University of Waterloo
CS 486
r2knowle: 2023-10-1

Assignment # 2

Q1) To begin, the probability the car is involved in the accident is blue equals to the probability the car is blue given that it is identified as blue. In other words given the following events:

$$B = \text{Car is blue.}$$
$$I = \text{Car is identified as blue.}$$

Thus our goal is to solve for:

$$P(B|I)$$

Note that from Bayes theorem, this is equivalent to:

$$\begin{aligned} P(B|I) &= \frac{P(I|B) \times P(B)}{P(I)} \\ &= \frac{P(I|B) \times P(B)}{P(I|B)P(B) + P(I|\neg B)P(\neg B)} \\ &= \frac{0.8 \times 0.15}{0.8 \times 0.15 + 0.2 \times 0.85} \\ &= \frac{0.12}{0.12 + 0.17} \\ &= \frac{0.12}{0.29} \\ &= 41.38\% \end{aligned}$$

Thus the probability that the car is blue given that it is correctly identified is equal to 41.38%.

Q2) We get the following answers and rational to the given questions:

a) J is independant of A, as E blocks them by rule 3.
b) J is not independant of A given G, as we have the given undirected path:
$J- > H- > F- > B- > E- > A$
c) J is independant of A given F, as the path with G is blocked by I
d) J is independant of A given {G,F}, as there is no undirected path between them.
e) J is not independant of G, as there is an undirected path between them:
$J- > H- > F- > B- > E- > G$
f) J is not independant of G given I, as we have the given undirected path:
$J- > H- > F- > B- > E- > G$
g) J is independant of G given B, as I blockes the path
h) J is not independant of G given {I,B}, as by case 3 in d seperation I is blocking.
i) For G to be independant of J, we need to observe H. We dont need to oberserve any of the following:
$\{A, B, C, D, E, F, I\}$

Q3) Before outputting the results of the code, its important to know how I structured this problem. Each value in the CPT, is expressed an tuple of an array of random variables plus the probability. For example P(A=T, B=F) = 0.2 is saved as:

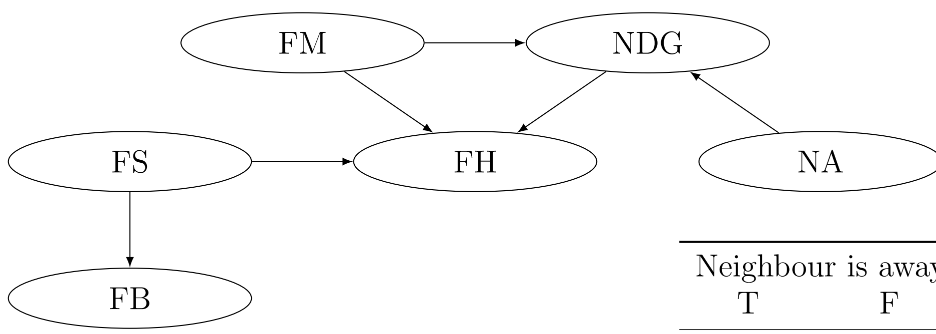
$$([\text{"}A\text{"}, \text{"}NB\text{"}], 0.2)$$

Where the "N" infront of the random variable represents not. Code is provided at the end of this document. Relevant pieces of code are provided in the questions that use them.

Q4a) Below we see the Bayenisan network for the given problem:

Moon	Neighbour	Neighour's dog is howling T	F
F	F	0	1
F	T	0.5	0.5
T	F	0.4	0.6
T	T	0.8	0.2

Full moon T	F
$\frac{1}{28}$	$\frac{27}{28}$



Fido is sick T	F
0.05	0.95

Neighbour is away T	F
0.3	0.7

Sick	Food not eaten T	F
F	0.1	0.9
T	0.6	0.4

Sick	Moon	Neighbour	Fido is howling T	F
F	F	F	0	1
F	F	T	0.2	0.8
F	T	F	0.4	0.6
F	T	T	0.65	0.35
T	F	F	0.5	0.5
T	F	T	0.75	0.25
T	T	F	0.9	0.1
T	T	T	0.99	0.01

Q4b) For this question we will make the following factors out of our random variables:

$$
\begin{aligned}
&f_1(FM) = P(FM)\\
&f_2(NA) = P(NA)\\
&f_3(FS) = P(FS)\\
&f_4(NDG, FM, NA) = P(NDG|FM, NA)\\
&f_5(NDG, NA) = \sum_{FM} f_4(NDG, FM, NA) f_1(FM)\\
&f_6(NDG) = \sum_{NA} f_5(NDG, NA) f_1(NA)\\
&f_7(FH, FS, FM, NDG) = P(FH|FS, FM, NDG)\\
&f_8(FH, FS, FM) = \sum_{NDG} f_7(FH, FS, FM, NDG) f_6(NDG)\\
&f_9(FH, FS) = \sum_{FM} f_8(FH, FS, FM) f_6(FM)\\
&f_{10}(FH) = \sum_{FS} f_8(FH, FS) f_6(FS)
\end{aligned}
$$

Plugging this into our code gives us the following answers:

```
#  FULL MOON
f1 = ((["M"], 1 / 28), (["NM"], 27 / 28))
#  NEIGHBOUR IS AWAY
f2 = ((["N"], 0.3), (["NN"], 0.7))
#  FIDO IS SICK
f3 = ((["S"], 0.05), (["NS"], 0.95))
#  NEIGHBOURS DOG IS HOWLING
f4 = ((["M", "N", "D"], 0.8), (["M", "N", "ND"], 0.2),
     (["M", "NN", "D"], 0.4), (["M", "NN", "ND"], 0.6),
     (["NM", "N", "D"], 0.5), (["NM", "N", "ND"], 0.5),
     (["NM", "NN", "D"], 0), (["NM", "NN", "ND"], 1))
f5 = ((['N', 'D'], 0.5107142857142857), (['N', 'ND'], 0.48928571428571427),
     (['NN', 'D'], 0.014285714285714285), (['NN', 'ND'], 0.9857142857142858))
f6 = ((['D'], 0.1632142857142857), (['ND'], 0.8367857142857142))
#  FIDO HOWLS
f7 = ((["S", "M", "D", "H"], 0.99), (["S", "M", "D", "NH"], 0.01),
     (["S", "M", "ND", "H"], 0.9), (["S", "M", "ND", "NH"], 0.1),
     (["S", "NM", "D", "H"], 0.75), (["S", "NM", "D", "NH"], 0.25),
     (["S", "NM", "ND", "H"], 0.5), (["S", "NM", "ND", "NH"], 0.5),
     (["NS", "M", "D", "H"], 0.65), (["NS", "M", "D", "NH"], 0.35),
     (["NS", "M", "ND", "H"], 0.4), (["NS", "M", "ND", "NH"], 0.6),
     (["NS", "NM", "D", "H"], 0.2), (["NS", "NM", "D", "NH"], 0.8),
     (["NS", "NM", "ND", "H"], 0), (["NS", "NM", "ND", "NH"], 1))
f8 = ((['H', 'S', 'M'], 0.9146892857142856), (['S', 'NH', 'M'], 0.0853107142857143),
     (['H', 'S', 'NM'], 0.5408035714285714), (['S', 'NH', 'NM'], 0.45919642857142856),
     (['H', 'NS', 'M'], 0.44080357142857146), (['NS', 'NH', 'M'], 0.5591964285714285),
     (['H', 'NS', 'NM'], 0.03264285714285714), (['NS', 'NH', 'NM'], 0.9673571428571428))
f9 = ((['H', 'S'], 0.5541566326530611), (['S', 'NH'], 0.4458433673469388),
     (['H', 'NS'], 0.047220025510204086), (['NS', 'NH'], 0.9527799744897958))
f10 = ((['H'], 0.07256685586734693), (['NH'], 0.9274331441326529))
```

Thus, there's only a 7.26% chance that Fido will howl.

Q4c) We now need to solve for FS, given FH = 1 (Fido is howling) and FM = 1. Thus we get the factors:

$$
\begin{aligned}
&f_1(FM = 1) = P(FM = 1) = 1 \\
&f_2(NA) = P(NA) \\
&f_3(NDG, FM = 1, NA) = P(NDG|FM = 1, NA) \\
&f_4(NDG, NA) = f_4(NDG, FM = 1, NA)f_1(FM = 1) \\
&f_5(NDG) = \sum_{NA} f_5(NDG, NA)f_1(NA) \\
&f_6(FS, NDG) = P(FS|FH = 1, FM = 1, NDG) \\
&f_7(FS) = \sum_{NDG} f_7(FS, NDG)f_6(NDG) \\
&f_8(FS) = P(FS) \\
&f_9(FS) = f_7(FS)f_8(FS)
\end{aligned}
$$

Plugging this into our code gives us the following answers (note we only normalize at the end):

```
#   FULL MOON
f1 = (([], 0.03571428571428571))
#   NEIGHBOUR IS AWAY
f2 = ((["N"], 0.3), (["NN"], 0.7))
#   NEIGHBOURS DOG IS HOWLING
f3 = ((['N', 'D'], 0.8), (['N', 'ND'], 0.2), (['NN', 'D'], 0.4), (['NN', 'ND'], 0.6))
f4 = ((['N', 'D'], 0.8), (['N', 'ND'], 0.2), (['NN', 'D'], 0.4), (['NN', 'ND'], 0.6))
f5 = ((['D'], 0.52), (['ND'], 0.48))
#   FIDO SICK
f6 = ((['S', 'D'], 0.99), (['S', 'ND'], 0.9),
     (['NS', 'D'], 0.65), (['NS', 'ND'], 0.4))
f7 = ((['S'], 0.9468000000000001), (['NS'], 0.53))
f8 = ((['S'], 0.05), (['NS'], 0.95))
f9 = ((['S'], 0.0859414712076), (['NS'], 0.914058528792))
```

Thus there is a 8.59% chance that Fido is sick.

Q4d) We now need to solve for FS, given FH = 1 (Fido is howling) and FM = 1, FB = 1. Thus we get the factors:

$$
\begin{aligned}
&f_1(FM=1) = P(FM=1) = 1 \\
&f_2(NA) = P(NA) \\
&f_3(NDG, FM=1, NA) = P(NDG|FM=1, NA) \\
&f_4(NDG, NA) = f_4(NDG, FM=1, NA)f_1(FM=1) \\
&f_5(NDG) = \sum_{NA} f_5(NDG, NA)f_1(NA) \\
&f_6(FS, NDG) = P(FS|FH=1, FM=1, NDG) \\
&f_7(FS) = \sum_{NDG} f_7(FS, NDG)f_6(NDG) \\
&f_8(FS) = P(FS) \\
&f_9(FS) = P(FS|FB=1) \\
&f_10(FS) = f_7(FS)f_8(FS)f_9(FS)
\end{aligned}
$$

Plugging this into our code gives us the following answers (note we only normalize at the end):

```
#   FULL MOON
f1 = (([], 0.03571428571428571))
#   NEIGHBOUR IS AWAY
f2 = ((["N"], 0.3), (["NN"], 0.7))
#   NEIGHBOURS DOG IS HOWLING
f3 = ((['N', 'D'], 0.8), (['N', 'ND'], 0.2), (['NN', 'D'], 0.4), (['NN', 'ND'], 0.6))
f4 = ((['N', 'D'], 0.8), (['N', 'ND'], 0.2), (['NN', 'D'], 0.4), (['NN', 'ND'], 0.6))
f5 = ((['D'], 0.52), (['ND'], 0.48))
#   FIDO SICK
f6 = ((['S', 'D'], 0.99), (['S', 'ND'], 0.9),
    (['NS', 'D'], 0.65), (['NS', 'ND'], 0.4))
f7 = ((['S'], 0.9468000000000001), (['NS'], 0.53))
f8 = ((['S'], 0.05), (['NS'], 0.95))
f9 = ((['S'], 0.6), (['NS'], 0.1))
f10 = ((['S'], 0.360667394672), (['NS'], 0.639332605328))
```

Thus there is a 34.76% chance that Fido is sick.

Q4e) We now need to solve for FS, given FH = 1 (Fido is howling) and FM = 1, FB = 1, NA = 1. Thus we get the factors:

$$
\begin{aligned}
f_1(FM = 1) &= P(FM = 1) = 1 \\
f_2(NA = 1) &= P(NA = 1) \\
f_3(NDG) &= P(NDG|FM = 1, NA = 1) \\
f_4(FS, NDG) &= P(FS|FH = 1, FM = 1, NDG) \\
f_5(FS) &= \sum_{NDG} f_7(FS, NDG) f_6(NDG) \\
f_6(FS) &= P(FS|FB = 1) \\
f_7(FS) &= P(FS) \\
f_8(FS) &= f_5(FS) f_6(FS) f_7(FS)
\end{aligned}
$$

Plugging this into our code gives us the following answers (note we only normalize at the end):

```
#   FULL MOON
f1 = (([], 0.03571428571428571))
#   NEIGHBOUR IS AWAY
f2 = ((["N"], 0.3))
#   NEIGHBOURS DOG IS HOWLING
f3 = ((['D'], 0.8), ('ND'], 0.2))
#   FIDO SICK
f4 = ((['S', 'D'], 0.99), (['S', 'ND'], 0.9),
     (['NS', 'D'], 0.65), (['NS', 'ND'], 0.4))
f5 = ((['S'], 0.972), (['NS'], 0.576))
f6 = ((['S'], 0.6), (['NS'], 0.1))
f7 = ((['S'], 0.05), (['NS'], 0.95))
f8 = ((['S'], 0.375152687895), (['NS'], 0.624847312105))
```

Thus there is a 91.01% chance that Fido is sick.

Q5a) Kate needs to provide the probabilites between each state $P(S_T|S_{T-1})$. Kate will also need to provide the observation model, which for any observation $num \in \{1, 2, 3, 4, 5\}$ is $P(O_T^{num}|S_T)$.

Q5b) We assume that each observation from time T, is independent of everything given state T. We also assume that each state is independent is independent of everything given the state before it.

Q5c) Given so many sensors, its possible that some of our observations are inaccurate and wont as useful for predictions. This could be due to the fact that this problem is stochastic and not deterministic, so sensor readings can be inaccurate. This could also be due to over fitting, which can happen when we over train on the training data (having so many observations).

Below is the code used to solve Q3 and Q4:

```
import numpy as np


class Factor:
    def __init__(self, setOfFactors):
        self.SetOfValues = setOfFactors

    def getAllTerms(self):
        return self.SetOfValues
    def getSizeOfFactor(self):
        return len(self.SetOfValues[0][0])
    def getListOfVariables(self):
        allVariables = set()
        for entry in self.SetOfValues:
            for variable in entry[0]:
                allVariables.add(variable)
        return allVariables

    def getVariable(self, variable):
        numberOfRelivent = []
        for factor in self.SetOfValues:
            listOfVals = set(factor[0])
            if variable in listOfVals:
                numberOfRelivent += [factor]
        return numberOfRelivent

def restrict(factor, variable, value):
    if not value:
        variable = "N" + variable

    terms = factor.getVariable(variable)
    updatedTerms = []
    for term in terms:
        factors = term[0]
        probability = term[1]
        newfactor = []
        for factor in factors:
            if factor != variable:
                newfactor += [factor]

        updatedTerms.append((newfactor, probability))

    return Factor(updatedTerms)


def multiply(factor1, factor2):

    newTerms = []
    largestTerm = max(len(factor1.getAllTerms()[0][0]), len(factor2.getAllTerms()[0][0]))
    for term1 in factor1.getAllTerms():
        for term2 in factor2.getAllTerms():
            combine = set(term1[0]).union(set(term2[0]))
            if len(combine) == largestTerm:
                newTerms.append((list(combine), term1[1]*term2[1]))

    return Factor(newTerms)
```

```
def sumout(factor, variable):
    positiveTerms = restrict(factor, variable, True)
    negativeTerms = restrict(factor, variable, False)

    updatedTerms = []
    for posTerm in positiveTerms.getAllTerms():
        for negTerm in negativeTerms.getAllTerms():
            if sorted(posTerm[0]) == sorted(negTerm[0]):
                updatedTerms.append((posTerm[0], negTerm[1]+posTerm[1]))

    return Factor(updatedTerms)

def inference(factorList, queryVarible, orderedListOfHiddenVariables, evidenceList):
    newFactorList = []
    for V in evidenceList:

        for factor in factorList:

            if V in factor.getListOfVariables():
                if len(V) == 1:
                    newFactorList.append(restrict(factor, V, True))
                else:
                    newFactorList.append(restrict(factor, V, False))
            else:
                newFactorList.append(factor)

        factorList = []

        for factor in newFactorList:
            factorList.append(factor)

    for currentVar in orderedListOfHiddenVariables:
        f1 = factorList[0]
        f2 = factorList[0]

        newFactorList = []
        for factor in factorList:

            if currentVar in factor.getListOfVariables():
                if len(factor.getListOfVariables()) == 2:
                    f1 = factor
                else:
                    f2 = factor
            else:
                newFactorList.append(factor)

        f3 = multiply(f1, f2)
        f3 = sumout(f3, currentVar)
        newFactorList.append(f3)
        factorList = newFactorList

    return factorList[0]

def normalize(factor):
    sum = factor.getAllTerms()[0][1] + factor.getAllTerms()[1][1]

    normalized1 = (factor.getAllTerms()[0][0], factor.getAllTerms()[0][1] / sum)
    normalized2 = (factor.getAllTerms()[1][0], factor.getAllTerms()[1][1] / sum)
```

```
    return Factor([normalized1, normalized2])

# Values for validation
f1values = ((["A", "B"], 0.9), (["A", "NB"], 0.1), (["NA", "B"], 0.4), (["NA", "NB"], 0.6))
f2values = ((["B", "C"], 0.7), (["B", "NC"], 0.3), (["NB", "C"], 0.8), (["NB", "NC"], 0.2))

g1values = ((["A"], 0.9), (["NA"], 0.1))
g2values = ((["A", "B"], 0.9), (["A", "NB"], 0.1), (["NA", "B"], 0.4), (["NA", "NB"], 0.6))
g3values = ((["B", "C"], 0.7), (["B", "NC"], 0.3), (["NB", "C"], 0.2), (["NB", "NC"], 0.8))

f1 = Factor(g1values)
f2 = Factor(g2values)
f3 = Factor(g3values)

#   FULL MOON
a1values = ((["M"], 1 / 28), (["NM"], 27 / 28))
#   NEIGHBOUR IS AWAY
a2values = ((["N"], 0.3), (["NN"], 0.7))
#   NEIGHBOURS DOG IS HOWLING
a3values = ((["M", "N", "D"], 0.8), (["M", "N", "ND"], 0.2),
           (["M", "NN", "D"], 0.4), (["M", "NN", "ND"], 0.6),
           (["NM", "N", "D"], 0.5), (["NM", "N", "ND"], 0.5),
           (["NM", "NN", "D"], 0), (["NM", "NN", "ND"], 1))

#   FIDO IS SICK
a4values = ((["S"], 0.05), (["NS"], 0.95))
#   FIDO HOWLS
a5values = ((["S", "M", "D", "H"], 0.99), (["S", "M", "D", "NH"], 0.01),
           (["S", "M", "ND", "H"], 0.9), (["S", "M", "ND", "NH"], 0.1),
           (["S", "NM", "D", "H"], 0.75), (["S", "NM", "D", "NH"], 0.25),
           (["S", "NM", "ND", "H"], 0.5), (["S", "NM", "ND", "NH"], 0.5),
           (["NS", "M", "D", "H"], 0.65), (["NS", "M", "D", "NH"], 0.35),
           (["NS", "M", "ND", "H"], 0.4), (["NS", "M", "ND", "NH"], 0.6),
           (["NS", "NM", "D", "H"], 0.2), (["NS", "NM", "D", "NH"], 0.8),
           (["NS", "NM", "ND", "H"], 0), (["NS", "NM", "ND", "NH"], 1))
#   FOOD BOWL
a6values = ((["S", "B"], 0.6), (["S", "NB"], 0.4),
           (["NS", "B"], 0.1), (["NS", "NB"], 0.9))

# ========================= Q2B ===========================
FS = Factor(a4values)
FH = Factor(a5values)
FM = Factor(a1values)
NA = Factor(a2values)
NDG = Factor(a3values)

NDG = inference([FM, NA, NDG], "D", ["M", "N"], [])
FH = inference([FH, FS, FM, NDG], "H", ["D", "M", "S"], [])
# ========================= Q2C ===========================
FS = Factor(a4values)
FH = Factor(a5values)
FM = Factor(a1values)
NA = Factor(a2values)
NDG = Factor(a3values)

NDG = inference([NA, NDG], "D", ["N"], ["M"])
FS = inference([FH, NDG], "S", ["D"], ["M", "H"])
```

```python
# ========================= Q2D ===========================
FB = Factor(a6values)
FB = inference([FB], "S", [], ["B"])

# ========================= Q2E ===========================
FS = Factor(a4values)
FH = Factor(a5values)
FM = Factor(a1values)
NA = Factor(a2values)
NDG = Factor(a3values)
print("A")
NDG = inference([NDG], "D", [], ["M", "N"])
print(NDG.getAllTerms())
FS = inference([FH, NDG], "S", ["D"], ["M", "H"])
print(FS.getAllTerms())

FB = Factor(a6values)
FB = inference([FB], "S", [], ["B"])
print(FB.getAllTerms())
print("B")
print(normalize(multiply(FB, FS)).getAllTerms())
```