AI 3603 Artificial Intelligence: Principles and Techniques

By: LiuQiaoan (520030910220)

HW#: 3

December 7, 2022

Contents

I. INTRODUCTION

In this homework, I implement code for computing exact inferences in Bayesian networks of discrete random variables. After that, I use the implementation of Bayes Network to make query, which gives hidden facts behind the data.

II. CODING PART

In this section, I implement the code for computing exact inferences in Bayesian networks of discrete random variables.

A. joinFactors(Factor1,Factor2)

In this function, I use `pandas.merge` to join to factors together. There are two situation, both of which can be dealed with by `pandas.merge`.

- `Factor1` and `Factor2` have no common columns. For this situation, we need to extent the table and then crossly multiply the probability.
- `Factor1` and `Factor2` have common columns. For this situation, we need to concat the different columns together.

After merge two factors to one factor, we need to multiply their probability together to get new probability.

```python
## Join of two factors
## Factor1, Factor2: two factor tables
##
## Should return a factor table that is the join of factor 1 and 2.
## You can assume that the join of two factors is a valid operation.
## Hint: You can look up pd.merge for mergin two factors
def joinFactors(Factor1, Factor2):
    # your code
    intersect = set(Factor1).intersection(set(Factor2))
    intersect.remove("probs")
    if len(intersect) == 0:
        Factor =  pd.merge(Factor1, Factor2, how = "cross")
    else:
        Factor = pd.merge(Factor1, Factor2, on = list(intersect), how = "left")

    Factor["probs_x"] = Factor["probs_x"] * Factor["probs_y"]
    Factor.rename(columns={"probs_x":"probs"}, inplace=True)
    Factor.drop(columns=["probs_y"], inplace = True)

    return Factor
```

B. marginalizeFactor(factorTable, hiddenVar)

In this section, I use `pandas.groupby` to eliminate `hiddenVar`.

- If `hiddenVar` in `factorTable`'s columns, we group all the columns except `hiddenVar` and `"probs"`. However, if there no columns left, we only return `None`.
- If `hiddenVar` not in `factorTable`'s columns, we just return `factorTable`.

```python
## Marginalize a variable from a factor
## table: a factor table in dataframe
## hiddenVar: a string of the hidden variable name to be marginalized
##
## Should return a factor table that marginalizes margVar out of it.
## Assume that hiddenVar is on the left side of the conditional.
## Hint: you can look can pd.groupby
def marginalizeFactor(factorTable, hiddenVar):
    # your code
    labels = list(factorTable)
    if hiddenVar in labels:
        labels.remove(hiddenVar)
        labels.remove("probs")
        if labels:
            return factorTable.groupby(labels)["probs"].sum().to_frame().reset_index()
        else:
            return None
    else:
        return factorTable
```

C. marginalizeNetworkVariables(bayesNet, hiddenVar)

Since the parameters is a list of factors and a list of hiddenVar, so we need to call `marginalizeFactor` for each hidden variable. And because we generate `None` in `marginalizeFactor` when their are no other columns except hidden variable and `"probs"`, we need to filter the `None` element out.

```python
## Marginalize a list of variables
## bayesnet: a list of factor tables and each table iin dataframe type
## hiddenVar: a string of the variable name to be marginalized
## NOTE: hiddenVar should be a list of string of the variable name to be marginalized according to PDF.
##
## Should return a Bayesian network containing a list of factor tables that results
## when the list of variables in hiddenVar is marginalized out of bayesnet.
def marginalizeNetworkVariables(bayesNet, hiddenVar):
    # your code
    for h in hiddenVar:
        bayesNet = [marginalizeFactor(factorTable, h) for factorTable in bayesNet]
        bayesNet = [factor for factor in bayesNet if factor is not None]
    return bayesNet
```

D. evidenceUpdateFactor(factorTable, evidenceVars, evidenceVals)

In order to make the process of `evidenceUpdateNet`, I define a new function. In this function, I first check whether `evidenceVars` is a string, an empty list or a non-empty list.

- If `evidenceVars` is a string, we use a boolean series to filter evidence variable with their known value.
- If `evidenceVars` is a non-empty list, we can use `pandas.query` method to set multiple conditions.
- If `evidenceVars` is an empty list, we just return the initial `factorTable`.

```
def evidenceUpdateFactor(factorTable, evidenceVars, evidenceVals):
    # use `query` method.
    if isinstance(evidenceVars, list):
        if evidenceVars and set(evidenceVars).issubset(set(factorTable)):
            query_condition = [f"({var} == {val})" for var, val in zip(evidenceVars, evidenceVals)]
            return factorTable.query("&".join(query_condition)).reset_index().drop(columns = ["index"])
        else:
            return factorTable
    else:
        if evidenceVars and evidenceVars in set(factorTable):
            return factorTable[factorTable[evidenceVars] == int(evidenceVals)].reset_index().drop(columns = ["index
                "])
        else:
            return factorTable
```

E. evidenceUpdateNet(bayesnet, evidenceVars, evidenceVals)

For this function, we just need to call `evidenceUpdateFactor`.

```
## Update BayesNet for a set of evidence variables
## bayesnet: a list of factor and factor tables in dataframe format
## evidenceVars: a vector of variable names in the evidence list
## evidenceVals: a vector of values for corresponding variables (in the same order)
##
## Set the values of the evidence variables. Other values for the variables
## should be removed from the tables. You do not need to normalize the factors
def evidenceUpdateNet(bayesnet, evidenceVars, evidenceVals):
    # your code
    return [evidenceUpdateFactor(factorTable, evidenceVars, evidenceVals) for factorTable in bayesnet]
```

F. inference(bayesnet, hiddenVar, evidenceVars, evidenceVals)

For the last inference function, we can use `functools.reduce` to loop through the `bayesnet` to join each factor. Then we call `marginalizeNetworkVariables` to marginalize hidden variable, and call `evidenceUpdateNet` to update evidence. Last, we normalize the probability.

```
## Run inference on a Bayesian network
## bayesnet: a list of factor tables and each table iin dataframe type
## hiddenVar: a string of the variable name to be marginalized
## evidenceVars: a vector of variable names in the evidence list
## evidenceVals: a vector of values for corresponding variables (in the same order)
##
## This function should run variable elimination algorithm by using
## join and marginalization of the sets of variables.
## The order of the elimiation can follow hiddenVar ordering
## It should return a single joint probability table. The
## variables that are hidden should not appear in the table. The variables
## that are evidence variable should appear in the table, but only with the single
## evidence value. The variables that are not marginalized or evidence should
## appear in the table with all of their possible values. The probabilities
## should be normalized to sum to one.
def inference(bayesnet, hiddenVar, evidenceVars, evidenceVals):
    # your code
    bayesnet = reduce(joinFactors, bayesnet)
    bayesnet = marginalizeNetworkVariables([bayesnet], hiddenVar)
    bayesnet = evidenceUpdateNet(bayesnet, evidenceVars, evidenceVals)[0]
    bayesnet["probs"] = bayesnet["probs"] / bayesnet["probs"].sum()
    return bayesnet
```

III. WRITTEN PART

A. sub-problem 1

The total number of probabilities needed to store the full joint distribution is

$$2^8 * 4^3 * 8 = 2^{17}$$

B. sub-problem 2

First, I write down the code for both problem 2. I define an auxiliary function `Problem2Or4` to do inference in problem 2 and 4.

```python
## Problem 2
print("==================== Problem 2 BEGIN ====================")
# used in plt.pie
def make_autopct(values):
    def my_autopct(pct):
        return '{p:.2f}%'.format(p=pct)
    return my_autopct
riskFactorNet = pd.read_csv('RiskFactorsData.csv')
income      = readFactorTablefromData(riskFactorNet, ['income'])
smoke       = readFactorTablefromData(riskFactorNet, ['smoke', 'income'])
exercise    = readFactorTablefromData(riskFactorNet, ['exercise', 'income'])
long_sit    = readFactorTablefromData(riskFactorNet, ['long_sit', 'income'])
stay_up     = readFactorTablefromData(riskFactorNet, ['stay_up', 'income'])
bmi         = readFactorTablefromData(riskFactorNet, ['bmi', 'income', 'exercise', 'long_sit'])
diabetes    = readFactorTablefromData(riskFactorNet, ['diabetes', 'bmi'])
bp          = readFactorTablefromData(riskFactorNet, ['bp', 'exercise', 'long_sit', 'income', 'stay_up', 'smoke'])
cholesterol = readFactorTablefromData(riskFactorNet, ['cholesterol', 'exercise', 'stay_up', 'income', 'smoke'])
stroke      = readFactorTablefromData(riskFactorNet, ['stroke', 'bmi', 'bp', 'cholesterol'])
attack      = readFactorTablefromData(riskFactorNet, ['attack', 'bmi', 'bp', 'cholesterol'])
angina      = readFactorTablefromData(riskFactorNet, ['angina', 'bmi', 'bp', 'cholesterol'])
def Problem2Or4(flag, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp, cholesterol,
    stroke, attack, angina):
    def good_or_bad(flag, obsVals, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp,
        cholesterol, stroke, attack, angina):
        ## 1. Calculate p(diabetes|smoke,exercise,long_sit,stay_up)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, diabetes]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'diabetes', 'smoke', 'exercise','long_sit', 'stay_up'})
        obsVars  = ['smoke', 'exercise','long_sit','stay_up']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(diabetes|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals[3]}):
            ")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["diabetes"], autopct=make_autopct(p["probs"]))
        plt.title(f"p(diabetes|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals
            [3]})")
        plt.savefig(f"./output/Problem{flag}_p_of_diabetes_GIVEN_smoke_{obsVals[0]}_exercise_{obsVals[1]}_long_sit_
            {obsVals[2]}_stay_up_{obsVals[3]}.pdf")

        ## 2. Calculate p(stroke|smoke,exercise,long_sit,stay_up)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, cholesterol, stroke]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'stroke', 'smoke', 'exercise','long_sit', 'stay_up'})
        obsVars  = ['smoke', 'exercise','long_sit','stay_up']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(stroke|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals[3]}):")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["stroke"], autopct=make_autopct(p["probs"]))
        plt.title(f"p(stroke|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals
            [3]})")
        plt.savefig(f"./output/Problem{flag}_p_of_stroke_GIVEN_smoke_{obsVals[0]}_exercise_{obsVals[1]}_long_sit_{
            obsVals[2]}_stay_up_{obsVals[3]}.pdf")

        ## 3. Calculate p(attack|smoke,exercise,long_sit,stay_up)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, cholesterol, attack]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'attack', 'smoke', 'exercise','long_sit', 'stay_up'})
        obsVars  = ['smoke', 'exercise','long_sit','stay_up']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(attack|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals[3]}):")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["attack"], autopct=make_autopct(p["probs"]))
```

```python
        plt.title(f"p(attack|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals[3]})")
        plt.savefig(f"./output/Problem{flag}_p_of_attack_GIVEN_smoke_{obsVals[0]}_exercise_{obsVals[1]}_long_sit_{obsVals[2]}_stay_up_{obsVals[3]}.pdf")

        ## 4. Calculate p(angina|smoke,exercise,long_sit,stay_up)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, cholesterol, angina]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'angina', 'smoke', 'exercise','long_sit', 'stay_up'})
        obsVars  = ['smoke', 'exercise','long_sit','stay_up']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(angina|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals[3]}):")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["angina"], autopct=make_autopct(p["probs"]))
        plt.title(f"p(angina|smoke={obsVals[0]}, exercise={obsVals[1]}, long_sit={obsVals[2]}, stay_up={obsVals[3]})")
        plt.savefig(f"./output/Problem{flag}_p_of_angina_GIVEN_smoke_{obsVals[0]}_exercise_{obsVals[1]}_long_sit_{obsVals[2]}_stay_up_{obsVals[3]}.pdf")

    ## (a) Bad habits:
    obsVals = [1, 2, 1, 1]
    good_or_bad(flag, obsVals, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp,
        cholesterol, stroke, attack, angina)
    ## (a) Good habits:
    obsVals = [2, 1, 2, 2]
    good_or_bad(flag, obsVals, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp,
        cholesterol, stroke, attack, angina)

    def good_or_bad_b(flag, obsVals, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp,
        cholesterol, stroke, attack, angina):
        ## 1. Calculate p(diabetes|bp, cholesterol, bmi)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, cholesterol, diabetes]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'diabetes', 'bp', 'cholesterol','bmi'})
        obsVars  = ['bp', 'cholesterol', 'bmi']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(diabetes|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]}):")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["diabetes"], autopct=make_autopct(p["probs"]))
        plt.title(f"p(diabetes|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]})")
        plt.savefig(f"./output/Problem{flag}_p_of_diabetes_GIVEN_bp_{obsVals[0]}_cholesterol_{obsVals[1]}_bmi_{obsVals[2]}.pdf")

        ## 2. Calculate p(stroke|bp, cholesterol, bmi)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, cholesterol, stroke]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'stroke', 'bp', 'cholesterol','bmi'})
        obsVars  = ['bp', 'cholesterol','bmi']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(stroke|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]}):")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["stroke"], autopct=make_autopct(p["probs"]))
        plt.title(f"p(stroke|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]})")
        plt.savefig(f"./output/Problem{flag}_p_of_stroke_GIVEN_bp_{obsVals[0]}_cholesterol_{obsVals[1]}_bmi_{obsVals[2]}.pdf")

        ## 3. Calculate p(attack|bp, cholesterol, bmi)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, cholesterol, attack]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'attack', 'bp', 'cholesterol','bmi'})
        obsVars  = ['bp', 'cholesterol','bmi']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(attack|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]}):")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["attack"], autopct=make_autopct(p["probs"]))
        plt.title(f"p(attack|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]})")
        plt.savefig(f"./output/Problem{flag}_p_of_attack_GIVEN_bp_{obsVals[0]}_cholesterol_{obsVals[1]}_bmi_{obsVals[2]}.pdf")

        ## 4. Calculate p(angina|bp, cholesterol, bmi)
        risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, cholesterol, angina]
        factors = riskFactorNet.columns

        margVars = list(set(factors) - {'angina', 'bp', 'cholesterol','bmi'})
        obsVars  = ['bp', 'cholesterol','bmi']

        p = inference(risk_net, margVars, obsVars, obsVals)
        print(f"p(angina|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]}):")
        print(p)
        plt.figure()
        plt.pie(p["probs"], labels = p["angina"], autopct=make_autopct(p["probs"]))
        plt.title(f"p(angina|bp={obsVals[0]}, cholesterol={obsVals[1]}, bmi={obsVals[2]})")
        plt.savefig(f"./output/Problem{flag}_p_of_angina_GIVEN_bp_{obsVals[0]}_cholesterol_{obsVals[1]}_bmi_{obsVals[2]}.pdf")
```

```
    ## (b) Poor health:
    obsVals = [1, 1, 3]
    good_or_bad_b(flag, obsVals, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp,
         cholesterol, stroke, attack, angina)
    ## (b) Good health:
    obsVals = [3, 2, 2]
    good_or_bad_b(flag, obsVals, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp,
         cholesterol, stroke, attack, angina)

Problem2Or4(2,riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp, cholesterol, stroke,
     attack, angina)

print("=================== Problem 2 END ===================")
print()
```

(a)

The probability of the outcome if I have bad habits (smoke, don't exercise, long sitting and stay up), can be calculated by inference, the pie charts of the probability are listed below.

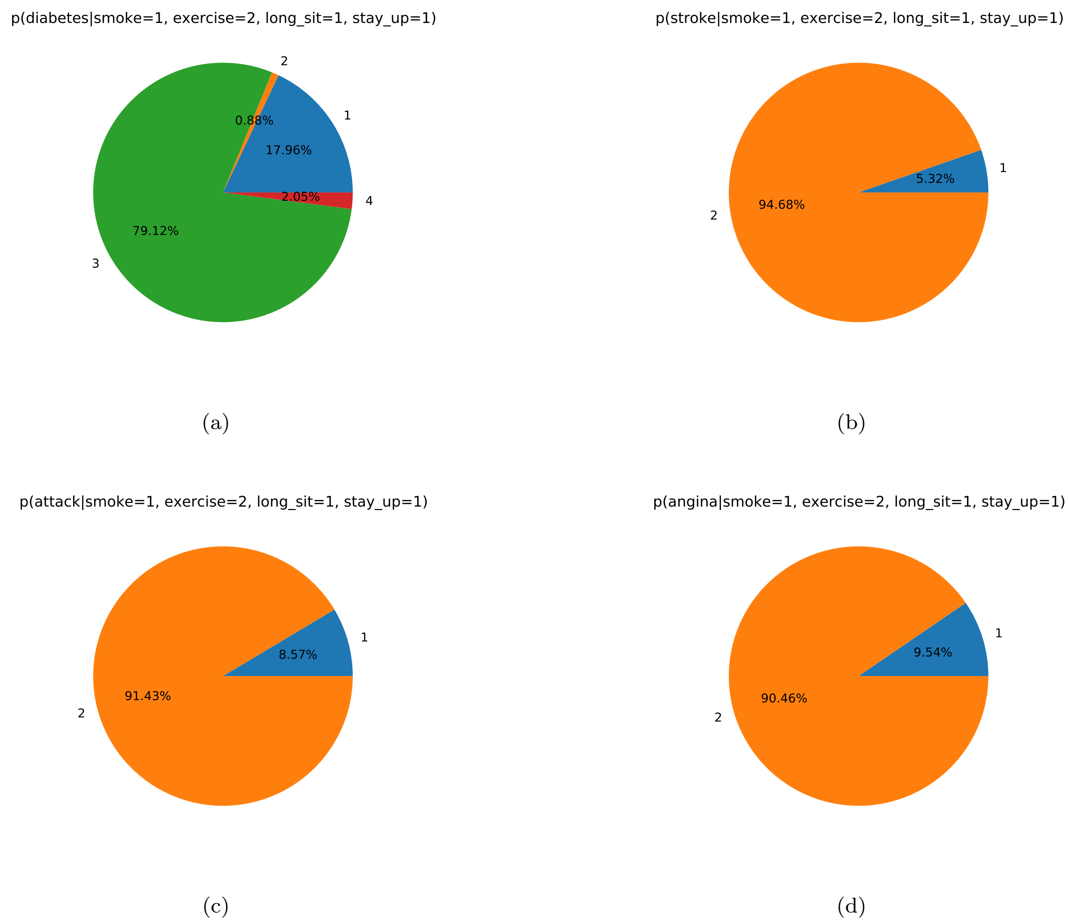


FIG. 1: The probability of the outcome if I have bad habits (smoke, don't exercise, long sitting and stay up)

The probability of the outcome if I have good habits (don't smoke, exercise, no long sitting and don't stay up), can be calculated by inference, the pie charts of the probability are listed below.

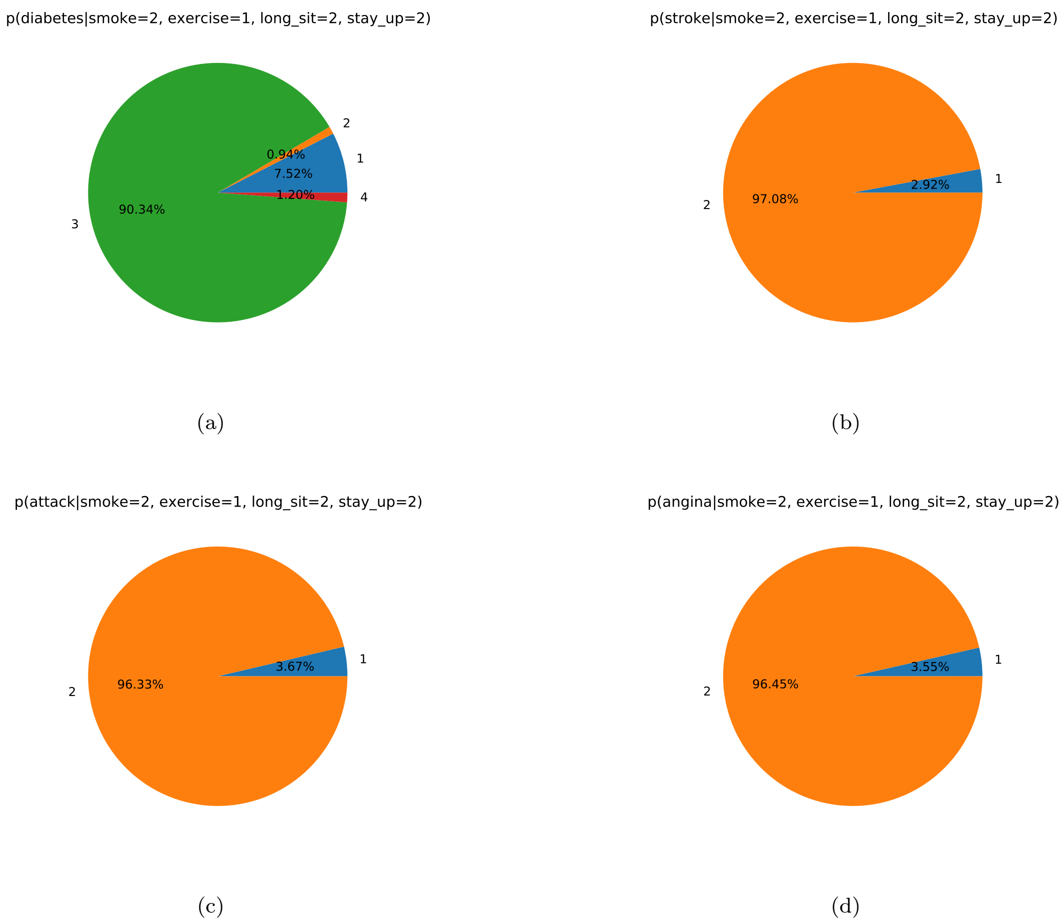


FIG. 2: The probability of the outcome if I have good habits (don't smoke, exercise, no long sitting and don't stay up)

(b)

The probability of the outcome if I have poor health (high blood pressure, high cholesterol, and overweight), and probability of the outcome if I have good health, can be calculated by inference. The pie charts are listed below.

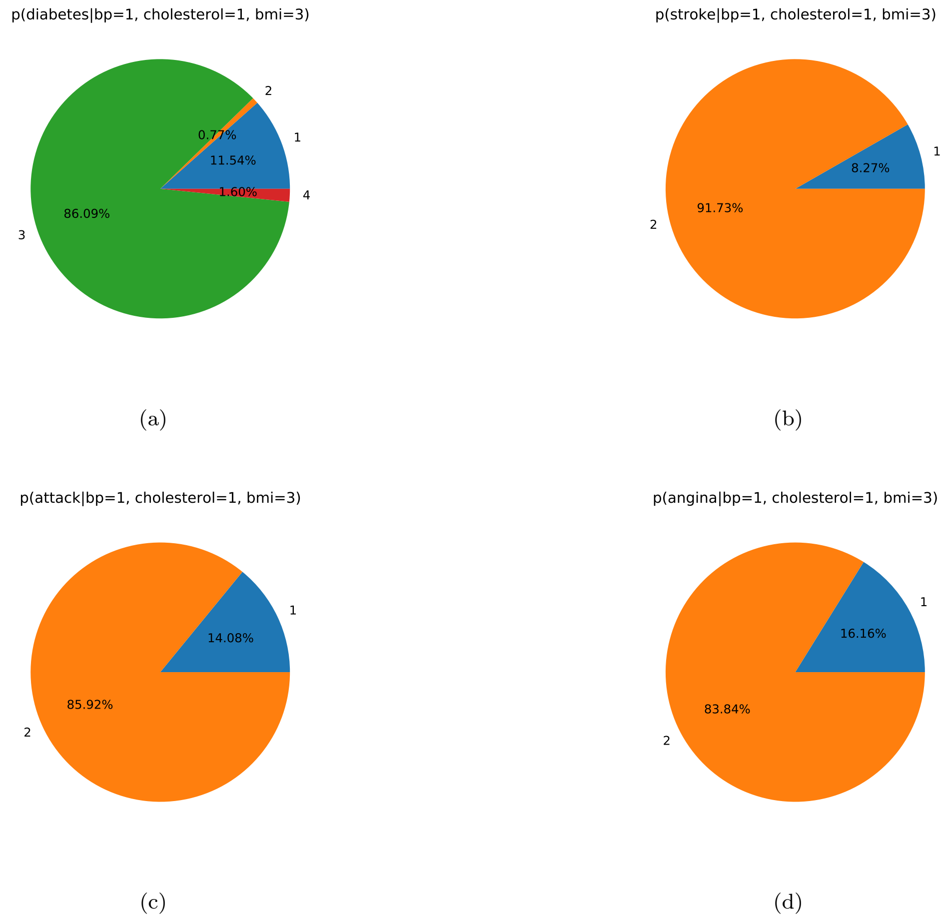
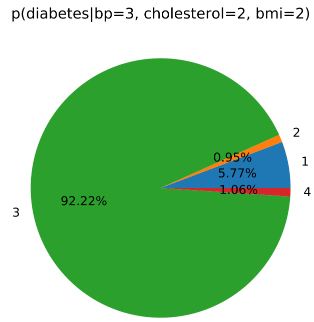
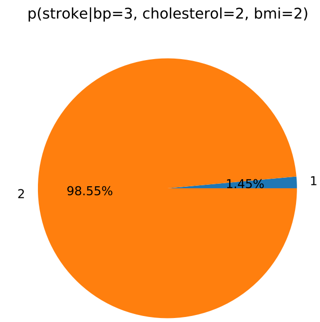


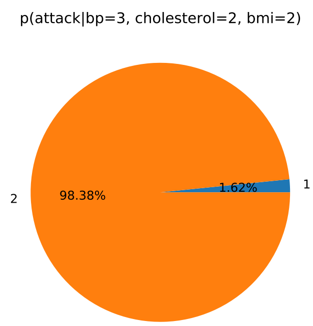
FIG. 3: The probability of the outcome if I have poor health (high blood pressure, high cholesterol, and overweight)



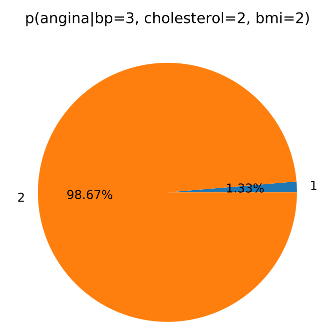
(a)



(b)



(c)



(d)

FIG. 4: The probability of the outcome if I have good health (low blood pressure, low cholesterol, and normal weight)

C. sub-problem 3

Now we are going to evaluate the effect a person's income has on their probability of having one of the four health outcomes (diabetes, stroke, heart attack, angina). I first write down the code.

```python
## Problem 3
print("==================== Problem 3 BEGIN ====================")
## 1. Calculate p(diabetes = 1 | incomes = i)
risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, diabetes]
factors = riskFactorNet.columns

margVars = list(set(factors) - {'diabetes', 'income'})
obsVars  = ['income']

p_diabetes = []
for obsVals in range(1,9):
    p = inference(risk_net, margVars, obsVars, [obsVals])
    p_diabetes.append(p["probs"][0])
print(p_diabetes)

## 2. Calculate p(stroke = 1 | incomes = i)
risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, stroke]
factors = riskFactorNet.columns

margVars = list(set(factors) - {'stroke', 'income'})
obsVars  = ['income']

p_stroke = []
for obsVals in range(1,9):
    p = inference(risk_net, margVars, obsVars, [obsVals])
    p_stroke.append(p["probs"][0])
print(p_stroke)

## 3. Calculate p(attack = 1 | incomes = i)
risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, attack]
factors = riskFactorNet.columns

margVars = list(set(factors) - {'attack', 'income'})
obsVars  = ['income']

p_attack = []
for obsVals in range(1,9):
    p = inference(risk_net, margVars, obsVars, [obsVals])
    p_attack.append(p["probs"][0])
print(p_attack)

## 4. Calculate p(angina = 1 | incomes = i)
risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, bp, angina]
factors = riskFactorNet.columns

margVars = list(set(factors) - {'angina', 'income'})
obsVars  = ['income']

p_angina = []
for obsVals in range(1,9):
    p = inference(risk_net, margVars, obsVars, [obsVals])
    p_angina.append(p["probs"][0])
print(p_angina)

## plot
plt.figure()
plt.plot(list(range(1,9)), p_diabetes, label="diabetes")
plt.plot(list(range(1,9)), p_stroke, label="stroke")
plt.plot(list(range(1,9)), p_attack, label="attack")
plt.plot(list(range(1,9)), p_angina, label="angina")
plt.legend()
plt.xlabel(r"$i$")
plt.ylabel(r"$P(y=1|income=i)$")
plt.title("The probability of outcome given income status")
plt.savefig("./output/Problem3.pdf")

print("==================== Problem 3 END ====================")
print()
```

And the probability given income status are showed below, the horizontal axis is $i = 1, 2, ..., 8$, and the vertical axis is $P(y = 1|income = i)$, where y is the outcome.

Then start to analysis. We can see from Figure 5 that the **more** of someone's outcome, the **less** probability they will have one of the four health outcomes. This can be a fact that the low-income person often work more busily, thus have less exercise, more long sitting, more staying up, more smoke, from problem 2 we know that these bad habits will more likely to cause high blood pressure, high cholesterol, and overweight. Ultimately, they will cause the four health outcomes.

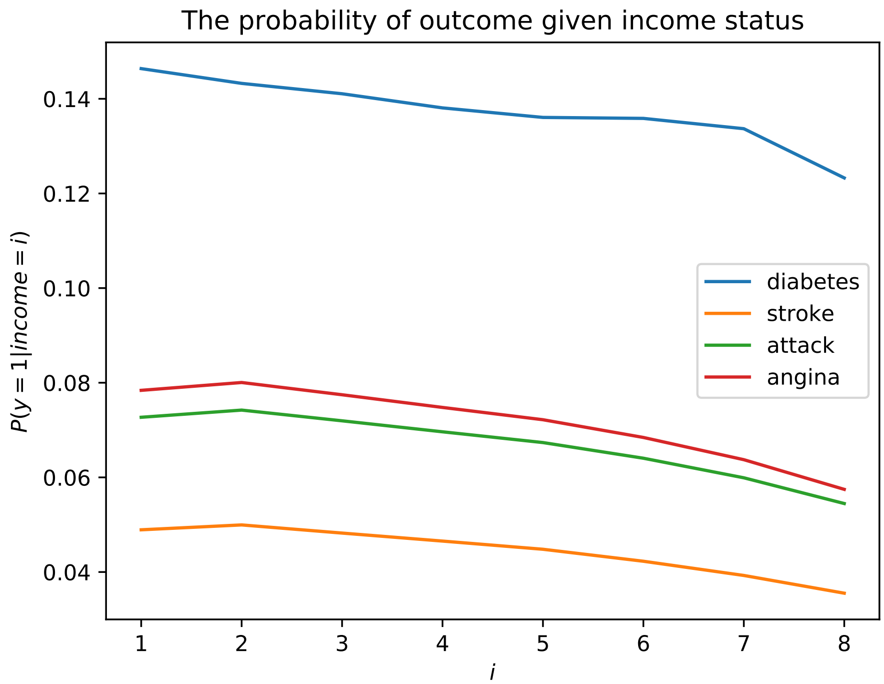


FIG. 5: The probability of outcome given income status, i.e. $P(y = 1|income = i)$

D. sub-problem 4

The **assumption** made in the first network is that *there are no direct effects between habits and health outcomes*. Now we are going to check whether the assumption is valid. In this problem, we can reuse the code in problem 2. I just need to reset the network.

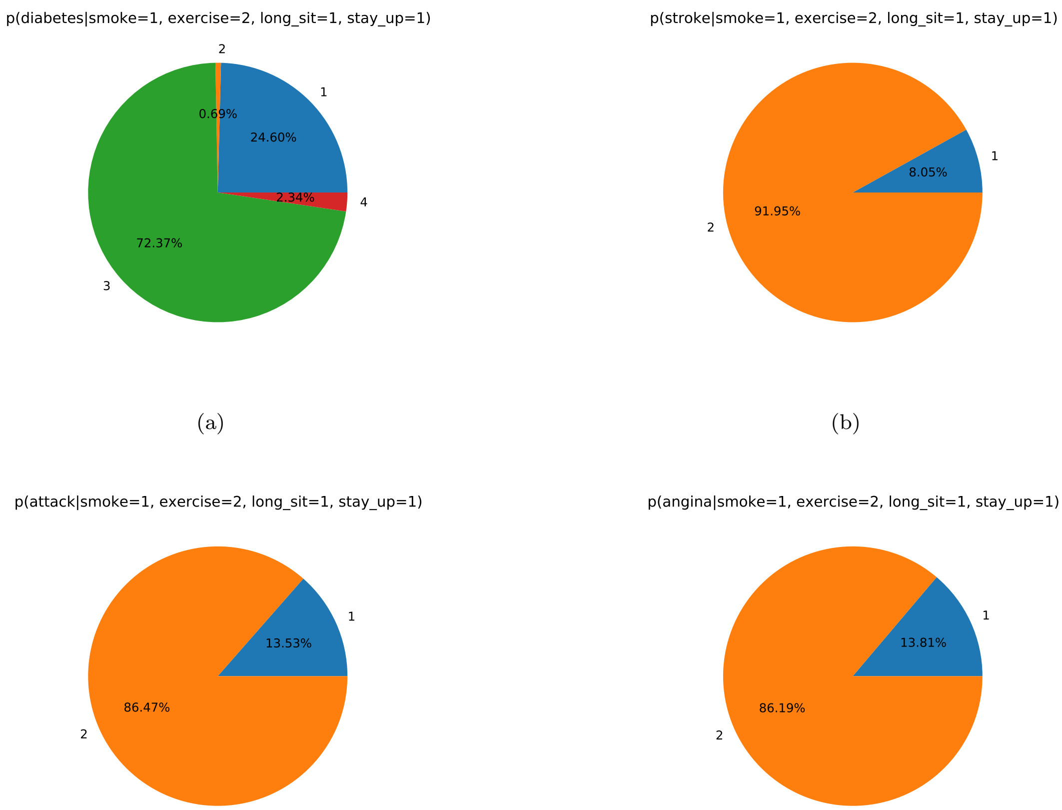
```
## Problem 4
print("=================== Problem 4 BEGIN ====================")
income      = readFactorTablefromData(riskFactorNet, ['income'])
smoke       = readFactorTablefromData(riskFactorNet, ['smoke', 'income'])
exercise    = readFactorTablefromData(riskFactorNet, ['exercise', 'income'])
long_sit    = readFactorTablefromData(riskFactorNet, ['long_sit', 'income'])
stay_up     = readFactorTablefromData(riskFactorNet, ['stay_up', 'income'])
bmi         = readFactorTablefromData(riskFactorNet, ['bmi', 'income', 'exercise', 'long_sit'])
diabetes    = readFactorTablefromData(riskFactorNet, ['diabetes', 'bmi', 'smoke', 'exercise']) # add 'smoke' and 'exercise'
bp          = readFactorTablefromData(riskFactorNet, ['bp', 'exercise', 'long_sit', 'income', 'stay_up', 'smoke'])
cholesterol = readFactorTablefromData(riskFactorNet, ['cholesterol', 'exercise', 'stay_up', 'income', 'smoke'])
stroke      = readFactorTablefromData(riskFactorNet, ['stroke', 'bmi', 'bp', 'cholesterol', 'smoke', 'exercise']) # add 'smoke' and 'exercise'
attack      = readFactorTablefromData(riskFactorNet, ['attack', 'bmi', 'bp', 'cholesterol', 'smoke', 'exercise']) # add 'smoke' and 'exercise'
angina      = readFactorTablefromData(riskFactorNet, ['angina', 'bmi', 'bp', 'cholesterol', 'smoke', 'exercise']) # add 'smoke' and 'exercise'

Problem2Or4(4, riskFactorNet, income, smoke, exercise, long_sit, stay_up, bmi, diabetes, bp, cholesterol, stroke, attack, angina)

print("=================== Problem 4 END ===================")
print()
```

(a)

The probability of the outcome if I have bad habits (smoke, don't exercise, long sitting and stay up), can be calculated by inference, the pie charts of the probability are listed below.



(a) (b)

(c) (d)

FIG. 6: The probability of the outcome if I have bad habits (smoke, don't exercise, long sitting and stay up)

The probability of the outcome if I have good habits (don't smoke, exercise, no long sitting and don't stay up), can be calculated by inference, the pie charts of the probability are listed below.

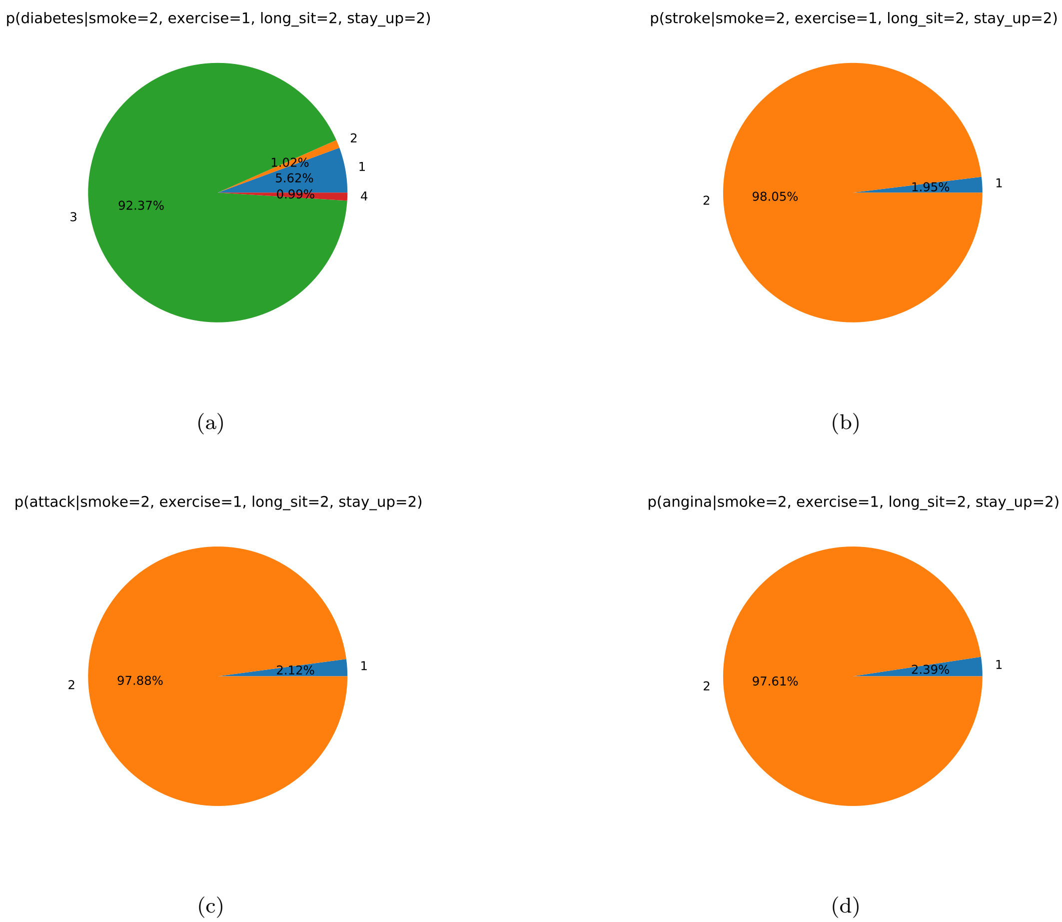


FIG. 7: The probability of the outcome if I have good habits (don't smoke, exercise, no long sitting and don't stay up)

(b)

The probability of the outcome if I have poor health (high blood pressure, high cholesterol, and overweight), and probability of the outcome if I have good health, can be calculated by inference. The pie charts are listed below.

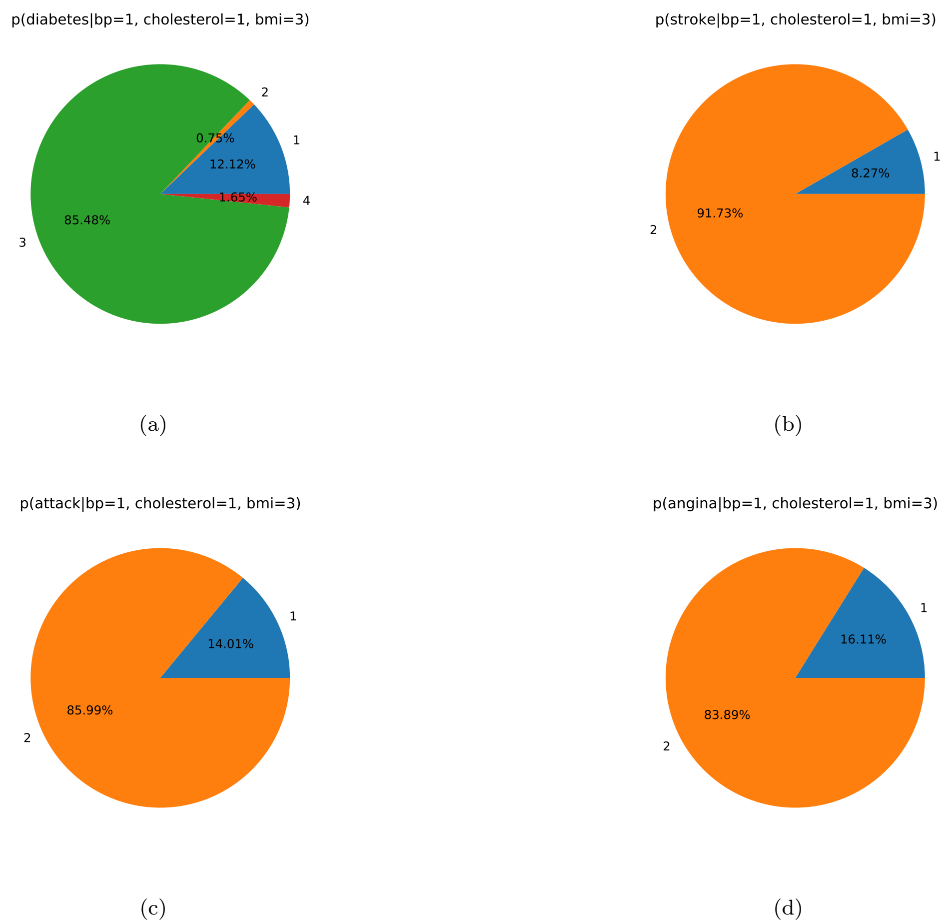


FIG. 8: The probability of the outcome if I have poor health (high blood pressure, high cholesterol, and overweight)

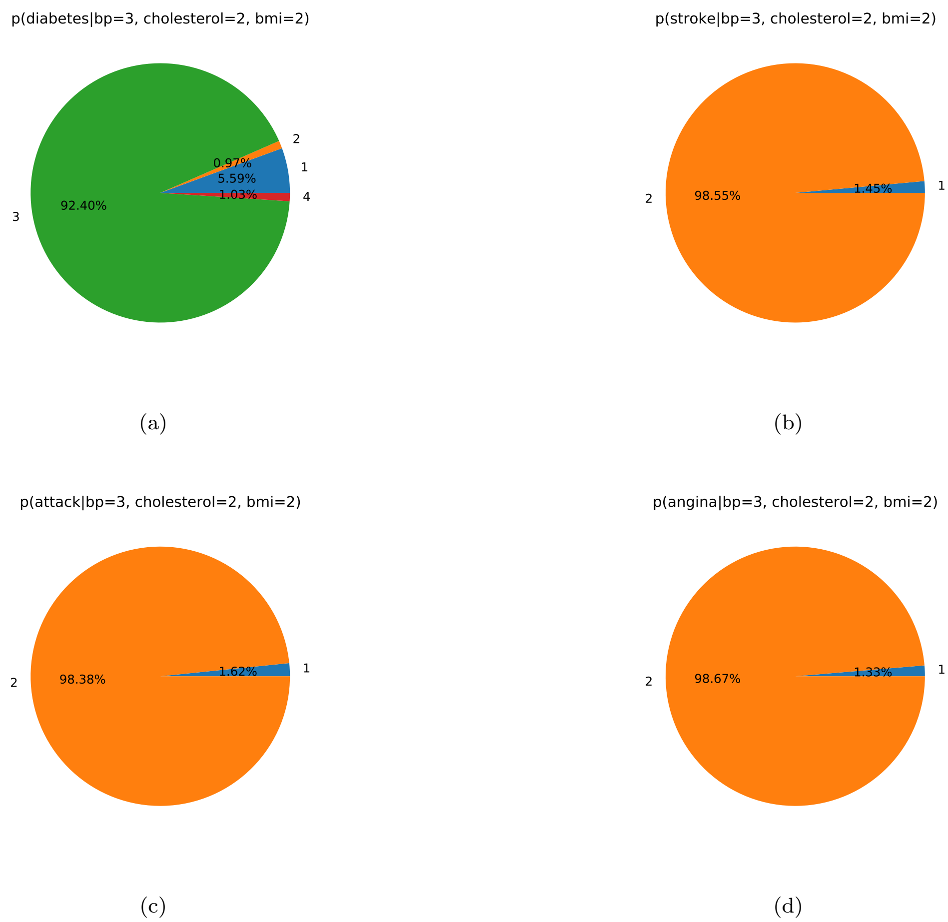


FIG. 9: The probability of the outcome if I have good health (low blood pressure, low cholesterol, and normal weight)

conclusion of sub-problem 4

Then, we can compare the result with problem 2. Compare Figure 1 with Figure 6, Figure 2 with Figure 7, Figure 4 with Figure 9, Figure 3 with Figure 8, we can find that they are all different. So the assumption is invalid.

E. sub-problem 5

When there are no edges between the four outcomes, the **assumption** made is that *there are no direct effects between the four outcomes.* Now we make a third network to test the assumption.

```python
## Problem 5
print("==================== Problem 5 BEGIN ====================")
## 1. Calculate p(stroke = 1 | diabetes = 1) and p(stroke = 1 | diabetes = 3) with the network in problem 4.
risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, diabetes, bp, cholesterol, stroke]
factors = riskFactorNet.columns

margVars = list(set(factors) - {'stroke','diabetes'})
obsVars  = ['diabetes']

obsVals  = [1]
p = inference(risk_net, margVars, obsVars, obsVals)
print("p(stroke = 1 | diabetes = 1) with the network in problem 4:")
print(p["probs"][0])

obsVals  = [3]
p = inference(risk_net, margVars, obsVars, obsVals)
print("p(stroke = 1 | diabetes = 3) with the network in problem 4:")
print(p["probs"][0])

## 2. Calculate p(stroke = 1 | diabetes = 1) and p(stroke = 1 | diabetes = 3) with third network.
stroke      = readFactorTablefromData(riskFactorNet, ['stroke', 'bmi', 'bp', 'cholesterol', 'smoke', 'exercise', '
    diabetes']) # add 'diabetes'

risk_net = [income, smoke, long_sit, stay_up, exercise, bmi, diabetes, bp, cholesterol, stroke]
factors = riskFactorNet.columns

margVars = list(set(factors) - {'stroke','diabetes'})
obsVars  = ['diabetes']

obsVals  = [1]
p = inference(risk_net, margVars, obsVars, obsVals)
print("p(stroke = 1 | diabetes = 1) with third network:")
print(p["probs"][0])

obsVals  = [3]
p = inference(risk_net, margVars, obsVars, obsVals)
print("p(stroke = 1 | diabetes = 3) with third network:")
print(p["probs"][0])

print("==================== Problem 5 END ====================")
```

After run the code, we get the output as:

```
==================== Problem 5 BEGIN ====================
p(stroke = 1 | diabetes = 1) with the network in problem 4:
0.04441718130847403
p(stroke = 1 | diabetes = 3) with the network in problem 4:
0.03995473229116679
p(stroke = 1 | diabetes = 1) with third network:
0.07654239679086357
p(stroke = 1 | diabetes = 3) with third network:
0.034456134859939015
==================== Problem 5 END ====================
```

FIG. 10: $P(stroke = 1|diabetes = 1)$ and $P(stroke = 1|diabetes = 3)$ in network 2 and network 3

From the output, we can see that the probability of $P(stroke = 1|diabetes = 1)$ and $P(stroke = 1|diabetes = 3)$ are both different for network 2 and network 3, so the assumption is invalid.

F. sub-problem 6

After run the initial **BayesNetworkTestScript.py**, I get the output as below, which is the same as that in the PDF document [1].

```
inference starts
   gauge  probs
0      0  0.315
1      1  0.685
   fuel  gauge  probs
0     0      0   0.81
1     0      1   0.19
   fuel  gauge     probs
0     0      0  0.257143
1     1      0  0.742857
      probs  battery  fuel  gauge
0  0.888889        0     1      0
1  0.111111        0     0      0
inference ends
income dataframe is
      probs  income
0  0.050848       1
1  0.059429       2
2  0.074042       3
3  0.094414       4
4  0.116356       5
5  0.150725       6
6  0.164430       7
7  0.289755       8
   smoke  long_sit  exercise  diabetes     probs
0      1         1         2         1  0.136815
1      1         1         2         2  0.008916
2      1         1         2         3  0.837218
3      1         1         2         4  0.017052
```

FIG. 11: My code runs correctly on all of the examples in **BayesNetworkTestScript.py**.

IV. CONCLUSION

In this homework, I combine the knowledge in the class with a real-world example, and use Bayes Network to analyze risk factors for certain health problems. During the procedure, I get familiar with `pandas` package, and enhance my ability of data handling.

[1] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.