# Assignment 1
# CMP9132M - Advanced Artificial Intelligence

Miguel Moreno Rodríguez - MOR18709146 - 18709146@students.lincoln.ac.uk

December 2019

# 1 Introduction

This assignment comprises two assessed Tasks. Task 1 is divided in two parts and these are TASKS ON PORBABILITIES. The second one is a TASK ON MARKOV MODELS. The objective for this assignment is to critically appraise a range of AI techniques and design and develop an AI-based software program for solving the proposed tasks.

# 2 Methodology

In this section we will describe the way the different methods and algorithms are implemented to solve each one of the proposed tasks.

## 2.1 Tasks on Probability

### 2.1.1 Task 1.A

The objective of this task is to implement a program to solve the *Rare Disease and test problem*. The program has as input the following probabilities:
$P(d)$ : prior probability of having a disease
$P(t|d)$ : probability that the test is positive given the person has the disease
$P(\neg t|\neg d)$ : probability that the test is negative given the person does not have the disease

Where $t$ means the test is positive and $d$ means the person has the disease.

This program must calculate $P(d|t)$ : the probability of having the disease given the test wast positive. The user should be able to change the initial probability values each time the program runs.

For approaching this problem, a program is implemented in file *bayes_disease.py* (listing 1) applying the Bayes Rule (1) to calculate $P(d|t)$.

$$P(A|B) = \frac{P(B|A)P(A)}{\sum_i^n P(B|A_i)P(A_i)} \tag{1}$$

Bayes Rule is applied in this case because it allows us to compute $P(A|B)$ in terms of $P(B|A)$ , $P(A)$ and $P(B)$, which in this case can be computed as $\sum P(B|A_i)P(A_i)$. This is really useful in medical diagnosis problems because it is common to have good estimates for these values as stated in (Russell and Norvig, 2009) . In our case, the Bayes Rule becomes:

$$P(d|t) = \frac{P(d|t)P(t)}{\sum_i^n P(t|d_i)P(d_i)} \tag{2}$$

The first thing we will need is our input probabilities. For that, the function *parse_arg()* is implemented. It asks the user for a valid input for each initial probability, filtering the input to check that it is a valid probability value (number between 0 and 1). The program will keep asking until all three initial values are valid.

Once we have the three initial probability values, the needed intermediate probabilities are obtained from them. The inputs provide us with $P(B|A)$ , $P(A)$ , and $P(\neg B|\neg A)$ . We need to obtain $P(B) = \sum P(B|A_i)P(A_i)$, which in this case can expand as $P(B) = P(B|A)P(A) + P(B|\neg A)P(\neg A)$. The final two values we need to compute this can be obtained as $P(\neg A) = 1 - P(A)$ and $P(B|\neg A) = 1 - P(\neg B|\neg A)$. Then we can apply the Bayes Rule to obtain our final porbability.

The task is tested with the following initial probabilities: $P(d) : 0.0001$ $P(t|d) : 0.99$ $P(t|d) : 0.95$ And the result obtained is: $P(d|t) = 0.001976$

### 2.1.2 Task 1.B

The objective of this task is to implement a program for solving a medical diagnosis problem using probabilistic inference. The program should have as an input a dataset from where it will perform parameter learning and the following Bayes Network structure:

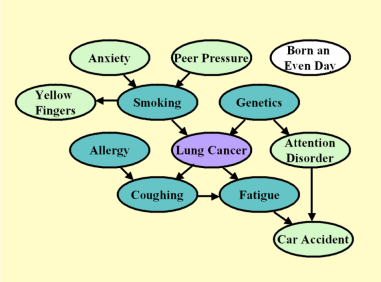


Figure 1: Bayesian Network structure for the Lung Cancer problem

The goal of this task is to obtain the probability distribution of Smoking Given Coughing and Fatigue $P(S|C, F)$. For that purpose, the method that is going to be implemented for inference is Rejection Sampling. This task is structured in three python files as follows:

**Parameter Learning:** For parameter learning the following equations will be applied to obtain the CPTs of this problem (maximum likelihood

estimation) (Nguyen, 2016) :

For CPTs with one variable:

$$P(X) = \frac{count(x) + 1}{count(X) + |X|}$$

For CPTs with two variables:

$$P(X) = \frac{count(x, y) + 1}{count(y) + |X|}$$

For CPTs with three variables:

$$P(X) = \frac{count(x, y, z) + 1}{count(y, z) + |X|}$$

This is done in the file *readcsv.py* (listing 2) . The first thing we do in this program is declaring the vectors that will store the CPTs of our problem. We also declare counter vectors that will store the count to be used on each of the presented equations. And name vectors that contain strings for the name of each probability, just for displaying purposes.Once we have declared all these vectors, we can read the dataset. For that, we use the function *reader()* from the *csv* Python library declaring ',' as our delimiter. Then we iterate through each row of the *csv* file with a *for* loop. Inside the main loop, we declare three individual *for* loops. Inside each one, we check, with *if* statements, for each possible probability value, for one, two or three variables respectively. If the row contains the combination of values we are looking for, we increase the counter for that combination. After iterating through the whole dataset and once we have all the counters we need, we apply the three equations presented earlier. Computing this way all the CPTs for our problem.

**Prior Sampling:** This CPTs will be imported in the file *sampling_parameters.py* (listing 3) . This file, as mentioned earlier, is based on the *PriorSampling.py* file provided in workshops. The net structure has been modified from the *burglary* net to this problem´s *lung cancer* net. Each probability value of the CPTs of this net is imported from the vectors created and populated in the previous file. The rest of the file implements the Prior Sampling algorithm. As stated in (Russell and Norvig, 2009) , this algorithm generates samples from the prior joint distribution specified by the network. Each variable is sampled according to the conditional distribution given the values already sampled for the variable's parents. This file creates the PriorSampling class, its functions will be used to implement the RejectionSampling algorithm.

**Rejection Sampling:** The Rejection Sampling algorithm is implemented in the file *luca_inference.py* (listing 4). This file makes use of the class *PriorSampling* implemented in the previous file. This file is structured in one main function and three auxiliar functions. The main function *RejctionSampling()* is the implementation of the rejection sampling algorithm. The pseudocode of this algorithm can be found in the lecture slides and is extracted from (Russell and Norvig, 2009).

As stated in (Russell and Norvig, 2009), Rejection Sampling is a method for approximate inference. The probability distribution obtained given the dataset *'lucas0_train.csv'* for $P(S|C,F)$ : probability of *Smoking* given *Coughing* and *Fatigue* is:

$P(S|C,F) = <0.84\ ,\ 0.16>$

The result seems logical as it is more likely that the person smokes given coughing and fatigue.

## 2.2 Task on Markov Models

### 2.2.1 Task 2

Considering a heater with two possible unknown states, ON and OFF, placed in a room where the measurable temperature can be Hot, Warm, or Cold. The objective of this task is to implement a program that, given any sequence of the above temperatures , returns the probability of observing such sequence.

The state diagram of the problem can be observed in figure 3

This program is implemented in the fle *markov.py* (listing 5) . The program should be able to get as input any sequence of the above temperatures. For that, the first thing we will do is implement the function *parse_arg()*. This function asks the user for an input , and then checks if the input is a valid sequence. Once we have processed the input, we can implement the solution for our problem. The probability of a sequence of observations $e1, e2, e3, ..., et$ is equal to the sum of the probabilities of the sequence for each possible ending state $xt$ (Russell and Norvig, 2009). which can be written as 3

$$P(E_t) = \sum_{xt} P(E_t, x_t) \tag{3}$$

If we apply recursively it can be written as 4

$$S_t = P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) S_{t-1} \tag{4}$$

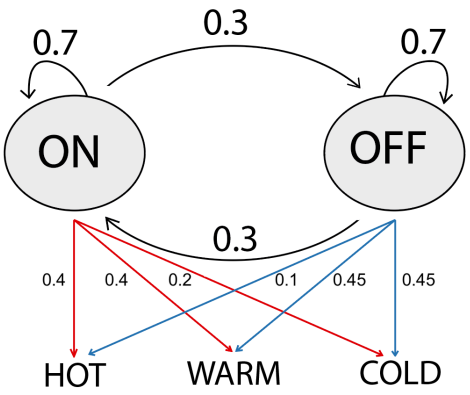


Figure 2: The probability between two consecutive time instants of remaining in the same state is 0.7, while the probability of switching to a different state is 0.3. In state ON, the probability of measuring a Warm or Hot temperature is 0.4, while the probability of measuring Cold is 0.2. In state OFF, the probability of measuring a Warm or Cold temperature is 0.45, while the probability of measuring Hot is 0.1.

being $S_t = P(E_t, x_t)$

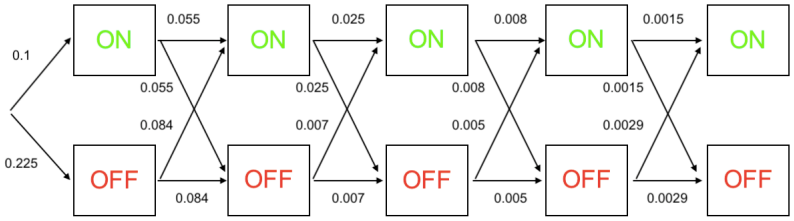


Figure 3: evolution of the probabilities for observing the sequence *Cold-Warm-Hot-Warm-Cold.*

In our python script, we compute 4 recursively for the length of the input sequence. For each state, we will have one probability for the next state being $ON$ and other one for the next state being $OFF$. We compute the final probability of having the complete sequence as the sum of the two final probabilities. In figure 3 we can see the evolution of the probabilities for observing the sequence *Cold-Warm-Hot-Warm-Cold.*

The final probability of having such sequence is 0.004399.

## References

Nguyen, L. (2016), 'Specifying prior probabilities in bayesian network by maximum likelihood estimation method', *Sylwan Journal* .

Russell, S. and Norvig, P. (2009), *Artificial Intelligence: A Modern Approach*, Prentice Hall Press.

# A Appendix - Code for each task in Python

## A.1 Task1a

```
#use python3

import sys
import random
import numpy
from decimal import Decimal

#these are my inputs
#PbGivenA = 0.1  #p(t/d)
#pA = 0.40  # p(d)
#pNotbGivenNotA = 0.2  #p(¬t/¬d)

#Type in p(t/d):0.99
#Type in p(d):0.0001
#Type in p(¬t/¬d):0.95


def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False


def parse_arg(s):
  input_valid = False
  while input_valid == False:
    raw_in = input('Type in ' + s + ':')
    if is_number(raw_in):
      inp = float(raw_in)
      if inp >= 0 and inp <= 1:
        input_valid = True
        #print('valid')
```

```
      else:
        print('Not a valid value for ' + s + ' (0-1)')
    else:
      print('Not a valid value for ' + s + ' (0-1)')
  return inp


if __name__ == "__main__":

  in1 = parse_arg('p(t/d)')
  in2 = parse_arg('p(d)')
  in3 = parse_arg('p(¬t/¬d)')

  PbGivenA = in1  #p(t/d)
  pA = in2 # p(d)
  pNotbGivenNotA = in3 #p(¬t/¬d)

  #calculate the needed probabilities
  pNotA = 1-pA
  pbGivenNotA = 1 - pNotbGivenNotA

  #apply bayes rule

  pB = (PbGivenA*pA) + (pbGivenNotA*pNotA) #p(t)

  PaGivenB = (PbGivenA * pA) / pB # output = p(d/t)

  print('P(d|t) = ' + str(round(PaGivenB,6)))
```

Listing 1: bayes_disease.py

## A.2 Task1b

```
import csv

'''
row[0] #smoking
row[1] #Yellow f
row[2] #Anxiety
row[3] #Peerpressure
row[4] #genetics
row[5] #attention disorder
row[6] #born evenday
row[7] #car accident
row[8] #fatigue
row[9] #allergy
row[10] #coughing
row[11] #lung cancer
'''
```

```
vector1 = [3,2,4,6,9]  #declare a vector with the identifiers for
    each variable
v1_names = ['P(PP)','P(A)','P(G)','P(ED)','P(All)']
#CPT with one variable
v1_probs =[0.0,0.0,0.0,0.0,0.0] #declare vectors to store the
    probability values
v1_probs_f =[0.0,0.0,0.0,0.0,0.0]
v1_names_f = ['P(¬PP)','P(¬A)','P(¬G)','P(¬ED)','P(¬All)']
cont1 = [0,0,0,0,0] #declare


vector2 = [1,5] #declare a vector with the identifiers for each
    variable
vector2_var = [0,4]  #declare a vector with the identifiers for
    evidence variables
v2_names_tt = ['P(Y|S)','P(AD|G)']
v2_names_ft = ['P(¬Y|S)','P(¬AD|G)']
v2_names_tf = ['P(Y|¬S)','P(AD|¬G)']
v2_names_ff = ['P(¬Y|¬S)','P(¬AD|¬G)']
#CPT with two variables
v2_probs_tt =[0.0,0.0] #declare vectors to store the probability
    values
v2_probs_ft =[0.0,0.0]
v2_probs_tf =[0.0,0.0]
v2_probs_ff =[0.0,0.0]
cont2tt = [0,0]
cont2ft = [0,0]
cont2tf = [0,0]
cont2ff = [0,0]
cont2_vart = [0,0]
cont2_varf = [0,0]

vector3 = [0,11,10,8,7]   #declare a vector with the identifiers
    for each variable
vector3_var1 = [2,0,9,11,5] #declare a vector with the identifiers
     for first evidence variables
vector3_var2 = [3,4,11,10,8] #declare a vector with the
    identifiers for second evidence variables
v3_names_ttt = ['P(S|A,PP)','P(LC|S,G)','P(C|All,LC)','P(F|LC,C)',
    'P(CA|AD,F)']
v3_names_tft = ['P(S|¬A,PP)','P(LC|¬S,G)','P(C|¬All,LC)','P(F|¬LC,
    C)','P(CA|¬AD,F)']
v3_names_ttf = ['P(S|A,¬PP)','P(LC|S,¬G)','P(C|All,¬LC)','P(F|LC,
    ¬C)','P(CA|AD,¬F)']
v3_names_tff = ['P(S|¬A,¬PP)','P(LC|¬S,¬G)','P(C|¬All,¬LC)','P(F|
    ¬LC,¬C)','P(CA|¬AD,¬F)']
v3_names_ftt = ['P(¬S|A,PP)','P(¬LC|S,G)','P(¬C|All,LC)','P(¬F|LC,
```

```
    C)','P(¬CA|AD,F)']
v3_names_fft = ['P(¬S|¬A,PP)','P(¬LC|¬S,G)','P(¬C|¬All,LC)','P(¬F|
    ¬LC,C)','P(¬CA|¬AD,F)']
v3_names_ftf = ['P(¬S|A,¬PP)','P(¬LC|S,¬G)','P(¬C|All,¬LC)','P(¬F|
    LC,¬C)','P(¬CA|AD,¬F)']
v3_names_fff = ['P(¬S|¬A,¬PP)','P(`LC|¬S,¬G)','P(¬C|¬All,¬LC)','P(
    ¬F|¬LC,¬C)','P(¬CA|¬AD,¬F)']
#CPT with three variables
v3_probs_ttt =[0.0,0.0,0.0,0.0,0.0] #declare vectors to store the
    probability values
v3_probs_tft =[0.0,0.0,0.0,0.0,0.0]
v3_probs_ttf =[0.0,0.0,0.0,0.0,0.0]
v3_probs_tff =[0.0,0.0,0.0,0.0,0.0]
v3_probs_ftt =[0.0,0.0,0.0,0.0,0.0]
v3_probs_fft =[0.0,0.0,0.0,0.0,0.0]
v3_probs_ftf =[0.0,0.0,0.0,0.0,0.0]
v3_probs_fff =[0.0,0.0,0.0,0.0,0.0]

cont3ttt =[0,0,0,0,0]
cont3tft =[0,0,0,0,0]
cont3ttf =[0,0,0,0,0]
cont3tff =[0,0,0,0,0]
cont3ftt =[0,0,0,0,0]
cont3fft =[0,0,0,0,0]
cont3ftf =[0,0,0,0,0]
cont3fff =[0,0,0,0,0]

cont3_varstt =[0,0,0,0,0]
cont3_varsft =[0,0,0,0,0]
cont3_varstf =[0,0,0,0,0]
cont3_varsff =[0,0,0,0,0]


with open('lucas0_train.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')
    line_count = 0
    for row in csv_reader:
         line_count += 1

         for i in range(0,len(vector1)):
         #count for 1
               if row[vector1[i]] == '1':
                  cont1[i] = cont1[i] + 1
         for i in range(0,len(vector2)):
               #count for 2
               if row[vector2[i]] == '1' and row[vector2_var[i]]
   == '1':
                 cont2tt[i] = cont2tt[i] + 1
               if row[vector2[i]] == '0' and row[vector2_var[i]]
```

```
   == '1':
                cont2ft[i] = cont2ft[i] + 1
              if row[vector2[i]] == '1' and row[vector2_var[i]]
   == '0':
                cont2tf[i] = cont2tf[i] + 1
              if row[vector2[i]] == '0' and row[vector2_var[i]]
   == '0':
                cont2ff[i] = cont2ff[i] + 1
              if row[vector2_var[i]] == '1':
                 cont2_vart[i] = cont2_vart[i] + 1
              if row[vector2_var[i]] == '0':
                 cont2_varf[i] = cont2_varf[i] + 1
        for i in range(0,len(vector3)):
              #count for 3
              if row[vector3[i]] == '1' and row[vector3_var1[i]]
   == '1' and row[vector3_var2[i]] == '1':
                 cont3ttt[i] = cont3ttt[i] + 1
              if row[vector3[i]] == '1' and row[vector3_var1[i]]
   == '0' and row[vector3_var2[i]] == '1':
                 cont3tft[i] = cont3tft[i] + 1
              if row[vector3[i]] == '1' and row[vector3_var1[i]]
   == '1' and row[vector3_var2[i]] == '0':
                 cont3ttf[i] = cont3ttf[i] + 1
              if row[vector3[i]] == '1' and row[vector3_var1[i]]
   == '0' and row[vector3_var2[i]] == '0':
                 cont3tff[i] = cont3tff[i] + 1
              if row[vector3[i]] == '0' and row[vector3_var1[i]]
   == '1' and row[vector3_var2[i]] == '1':
                 cont3ftt[i] = cont3ftt[i] + 1
              if row[vector3[i]] == '0' and row[vector3_var1[i]]
   == '0' and row[vector3_var2[i]] == '1':
                 cont3fft[i] = cont3fft[i] + 1
              if row[vector3[i]] == '0' and row[vector3_var1[i]]
   == '1' and row[vector3_var2[i]] == '0':
                 cont3ftf[i] = cont3ftf[i] + 1
              if row[vector3[i]] == '0' and row[vector3_var1[i]]
   == '0' and row[vector3_var2[i]] == '0':
                 cont3fff[i] = cont3fff[i] + 1

              if row[vector3_var1[i]] == '1' and row[vector3_var2
   [i]] == '1':
                cont3_varstt[i] = cont3_varstt[i] + 1
              if row[vector3_var1[i]] == '0' and row[vector3_var2
   [i]] == '1':
                cont3_varsft[i] = cont3_varsft[i] + 1
              if row[vector3_var1[i]] == '1' and row[vector3_var2
   [i]] == '0':
                cont3_varstf[i] = cont3_varstf[i] + 1
              if row[vector3_var1[i]] == '0' and row[vector3_var2
```

```
    [i]] == '0':
                cont3_varsff[i] = cont3_varsff[i] + 1


for i in range(0,len(vector1)):
        v1_probs[i] =  (cont1[i] + 1)/(line_count + 2)
        print(v1_names[i] + ' = ' + str(v1_probs[i]))
        v1_probs_f[i] = 1 - v1_probs[i]
        print(v1_names_f[i] + ' = ' + str(v1_probs_f[i]))

for i in range(0,len(vector2)):
        v2_probs_tt[i] =  (cont2tt[i] + 1)/(cont2_vart[i] + 2)
        print(v2_names_tt[i] + ' = ' + str(v2_probs_tt[i]))
        v2_probs_ft[i] = (cont2ft[i] + 1)/(cont2_vart[i] + 2)
        print(v2_names_ft[i] + ' = ' + str(v2_probs_ft[i]))
        v2_probs_tf[i] =  (cont2tf[i] + 1)/(cont2_varf[i] + 2)
        print(v2_names_tf[i] + ' = ' + str(v2_probs_tf[i]))
        v2_probs_ff[i] = (cont2ff[i] + 1)/(cont2_varf[i] + 2)
        print(v2_names_ff[i] + ' = ' + str(v2_probs_ff[i]))

for i in range(0,len(vector3)):
        v3_probs_ttt[i] =  (cont3ttt[i] + 1)/(cont3_varstt[i] + 2)
        print(v3_names_ttt[i] + ' = ' + str(v3_probs_ttt[i]))
        v3_probs_tft[i] =  (cont3tft[i] + 1)/(cont3_varsft[i] + 2)
        print(v3_names_tft[i] + ' = ' + str(v3_probs_tft[i]))
        v3_probs_ttf[i] =  (cont3ttf[i] + 1)/(cont3_varstf[i] + 2)
        print(v3_names_ttf[i] + ' = ' + str(v3_probs_ttf[i]))
        v3_probs_tff[i] =  (cont3tff[i] + 1)/(cont3_varsff[i] + 2)
        print(v3_names_tff[i] + ' = ' + str(v3_probs_tff[i]))
        v3_probs_ftt[i] =  (cont3ftt[i] + 1)/(cont3_varstt[i] + 2)
        print(v3_names_ftt[i] + ' = ' + str(v3_probs_ftt[i]))
        v3_probs_fft[i] =  (cont3fft[i] + 1)/(cont3_varsft[i] + 2)
        print(v3_names_fft[i] + ' = ' + str(v3_probs_fft[i]))
        v3_probs_ftf[i] =  (cont3ftf[i] + 1)/(cont3_varstf[i] + 2)
        print(v3_names_ftf[i] + ' = ' + str(v3_probs_ftf[i]))
        v3_probs_fff[i] =  (cont3fff[i] + 1)/(cont3_varsff[i] + 2)
        print(v3_names_fff[i] + ' = ' + str(v3_probs_fff[i]))



print('Processed ' + str(line_count) + ' lines.')
#print(v1_probs)
```

Listing 2: readcsv.py

```
import sys
import random
from readcsv import *
```

```
print(v2_probs_tt)
class PriorSampling:
  CPTs={} #dictionary

  def __init__(self, netID):
    self.initialiseNet(netID)

  def initialiseNet(self, netID):
    #print(v1_probs)
    #print(v1_probs_f)

    if netID == "luca":
      self.CPTs["P"]={"+p":v1_probs[0], "-p":v1_probs_f[0]} #
    peerpresssure
      self.CPTs["A"]={"+a":v1_probs[1], "-a":v1_probs_f[1]} #
    anxiety  #keys = {values}

      self.CPTs["S"]={"+s|+a+p":v3_probs_ttt[0], "-s|+a+p":
    v3_probs_ftt[0], #smoking|anxiety,pp
          "+s|-a+p":v3_probs_tft[0], "-s|-a+p":v3_probs_fft[0],
          "+s|+a-p":v3_probs_ttf[0], "-s|+a-p":v3_probs_ftf[0],
          "+s|-a-p":v3_probs_tff[0], "-s|-a-p":v3_probs_fff[0]}
      self.CPTs["Y"]={"+y|+s":v2_probs_tt[0], "-y|+s":v2_probs_ft
    [0],
          "+y|-s":v2_probs_tf[0], "-y|-s":v2_probs_ff[0]}
      self.CPTs["G"]={"+g":v1_probs[2], "-g":v1_probs_f[2]}
      self.CPTs["LC"]={"+lc|+s+g":v3_probs_ttt[1], "-lc|+s+g":
    v3_probs_ftt[1],
          "+lc|-s+g":v3_probs_tft[1], "-lc|-s+g":v3_probs_fft[1],
          "+lc|+s-g":v3_probs_ttf[1], "-lc|+s-g":v3_probs_ftf[1],
          "+lc|-s-g":v3_probs_tff[1], "-lc|-s-g":v3_probs_fff[1]}
      self.CPTs["AD"]={"+ad|+g":v2_probs_tt[1], "-ad|+g":
    v2_probs_tt[1],
          "+ad|-g":v2_probs_tf[1], "-ad|-g":v2_probs_ff[1]}
      self.CPTs["ED"]={"+ed":v1_probs[3], "-ed":v1_probs_f[3]}
      self.CPTs["All"]={"+all":v1_probs[4], "-all":v1_probs_f[4]}
      self.CPTs["C"]={"+c|+all+lc":v3_probs_ttt[2], "-c|+all+lc":
    v3_probs_ftt[2],
          "+c|+all-lc":v3_probs_tft[2], "-c|+all-lc":v3_probs_fft
    [2],
          "+c|-all+lc":v3_probs_ttf[2], "-c|-all+lc":v3_probs_ftf
    [2],
          "+c|-all-lc":v3_probs_tff[2], "-c|-all-lc":v3_probs_fff
    [2]}
      self.CPTs["F"]={"+f|+lc+c":v3_probs_ttt[3], "-f|+lc+c":
    v3_probs_ftt[3],
          "+f|+lc-c":v3_probs_tft[3], "-f|+lc-c":v3_probs_fft[3],
          "+f|-lc+c":v3_probs_ttf[3], "-f|-lc+c":v3_probs_ftf[3],
```

```
          "+f|-lc-c":v3_probs_tff[3], "-f|-lc-c":v3_probs_fff[3]}
      self.CPTs["CA"]={"+ca|+ad+f":v3_probs_ttt[4], "-ca|+ad+f":
    v3_probs_ftt[4],
           "+ca|+ad-f":v3_probs_tft[4], "-ca|+ad-f":v3_probs_fft
    [4],
           "+ca|-ad+f":v3_probs_ttf[4], "-ca|-ad+f":v3_probs_ftf
    [4],
           "+ca|-ad-f":v3_probs_tff[4], "-ca|-ad-f":v3_probs_fff
    [4]}
      self.CPTs["order"]=["P","A", "S", "Y", "G", "LC", "AD", "ED"
    , "All", "C", "F","CA"]
      self.CPTs["parents"]={"P":None,"A":None,  "S":"A,P", "Y":"S"
    ,"G":None, "LC":"S,G","AD":"G","ED":None,"All":None,"C":"All,
    LC","F":"LC,C","CA":"AD,F"}


    else:
      print("UNKNOWN network="+str(netID))
      exit(0)

  def sampleVariable(self, CPT, conditional):
    sampledValue=None
    randnumber=random.random()

    value1=CPT["+"+conditional]
    value2=CPT["-"+conditional]

    if randnumber<=value1:
      sampledValue="+"+conditional
    else:
      sampledValue="-"+conditional

    return sampledValue.split("|")[0]

  def sampleVariables(self, printEvent):
    event=[]
    sampledVars={}

    for variable in self.CPTs["order"]:
      evidence=""
      conditional=""
      parents=self.CPTs["parents"][variable]
      if parents==None:
        conditional=variable.lower()
      else:
        for parent in parents.split(","):
          evidence+=sampledVars[parent]
        conditional=variable.lower()+"|"+evidence
```

```
      sampledValue=self.sampleVariable(self.CPTs[variable],
    conditional)
      event.append(sampledValue)
      sampledVars[variable]=sampledValue

    if printEvent: print(event)
    return event

if __name__ == "__main__":
  ps=PriorSampling("luca")
  ps.sampleVariables(True)# this method will give you an event
```

Listing 3: sampling_parameters.py

```
import sys
import random

from sampling_parameters import PriorSampling

count={}
#ps = PriorSampling("sprinkler")
ps = PriorSampling("luca")

#comentar
def RejectionSampling(X,e,N):

  index = getIndex(X)
  for j in range(1, N):

    x = ps.sampleVariables(False)
    if isConsistent(x,e):
      val=x[index]
      if val in count:
        count[val]+=1
      else:
        count[val]=1
  #print(count)
  return count


    #comentar
def isConsistent(event,evidence):
  if evidence =='': return True

  for varvalue in evidence:
    if varvalue not in event:
      return False
  #print("se cumple")
  return True
```

```
#comentar
def getIndex(X):
  variables = ps.CPTs["order"]
  for i in range(0,len(variables)):
    if variables[i] == X:
      return i
#comentar
def normalize(x,y):
  aux = x+y
  alpha = 1.0/aux
  nx = x*alpha
  ny = y*alpha
  return nx,ny


if __name__ == "__main__":
  e ={"+c","+f"} #given coughing and fatigue
  X = "S" #smoking
  N=10000
  count = RejectionSampling(X,e,N)
  print(count)
  #cosa = str(ps.CPTs[X].keys())

  #comentar
  x=count.pop("+s")
  y=count.pop("-s")
  xnorm, ynorm = normalize(x,y)
  print("probability distribution: <" + str(xnorm) + "," + str(
    ynorm) + ">" )
```

Listing 4: luca_inference.py

### A.3 Task2

```
#use always python3

import sys
import random
import numpy
from decimal import Decimal
```

```
def parse_arg():
  input_valid = False
  while input_valid == False:
    input_valid = True
    raw_in = input('Type in your sequence using a "," to separate:
    ')

    sep = raw_in.split(",")
    for i in range(1,len(sep)+1):

      e = sep[i-1]
      #print(e)
      if e == 'hot':
        e_idx=0
      elif e =='warm':
        e_idx=1
      elif e =='cold':
        e_idx=2
      else:
        print('the sequence was not valid, try again')
        input_valid = False

  return sep


if __name__ == "__main__":


  p_same = 0.7
  p_change = 0.3

  hot_on = 0.4
  warm_on = 0.4
  cold_on = 0.2

  hot_off = 0.1
  warm_off = 0.45
  cold_off = 0.45

  state_on = [hot_on,warm_on,cold_on]
  state_off = [hot_off,warm_off,cold_off]

  #cold,warm,hot,warm,cold

  #raw_in = input('Type in your sequence using a "," to separate
    :')

  sep = parse_arg()
  #print(sep)
```

```
p_on0 = 0.5
p_off0 = 0.5
#step1 cold
p_on = [0]*(len(sep)+1)
p_off = [0]*(len(sep)+1)

p_on[0]=0.5
p_off[0]=0.5

#print(sep[0])


for i in range(1,len(sep)+1):

  e = sep[i-1]
  print(e)
  if e == 'hot':
    e_idx=0
  elif e =='warm':
    e_idx=1
  elif e =='cold':
    e_idx=2
  else:
    print('the sequence was incorrect')


  p_on[i] = state_on[e_idx]*((p_on[i-1]*p_same) + (p_off[i-1]*
  p_change))
  print(p_on[i])
  p_off[i] = state_off[e_idx]*((p_on[i-1]*p_change) + (p_off[i
  -1]*p_same))
  print(p_off[i])

  #print('P(x):' , p_on[i])#probability of having the sequence
  given the heater is on at the last observation
  #print('P(¬x):', p_off[i])#probability of having the sequence
  given the heater is off at the last observation

tot = p_on[len(sep)]+p_off[len(sep)]
print('Total probability:' ,tot)#probability of having the
  sequence
```

Listing 5: markov.py