs==1:
        obs_dict['cp'] = {'cp': 1}

    if raise_obs==0:
        obs_dict['raise'] = {'raise': 0}
    elif raise_obs==1:
        obs_dict['raise'] = {'raise': 1}

    approximate = query_model(approximate_trace, obs_dict, compare=True)

    exact = joint_probability(weather_obs, pp_obs, revenue_obs,

```

```

        mood_obs, cp_obs, raise_obs,
        compare=True)

results = pd.DataFrame({'app': approximate, 'exact': exact,
                        'names': ['rain', 'sun', 'snow', 'personal',
                                'below_revenue', 'above_revenue', 'neutral',
                                'mood', 'profits', 'raise']})

results = results.melt(id_vars='names', value_name='Probability',
                      var_name='Model')

figsize(10, 8)
matplotlib.rcParams['font.size'] = 18
sns.barplot('Probability', 'names', data = results, hue='Model')
plt.title('Probability Comparison')
plt.xlabel('Probability'); plt.ylabel('State');
plt.show()

```

3.1 Comparison with No Evidence

In [21]: compare()

```

Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>CategoricalGibbsMetropolis: [revenue, weather]
>BinaryGibbsMetropolis: [pp, mood, cp, raise_]

```

Approximate Probabilities from MCMC:

```

Weather: 0.30, 0.39, 0.30.
Personal Performance: 0.20, 0.80
Revenue: 0.40, 0.39, 0.21.
Boss's mood: 0.35, 0.65.
Profits: 0.52, 0.48.

```

Probability of a raise: 0.5345.

Exact Probabilities from the joint:

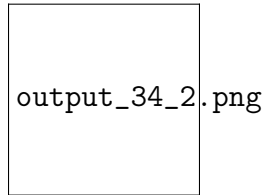
```

Weather 0.30, 0.40, 0.30.
Personal Performance: 0.20, 0.80.
Revenue: 0.40, 0.40, 0.20.
Boss's mood: 0.35, 0.65.

```

Profits: 0.53, 0.47.

Probabability of a raise: 0.5300.



The models are in close agreement, indicating that we did the approximate and exact inference correctly. The approximate does not agree exactly as expected, and we would expect it to approach the exact values as the number of samples increases. Now we can incorporate evidence to see how the posterior are impacted.

3.2 Comparisons with Evidence

```
In [22]: compare(weather_obs=2)
```

```
Multiprocess sampling (2 chains in 2 jobs)
```

```
CompoundStep
```

```
>BinaryGibbsMetropolis: [pp, mood, cp, raise_]
```

```
>CategoricalGibbsMetropolis: [revenue]
```

```
The number of effective samples is smaller than 25% for some parameters.
```

Approximate Probabilities from MCMC:

Weather: 0.00, 0.00, 1.00.

Personal Performance: 0.19, 0.81

Revenue: 0.41, 0.38, 0.20.

Boss's mood: 0.33, 0.67.

Profits: 0.54, 0.46.

Probability of a raise: 0.5265.

Exact Probabilities from the joint:

Weather 0.00, 0.00, 1.00.

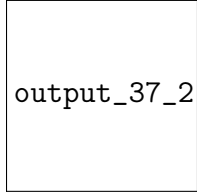
Personal Performance: 0.20, 0.80.

Revenue: 0.40, 0.40, 0.20.

Boss's mood: 0.33, 0.67.

Profits: 0.53, 0.47.

Probabability of a raise: 0.5361.



output_37_2.png

```
In [23]: compare(weather_obs=0, cp_obs=1)
```

Multiprocess sampling (2 chains in 2 jobs)

CompoundStep

>BinaryGibbsMetropolis: [pp, mood, raise_]

>CategoricalGibbsMetropolis: [revenue]

Approximate Probabilities from MCMC:

Weather: 1.00, 0.00, 0.00.

Personal Performance: 0.15, 0.85

Revenue: 0.07, 0.79, 0.14.

Boss's mood: 0.52, 0.48.

Profits: 0.00, 1.00.

Probability of a raise: 0.7680.

Exact Probabilities from the joint:

Weather 1.00, 0.00, 0.00.

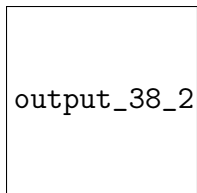
Personal Performance: 0.20, 0.80.

Revenue: 0.40, 0.40, 0.20.

Boss's mood: 0.53, 0.47.

Profits: 0.00, 1.00.

Probabability of a raise: 0.7675.



output_38_2.png

```
In [24]: compare(mood_obs=1, cp_obs=1)

Multiprocess sampling (2 chains in 2 jobs)
CompoundStep
>CategoricalGibbsMetropolis: [revenue, weather]
>BinaryGibbsMetropolis: [pp, raise_]
```

Approximate Probabilities from MCMC:

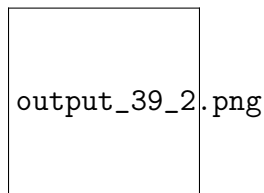
```
Weather: 0.22, 0.47, 0.31.
Personal Performance: 0.07, 0.93
Revenue: 0.09, 0.73, 0.18.
Boss's mood: 0.00, 1.00.
Profits: 0.00, 1.00.
```

Probability of a raise: 0.9075.

Exact Probabilities from the joint:

```
Weather 0.30, 0.40, 0.30.
Personal Performance: 0.20, 0.80.
Revenue: 0.40, 0.40, 0.20.
Boss's mood: 0.00, 1.00.
Profits: 0.00, 1.00.
```

Probabability of a raise: 0.9000.



4 Conclusions

In this notebook we looked at inference in a Baye's Network using both approximate sampling methods and exact inference. Both methods agree closely on the posterior probabilities without evidence and incorporating observations. We saw how with binary/discrete

variables, exact inference is not out of the question for a small network. For continuous variables, approximate methods are the only solution.

Overall, MCMC methods are a useful method to evaluate Bayes's Networks and are expected to converge on the "true" posterior with enough samples. With the decrease in cost of computing power, Markov Chain Monte Carlo methods are becoming a powerful tool for Bayesian Inference and allow us to generate an approximate answer for previously intractable problems. The simple problem evaluated here did not show the full benefits of Markov Chain Monte Carlo methods because it could be solved by hand, but we will explore the power of sampling methods in the next exercise using continuous variables in a hierarchical model.