UNIVERSITY OF
LINCOLN

Advanced Artificial Intelligence

CMP9132M | Assessment Item 1

Peter Hart | 12421031

The following report will entail an appraisal and analysis of a range of artificial intelligence techniques and consideration for the application of effective decision-making, problem solving and learning for solving a given issue. The solutions that are being described for each task have been developed in the programming language 'MATLAB' and can be located within the appendices of this report.

Task 1.A

The first task of this assignment involves the 'rare disease and test problem' scenario where the system environment being considered situates a total of two variables, 'd' – the person has the disease and 't' – the test is positive. The problem at hand consists of a computation for the probability that the person has the disease given the test was positive, the variable equivalent representation being: '$P(d|t)$'. Several probability variables are known and are assigned an initial probability distribution value at run-time by the user, these include '$P(d)$': the prior probability of having the disease, '$P(t|d)$': the probability that the test is positive given the person has the disease and '$P(\neg t|\neg d)$': probability that the test is negative given the person does not have the disease.

One issue presented with the task at hand is that it is difficult to calculate the probability of '$P(A|B)$' directly due to no or limited information, however more information could be grasped by applying the Bayes' Rule which enables the probability of '$P(A|B)$' to be rearranged and computed in terms of '$P(B|A)$'. In essence, the Bayes' Rule entails calculating the probability of '$P(A|B)$' by multiplying the likelihood of '$P(B|A)$' by the prior probability of '$P(A)$' divided by the likelihood that '$P(B)$' will occur. By substituting the probabilities involved in this system, the Bayes' Rule is evaluated as shown in the formula below:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad P(d|t) = \frac{P(t|d)P(d)}{P(t)}$$

Figure 1: Bayes' Rule formula.

However, this in turn presents another issue, this being the lack of information regarding the probability of '$P(t)$'. The outcome of this is that it obstructs the possibility of being able to utilise Bayes' Rule, as it is currently impossible to substitute a value for the prior probability '$P(A)$' and therefore incapable of completing the Bayes' Rule substitution. Therefore, a critical objective for this task was to estimate the probability distribution of '$P(t)$' by inferring information from the known probabilities within the system. Overall, the probability of '$P(t)$' can be found by evaluating both the positive and negative cases of '$P(d)$' and '$P(t|d)$', which were provided as input variables by the user. Assuming that these variables contain normalised values, the sum of the positive and negative cases of both variables should sum to one. Therefore, the negative cases of these variables can be established by subtracting the values of the positive cases away from 1.

$$P(\neg d) = 1 - P(d) \qquad P(t|\neg d) = 1 - P(\neg t|\neg d)$$

Figure 2: Equations used for estimating the probability values of P(¬d) and P(t|¬d).

As a result, these estimated negative variables can be substituted back into the previously discussed Bayes' Rule, where the equation can be rearranged to estimate the value of '$P(t)$' by eliminating '$P(d)$' from the equation.

$$P(t) = \frac{P(t|d)P(d)}{\cancel{P(t)}} + \frac{P(t|\neg d)P(\neg d)}{\cancel{P(t)}}$$

$$P(t) = P(t|d)P(d) \; + \; P(t|\neg d)P(\neg d)$$

Figure 3: Equation for finding P(t).

As a consequence, the objective of estimating a probability value for the variable '$P(t)$' has been completed, therefore this output value can then be input back into the original Bayes' Rule equation to estimate a final value which is indicative of the true value for the probability variable '$P(d|t)$'. Overall, the solution for this task has been adapted such that the program will prompt the user to input the initial probability values for '$P(d)$', '$P(t|d)$' and '$P(\neg t|\neg d)$' on each iteration that the program runs, so a different result can be easily generated on each iteration if desired.

Task 1.B

This next task involved a burglary problem which was provided in the form of a Bayesian network which features a series of events which could potentially occur in the system. The system in question features a total of five variables: '*B*' – there is a burglary, '*E*' – there is an earthquake, '*A*' – the alarm went off, '*J*' – John called and '*M*' – Mary called. However, the conditional probability table for each variable is unknown and therefore the direct probability for each possible value within the domain size of each variable is unknown.

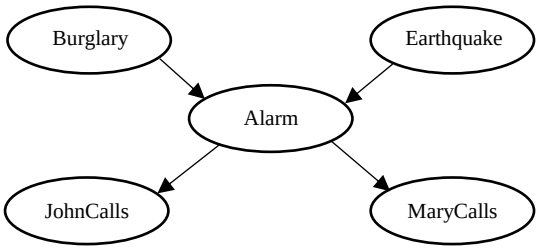

Figure 4: Bayesian Network structure.

Alarm	Mary	John	Burglary	Earthquake
T	T	T	T	F
F	F	F	F	F
T	T	F	F	T
F	F	T	F	F
F	F	F	F	F
F	F	F	F	F
T	T	T	T	F
T	T	T	T	F
F	F	F	F	T
T	T	T	F	T

Figure 5: Input dataset of variable values.

As a result, the first objective to be handled is acquiring an appropriate conditional probability table which can be argued to accurately describe the probability distributions involved in each variable for all possible cases. To accomplish this, a technique called parameter learning can be used to learn about the data distributions within an input dataset for each variable involved in the Bayesian Network.

Specifically, this was achieved by employing the maximum likelihood estimation (MLE) algorithm. The function for calculating the maximum likelihood estimation for one variable operates by iterating through the all of the data observations contained within the input dataset and counting the instances that each possible unique value appears in each variable. For example, the variable entitled '*B*' or '*Burglary*' has a domain size of two possible unique values: '*true*' or '*false*'. This count value can then be input into the numerator of the maximum likelihood estimation formula; thus the estimated probability value would be calculated by total count of a value divided by the number of observations within the dataset with the addition of the domain size of the variable. The domain size is indicative of the total number of unique values that are contained in a variable. This can be observed in figure 6.

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

Figure 6: One variable MLE.

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

Figure 7: Two variable MLE.

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

Figure 8: Three variable MLE.

However, currently the one-variable version of maximum likelihood estimation can only be applied to a small number of variables involved with this Bayesian network. Instead, two-variable and three-variable variations of maximum likelihood estimation were also established to ensure an exhaustive array of conditional probabilities relative to the Bayesian network. In contrast to the previously explored one-variable variation of the algorithm, the count operations are performed such that the numerator of both equations count instances where all variables involved match a specified value. Similarly, the count operations in the denominator are performed by only counting instances where the evidence variables of the conditional probability match a specified value. By utilising these three versions of the parameter learning algorithm, it is possible to estimate the probability values for all of the variables that will be considered as parameters for this particular Bayesian network which can subsequently be input into the next phase of the system architecture: the inference algorithm.

The inference algorithm stage of the system involves the system capacity for deriving new uncertain probabilities based on the known probabilities. As such, the methodology selected to be implemented in the system being developed was the methodology entitled 'rejection sampling'. This operates by first generating a sample for a possible prior probability of the variable, which is calculated by randomly selecting a number and comparing it to the probabilistic values that were previously generated for each variable involved in this Bayesian network through parameter learning. For example, the prior sample function could return a randomly selected value that could exist with the domain size of the '*Mary called*' variable, such as '$+m$' or '$-m$'. The outcome of this is then sent to the main rejection sampling function, whereby the sample generated is compared and checked to see if it is consistent with the conditional probability that is being queried by the user. In the case of the variable $P(b|j,m)$, the function would first compare the evidence variables '$+m$' and '$+j$' and check if they are consistent with the prior sample that was randomly generated. If it is found that the evidence of the sample generated is consistent with the conditional probability in question, then the rejection sampling algorithm counts the value contained within the query variable with respect to each possible value within the domain size of the variable, otherwise this sample is rejected and discarded. This algorithm continues to iterate with different random prior samples being generated in each iteration until the number of samples generated reach a total sample limit value of 'N', thus a total count for the number of cases where the conditional probabilities $P(b|j,m)$ and $P(\neg b|j,m)$ are true should be found.

The final objective of the inference phase of the system is to normalise the conditional probability value based on the total counts observed in all cases of the query variable. A normalised probability refers to ensuring that the estimated probability values for all cases of a variable will sum to '1', which otherwise implies that the final calculated probability distribution is an accurate representation for the conditional probability value. The normalised probability is calculated by one divided by the sum of the counts for all of the unique values of the conditional variable multiplied by the individual count value for the conditional probability being estimated. This is represented in the formula below:

$$Normalise(n1) = \frac{1}{(count(n1) + count(n2))} * count(n1)$$

Figure 9: Normalisation formula used for ensuring probabilities will sum to 1.

However, one alternative approximate inference methodology that could be implemented is likelihood weighting. Unlike rejection sampling, likelihood weighting does not discard any generated samples and in turn can provide a more efficient solution in comparison to rejection sampling. However, a weakness of the likelihood weighting methodology is that it will suffer an exponential degradation in performance as the number of evidence variables increase. Russel and Norvig (2003) argue that most samples contain low weights and therefore the weighted estimate value will be skewed by the tiny fraction of samples to congregate more than an extremely small likelihood to the evidence. Therefore, if the existing

Bayesian network was upgraded and contained a larger array of evidence variables, then the likelihood weighting methodology would cause an accuracy degradation for the inference phase of the system.

Task 2

A Markov model can be considered as a stochastic, statistical model which can be utilised for modelling randomly changing systems. For example, the Markov Chain model models the system state with a random variable which is assumed to change as time passes, where it is assumed that the probability distribution for the system only adheres to a dependency on the probability distribution observed in the previous state. As such, Markov Chain models lack the capacity to predict new probabilities if there are hidden states involved within the system. In contrast, a Hidden Markov Model operates similar to a Markov Chain model but for a system where the observations are partially observable and contain hidden states, such as the hidden states "ON" and "OFF". The Hidden Markov Model operates under the belief that each state in the system has a probability distribution relative to the possible output values, and therefore the sequence of output values can be used to determine more information about the hidden state sequence.

To embody this concept into the design of the solution that was being developed, the system required more information about the emission probability distribution at each new state within the sequence of states. This operates under a recursive loop which constantly requests for new user input during each iterations, whereby the user is prompted to input an appropriate sequence of symbols from the vocabulary defined within the system, these being either: "Warm", "Cold", "Hot", "Freezing". The input received from the user will refer to a series of possible emission probability distributions that are known and pre-set in the system.

	On	**Off**
Remain	0.7500	0.7500
Switch	0.2500	0.2500

Figure 10: Transition distribution.

	Warm	**Cold**	**Hot**	**Freezing**
On	0.4500	0.4500	0.0500	0.0500
Off	0.0500	0.0500	0.4500	0.4500

Figure 11: Emission distribution.

The emission probability distributions describe the raw probabilities for each emission symbol being output. In contrast, the transition probability distributions describe the raw probabilities of what the next state could be given the values that reside in the current state. However, as previously discussed, this system features hidden states which describe the operational status of the heater, these being either "ON" or "OFF". Imperatively, the Hidden Markov model can be used to estimate the values of the hidden states through time as per sequence specified by the user, based on the output probability distribution of the previous states.

The developed algorithm in question operates similar to a forward-filtering algorithm, which can be classified as a recursive algorithm which recurs until an entire sequence of observations have been considered.

$$f_t = \alpha\, O_t\, T^T f_{t-1}$$

Figure 12: Forward-Filtering Algorithm Formula

$$f_t = O_t\, T^T f_{t-1}$$

Figure 13: Formula used within this HMM.

To surmise, this algorithm operates by checking which event was selected by the user from the defined system vocabulary, where the probability of the event selected 'O_t' will be multiplied by the transpose of the transition probability 'T^T' and finally multiplied by the output distribution of the previous state 'f_{t-1}' to estimate the probability distribution of the hidden state 'f_t'. However, unlike the formula for

a forward-filtering algorithm, the developed algorithm does not require a normalisation factor 'α' as it can be assumed that the output contains normalised probability values which sum to '1'.

For example, for the following sequence of events: '*hot-freezing-cold-warm*', a recursive function consistently requests for new input from the user which can be used to select which event occurs at each state of the algorithm. As previously discussed, this Hidden Markov model assumes first-order Markov assumption and only considers the probability values that are contained within the previous state of the system, however the first state of the sequence '*S1*' assumes a starting state equiprobability distribution when referencing the previous state '*S0*'. As a result, the probability values of the hidden state '*S1*' would be calculated by multiplying the emission probability of '*hot*' with the transpose of the transition probability matrix multiplied by the output distribution of the previous state, which in this case would be '*S0*' and therefore the starting state matrix containing equiprobable values: [0.500, 0.500]. Overall, the probability of '*S1*' can be estimated to be [0.0250, 0.2250], which is then stored when updating variable '*s0*' with the new predicted values of '*S1*' to ensure first-order Markov assumption for the subsequent state in this sequence of events.

Therefore, this recursive algorithm can continue and request for more event input from the user for state '*S2*', where in the case of this example sequence would entail having the event '*freezing*' selected by the user. Therefore, the probability of hidden state '*S2*' can be estimated by multiplying the emission probability of '*freezing*' with the transpose of the transition probability multiplied by the previous state probability. However, unlike the previous state '*S1*', the probability estimation for state '*S2*' will not be referencing the initial starting state containing assumed equiprobable values, but instead referencing the updated probability values of the previous state which should now contain the previously predicted values of state '*S1*', in this case: [0.0250, 0.2250]. Thus, the probability of hidden state '*S2*' can be estimated to be: [0.0038, 0.0788].

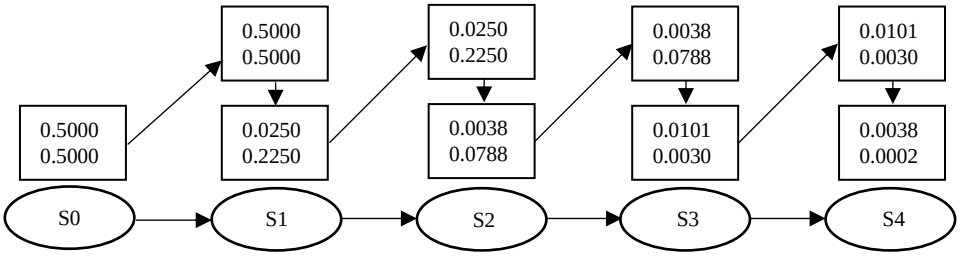

Figure 14: Demonstration of recursive algorithm being applied when updating/predicting hidden states.

This recursive algorithm will continue to operate under this procedure until a desirable number of events have been entered into the sequence, which in the case of this example will be until the fourth event '*warm*' has been considered by the algorithm. Thus, the final operation of this algorithm enforces a sum of the probability values that are contained in the final state, providing a final probability estimate value which describes the likelihood of observing the input sequence of events. In the case of this example, the final probability of the sequence occurring can be calculated by 0.0038 + 0.0002 which equates to 0.0039938.

Reference List

Russel, S.J. and Norvig, P. (2003) *Artificial Intelligence: A Modern Approach,* 2nd edition. New Jersey: Pearson Education Incorporated.

Appendix

Appendix A: MATLAB Solution Script for Task 1.A

```
% Task 1a Script | Advanced Artificial Intelligence | CMP9132M
% 12421031 | Peter Hart

%Clear workspace variables and command window.
clear;clc;

% Variables involved in this solution can be defined as:
% d = the person has the disease.
% t = the test is positive.

% The solution gathers user input for the following three variables:
% P(d)
prompt = 'What is P(d)? ';
Pd = input(prompt);

% P(t|d)
prompt = 'What is P(t|d)? ';
Ptd = input(prompt);

% P(¬t|¬d)
prompt = 'What is P(¬t|¬d)? ';
P_t_d = input(prompt);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Bayes' Rule = P(A|B) = P(B|A)*P(A)/P(B)
%             P(d|t) = P(t|d)*P(d)/P(t)

%P(¬d)
P_d = 1 - Pd;

% Output the Results:
disp('P(¬d) = 1 - P(d)');
UI = ['P(¬d) = 1 - ', num2str(Pd)];
disp(UI);
UI = ['P(¬d) = ', num2str(P_d)];
disp(UI);

%P(¬t|d)
Pt_d = 1 - P_t_d;

% Output the Results:
disp('P(t|¬d) = 1 - P(¬t|¬d)');
UI = ['P(t|¬d) = 1 - ', num2str(P_t_d)];
disp(UI);
UI = ['P(t|¬d) = ', num2str(Pt_d)];
disp(UI);

%P(t)
Pt = (Ptd*Pd)+(Pt_d*P_d);

% Output the Results:
disp('P(t) = (P(t|d) * P(d)) + (P(t|¬d) * P(¬d))');
UI = ['P(t) = (', num2str(Ptd), ' * ', num2str(Pd), ') + (', num2str(Pt_d),
' * ', num2str(P_d), ')'];
disp(UI);
```

```
UI = ['P(t) = ', num2str(Ptd*Pd), ' + ', num2str(Pt_d*P_d)];
disp(UI);
UI = ['P(t) = ', num2str(Pt)];
disp(UI);

%P(d|t)
Pdt = (Ptd*Pd) / Pt;

% Output the Results:
disp('P(d|t) = (P(t|d)*P(d))/P(t)');
UI = ['P(d|t) = (',num2str(Ptd), ' * ', num2str(Pd),') / ', num2str(Pt)];
disp(UI);
UI = ['P(d|t) = ',num2str(Pdt)];
disp(UI);

%end of script
```

Appendix B: MATLAB Solution Script for Task 1.B

```
% Task 1b Script | Advanced Artificial Intelligence | CMP9132M
% 12421031 | Peter Hart

% Clear the workspace and command window.
clear;clc;

% Input array:
%        A,     M,     J,     B,     E;
input = [true,  true,  true,  true,  false;
         false, false, false, false, false;
         true,  true,  false, false, true;
         false, false, true,  false, false;
         false, false, false, false, false;
         false, false, false, false, false;
         true,  true,  true,  true,  false;
         true,  true,  true,  true,  false;
         false, false, false, false, true;
         true,  true,  true,  false, true];

%Variables storing dimensions of input array
totalRow = size(input,1);
totalCol = size(input,2);

%Store column values for each variable to simplify referencing
colA = 1; colM = 2; colJ = 3; colB = 4; colE = 5;

% Conditional Probability Tables (CPT)
% Values for each prior and conditional probability have been
% calculated using Parameter Learning. (where 1 = true, 0 = false)

%P(+b)
Pb = varOneMLE(input,colB,1);
%P(-b)
P_b = varOneMLE(input,colB,0);

%P(+e)
Pe = varOneMLE(input,colE,1);
%P(-e)
P_e = varOneMLE(input,colE,0);

%P(+j|+a)
Pja = varTwoMLE(input,colJ,colA,1,1);
%P(-j|+a)
P_ja = varTwoMLE(input,colJ,colA,0,1);
%P(+j|-a)
Pj_a = varTwoMLE(input,colJ,colA,1,0);
%P(-j|-a)
P_j_a = varTwoMLE(input,colJ,colA,0,0);

%P(+m|+a)
Pma = varTwoMLE(input,colM,colA,1,1);
%P(-m|+a)
P_ma = varTwoMLE(input,colM,colA,0,1);
%P(+m|-a)
Pm_a = varTwoMLE(input,colM,colA,1,0);
%P(-m|-a)
P_m_a = varTwoMLE(input,colM,colA,0,0);
```

```
%P(+a|+b,+e)
Pabe = varThreeMLE(input,colA,colB,colE,1,1,1);
%P(-a|+b,+e)
P_abe = varThreeMLE(input,colA,colB,colE,0,1,1);
%P(+a|+b,-e)
Pab_e = varThreeMLE(input,colA,colB,colE,1,1,0);
%P(-a|+b,-e)
P_ab_e = varThreeMLE(input,colA,colB,colE,0,1,0);
%P(+a|-b,+e)
Pa_be = varThreeMLE(input,colA,colB,colE,1,0,1);
%P(-a|-b,+e)
P_a_be = varThreeMLE(input,colA,colB,colE,0,0,1);
%P(+a|-b,-e)
Pa_b_e = varThreeMLE(input,colA,colB,colE,1,0,0);
%P(-a|-b,-e)
P_a_b_e = varThreeMLE(input,colA,colB,colE,0,0,0);

% Data Dictionary:
%Store values from the conditional probability table into the data
%dictionary. Values for each conditional probability will be mapped to a
%key
keySetB = {'+b','-b'};
valueSetB = [Pb,P_b];
varB = containers.Map(keySetB,valueSetB);

keySetE = {'+e','-e'};
valueSetE = [Pe,P_e];
varE = containers.Map(keySetE,valueSetE);

keySetA = {'+a|+b+e', '+a|+b-e','+a|-b+e','+a|-b-e','-a|+b+e','-a|+b-e','-
a|-b+e','-a|-b-e'};
valueSetA = [Pabe, Pab_e, Pa_be, Pa_b_e, P_abe, P_ab_e, P_a_be, P_a_b_e];
varA = containers.Map(keySetA,valueSetA);

keySetJ = {'+j|+a','+j|-a','-j|+a','-j|-a'};
valueSetJ = [Pja,Pj_a,P_ja,P_j_a];
varJ = containers.Map(keySetJ,valueSetJ);

keySetM = {'+m|+a','+m|-a','-m|+a','-m|-a'};
valueSetM = [Pma, Pm_a, P_ma, P_m_a];
varM = containers.Map(keySetM,valueSetM);

%sampleSize = total amount of samples to be generated for rejection
%sampling
sampleSize = 100000;
Pbjm = rejectionSampling(varB,varE,varA,varM,varJ,sampleSize);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Output the results:
%P(b|j,m)
UI = ['P(+b|+j,+m) = ', num2str(Pbjm(1))];
disp(UI);

%P(-b|j,m)
UI = ['P(-b|+j,+m) = ', num2str(Pbjm(2))];
disp(UI);

function P = rejectionSampling(varB,varE,varA,varM,varJ,N)
```

```
    %Variable for counting positive cases where the sample is consistent
with
    %the evidence
    n1 = 0;
    %Variable for counting negative cases where the sample is consistent
with
    %the evidence
    n2 = 0;

    %for each sample that will be generated
    for i=1:N
        %Obtain a random sample for P(b)
        sampledValueB = priorSample(varB, 'b');
        %Obtain a random sample for P(e)
        sampledValueE = priorSample(varE, 'e');

        %Obtain a random sample for P(a|b,e), using prior probabilities
        %generated
        conditionalA = strcat('a|',sampledValueB,sampledValueE);
        sampledValueA = priorSample(varA, conditionalA);
        %Only store the randomly generated 'a' value. (+a or -a)
        sampledValueA_Q = extractBefore(sampledValueA,'|');

        %Obtain a random sample for P(j|a), using the previous conditional
        %probability that was randomly generated.
        conditionalJ = strcat('j|',sampledValueA_Q);
        sampledValueJ = priorSample(varJ, conditionalJ);
        %Only store the randomly generated 'j' value. (+j or -j)
        sampledValueJ_Q = extractBefore(sampledValueJ,'|');

        %Obtain a random sample for P(m|a), using the previous conditional
        %probability that was randomly generated.
        conditionalM = strcat('m|',sampledValueA_Q);
        sampledValueM = priorSample(varM, conditionalM);
        %Only store the randomly generated 'm' value. (+m or -m)
        sampledValueM_Q = extractBefore(sampledValueM,'|');

        %Check to see if the values generated in the sample are consistent
with
        %the conditional probability being estimated.

        %First check if the evidence variables are consistent.
        %For example: a sample containing -j, +m != P(b|j,m), and therefore
        %would be rejected.
        x1 = isConsistent('+j', sampledValueJ_Q);
        x2 = isConsistent('+m', sampledValueM_Q);
        if x1 > 0 && x2 > 0
           %If evidence variables are consistent, check the value of the
query
           %variable and add one count for which case it belongs to.
           x3 = isConsistent('+b',sampledValueB);

           if x3 > 0
               %If sample contains a positive case, add one to this
counter.
               n1 = n1 + 1;
           else
               %Otherwise add one to the counter for the negative case.
```

```
                n2 = n2 + 1;
            end
        end
    end

    %Once all samples have been considered, normalise the count values into
    %a probability.
    Pbjm = normalise(n1,n2);
    P_bjm = normalise(n2,n1);

    %Check to see if the normalised values sum to 1.
    if (Pbjm + P_bjm) ~= 1
        disp('Error! Values did not normalise correctly!');
    end

    %Final estimated probability output.
    P = [Pbjm,P_bjm];
end

%Normalisation Function
function n = normalise(count1,count2)
    n = 1 / (count1+count2) * count1;
end

%Sample Check Function
function x = isConsistent(e,sample)
    x = 0;
    %If the value in the sample is equal to the queried value
    if e == sample
        %Return with 1, indicating a match was found.
        x = x + 1;
    else
        %else return with 0, indicating a match was not found.
        x = 0;
    end
end

%Prior Sample Function
function sampledVariable = priorSample(var, value)
    % Generate a random number between 0 and 1.
    randnum = rand();

    %Acquire conditional probability value by referencing the key of the
    %respective variable in the data dictionary.
    value1 = var(strcat('+',value));

    %If the randomly generated value is lower than the probability stored
    %in the data dictionary, then:
    if randnum<value1
        %Generate this as a positive sample.
        sampledVariable = strcat('+',value);
    else
        %Generate this as a negative sample.
        sampledVariable = strcat('-',value);
    end
end

% Parameter Learning Functions: Maximum Likelihood Estimation (MLE)
% One Variable MLE Function
```

```
function P = varOneMLE (X,colx,valx)
    %Variable for counting number of instances a value appears (e.g. T/F)
    x1 = 0;

    %Calculate number of observations which will be considered
    N = size(X,1);
    %For all of the observations in the input dataset
    for i=1:N
        %If this value is equal to value that is being counted
        if X(i,colx) == valx
            %Add one to the counter variable
            x1 = x1 + 1;
        end
    end
    %Once all instances have been counted, conduct MLE calculation
    %P(X = x) = count(x) + 1 / count(X) + |X|
    %(where |X| = domain size of variable X. Domain size = number of
possible values for this variable.)
    P = (x1 + 1) / (N + 2);
end

%Two Variable MLE Function
function P = varTwoMLE (X,colx,coly,valx,valy)
    %Variables for counting number of instances a value appears (e.g. T/F)
    %x1 = x|y
    x1 = 0;
    %x2 = y
    x2 = 0;

    %N = total observations
    N = size(X,1);
    %For all of the observations in the input dataset
    for i=1:N
        %If x AND y match their respective values
        if X(i,colx) == valx && X(i,coly) == valy
            %Add 1 to the counter
            x1 = x1 + 1;
        end
        %If y matches its respective value
        if X(i,coly) == valy
            %Add 1 to the counter
            x2 = x2 + 1;
        end
    end
    %Once all instances have been counted, conduct MLE calculation
    %P(x|y) = count(x|y)+1 / count(y)+|X|
    %(where |X| = domain size of variable X. Domain size = number of
possible values for this variable.)
    P = (x1 + 1) / (x2 + 2);
end

%Three Variable MLE Function
function P = varThreeMLE (X,colx,coly,colz,valx,valy,valz)
    %Variables for counting number of instances a value appears (e.g. T/F)
    %x1 = x|y,z
    x1 = 0;
    %x2 = y,z
    x2 = 0;

    %N = total observations.
```

```
    N = size(X,1);
    %For all of the observations in the input dataset
    for i=1:N
        %If z AND y AND z match their respective values
        if X(i,colx) == valx && X(i,coly) == valy && X(i,colz) == valz
            %Add 1 to this counter
            x1 = x1 + 1;
        end
        %If y AND z match to their respective values
        if X(i,coly) == valy && X(i,colz) == valz
            %Add 1 to this counter
            x2 = x2 + 1;
        end
    end
    %Once all instances have been counted, conduct MLE calculation
    %P(x|y,z) = count(x|y,z)+1 / count(y,z)+|X|
    %(where |X| = domain size of variable X. Domain size = number of
possible values for this variable.)
    P = (x1 + 1) / (x2 + 2);
end

%end of script
```

Appendix C: MATLAB Solution Script for Task 2

```
% Task 2 Script | Advanced Artificial Intelligence | CMP9132M
% 12421031 | Peter Hart

%Clear workspace variables
clear;

% Transition Probabilities
% (ON, OFF)
T = [0.75,0.25;
     0.25,0.75];

% Emission Probabilities
% (Warm, Cold, Hot, Freezing)
Owarm = [0.45,0.0;
         0.0,0.05];
Ocold = [0.45,0.0;
         0.0,0.05];
Ohot = [0.05,0.0;
        0.0,0.45];
Ofreezing = [0.05,0.0;
             0.0,0.45];

%Starting state probability.
s0 = [0.5;
      0.5];
%Initialise current state.
st = s0;
%Variable used for controlling recursive loop.
UIcontinue = 1;
%Counter variable that will be used for tracking current state value.
ctr =  1;

%While the user wants to input more values...
while (UIcontinue == 1)
    %Clear command window.
    clc;
    %Display probability values of previous state.
    UI = ['s',num2str(ctr-1), ' = '];
    disp(UI);
    disp(st);
    %Gather user input for current state selection
    UI = ['What is the value of s', num2str(ctr), ' ? (warm, cold, hot,
freezing)'];
    disp(UI);
    prompt = ' ';
    %Store user input in variable i. Convert string to lowercase.
    i = lower(input(prompt,'s'));


    % Compute current state probabilities depending on which emission
    % probability is selected by the user.
    % st = Ot * T' * st-1
    if (strcmp(i,'warm') == 1)
        st = Owarm * T' * s0;
    elseif (strcmp(i,'cold') == 1)
        st = Ocold * T' * s0;
    elseif (strcmp(i,'hot') == 1)
        st = Ohot * T' * s0;
```

```
    elseif (strcmp(i,'freezing') == 1)
        st = Ofreezing * T' * s0;
    else
        %If user has made a mistake, ask for more input.
        disp('Invalid input.');
        continue;
    end

    %Computation successful. Display the computed probability results.
    disp(st);
    totalProb = st(1) + st(2);
    UI = ['s', num2str(ctr), ' Total Probability = ', num2str(totalProb), '
'];
    disp(UI);

    %Ask user if they wish to exit the program.
    disp('Would you like to continue? (Y/N)');
    prompt = ' ';
    %Store user input in variable i. Convert string to lowercase.
    i = lower(input(prompt,'s'));

    if strcmp(i,'y') == 1
        UIcontinue = 1;
        %Increase current state counter.
        ctr = ctr + 1;
        %Store the calculated probabilites of the current state in s0(st-1)
        %to ensure first-order Markov assumption.
        s0 = st;
    elseif strcmp(i,'n') == 1
        UIcontinue = 0;
    else
        UIcontinue = 0;
        disp('Invalid input. Exiting...');
    end
end

%end of script
```