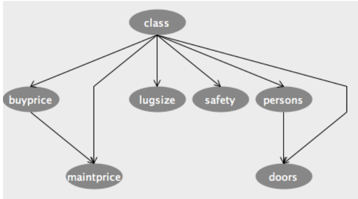
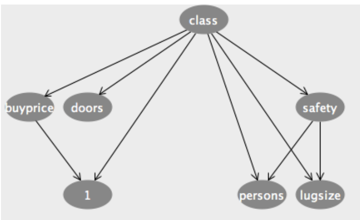
1. a. Naive Bayes model.

   b. 

   I add two edges, one between person and doors, another one between buyprice and maintprince. Because I think they are related given class, e.g. if a car can hold more person, it should have more doors to be convenient; a low maintprice can to some extent reconcile a high buyprice.

   c. 

(1 is maintprice)

   Changed parameters: maxNrOfParents = 5, randomOrder = True.

   This Bayes Net do not have $(buyprice \perp maintprice|class), (persons \perp safety|class), (lugsize \perp safety|class)$

2. a.

classifier	BN(a)	BN(b)	BN(c)	ZeroR
accuracy	85.3%	90.9%	94.1%	70%

   b. $\alpha = 0$

   c. Because the parameter of this network was manually tuned, and since the note can have more parent, the search is more complete.

3. a. $P(B|A = 1, C = 0) = P(B|A = 1)$
   $P(B)^{t \to \infty} = 0$

   Proof.

$$\begin{aligned}
P(B|A = 1, C = 0) &= \frac{P(B, A = 1, C = 0)}{\sum_B P(B, A = 1, C = 0)} \\
&= \frac{P(A = 1)P(B|A = 1)P(C = 0|B)}{P(A = 1)\sum_B P(B|A = 1)P(C = 0|B)} \\
&= \frac{0.5 \cdot P(C = 0|B)}{0.5 \cdot P(C = 0|B = 0) + 0.5 \cdot P(C = 0|B = 1)} \\
&= P(C = 0|B)
\end{aligned}$$

   For $B = 1$, $P(B = 1|A = 1, C = 0) = P(C = 0|B = 1) = 0$.
   Thus $B^1 = 0$, then we have $C^1 = 0$. No matter what A^1 is, we still get $B^2 = 0, C^2 = 0$ (Because $P(A = 0) = P(A = 1) = P(B = 0|A = 1) = P(B = 0|A = 0) = P(B = 1|A = 0) = P(B = 1|A = 1) = 0.5$). By induction $P(B)^{t \to \infty} = 0$ □

   b. Since $P(B = 0, C = 1) = P(B = 1, C = 0) = 0$, this gibbs chain is not regular. To make it mix, we make let $P(C = 1|B = 1) = P(C = 0|B = 0) = 1 - \epsilon$.
   Test with $\epsilon = 0.01$, number of iteration = 100000. The probabilty of A, B, C:

	=0	=1
P(A)	0.50062	0.49938
P(B)	0.50883	0.49117
P(C)	0.50909	0.49091

Listing 1: gibbsSampling.py

```
#Gibbs sampling test
import random
import numpy as np
```

```python
class GibbsSampling:
    """ init_state = [A, B, C]
    """
    def __init__(self, init_state, epsilon):
        self.state = init_state
        self.sample_count = np.zeros((3, 2))
        #self.prob = [0, 0, 0]
        self.epsilon = epsilon

    def sampling(self, numOfIteration):
        for i in range(numOfIteration):
                for j in range(len(self.sample_count)):
                        self.state[j] = (random.random() <= self.prob(j))
                        self.sample_count[j, self.state[j]] += 1
        return np.divide(self.sample_count.transpose(), np.sum(self.sample_count, axis
            =1)).transpose()

    """ return probability of certain state
    """
    def prob(self, stateNum):
        if stateNum == 0:
                return 0.5
        elif stateNum == 1:
                if self.state[2] == 1:
                        return 1 - self.epsilon
                else:
                        return self.epsilon
        else:
                if self.state[1] == 1:
                        return 1 - self.epsilon
                else:
                        return self.epsilon
print GibbsSampling([1, 0, 0], 0.1).sampling(10)
print GibbsSampling([1, 0, 0], 0.01).sampling(10)


print GibbsSampling([1, 1, 1], 0.1).sampling(100)
print GibbsSampling([1, 0, 0], 0.1).sampling(100)
print GibbsSampling([1, 0, 1], 0.1).sampling(100)
print GibbsSampling([0, 0, 0], 0.1).sampling(100000)

print GibbsSampling([1, 1, 1], 0.01).sampling(100000)
print GibbsSampling([1, 0, 0], 0.01).sampling(100000)
print GibbsSampling([1, 0, 1], 0.01).sampling(100000)
print GibbsSampling([0, 0, 0], 0.01).sampling(100000)
```