Artificial Intelligence

Bayesian Networks & Markov Models

James Heselden

HES15591313

15591313@students.lincoln.ac.uk

Contents

[Task 1a: Bayes Rule and the Product Rule 2](#_Toc25604253)

[Task 1b: Bayesian Networks and Inference 2](#_Toc25604254)

[Bayes Net: 2](#_Toc25604255)

[CPT Generation: 2](#_Toc25604256)

[Inference (Exact and Approximate): 3](#_Toc25604257)

[[Exact] Inference by Enumeration: 3](#_Toc25604258)

[[Approximate] Inference by Rejection Sampling: 4](#_Toc25604259)

[[Approximate] Inference by Likelihood Sampling: 4](#_Toc25604260)

[Results: 4](#_Toc25604261)

[Task 2: Markov Models 5](#_Toc25604262)

[Difference between States and Observations: 5](#_Toc25604263)

[Forward Algorithm: 5](#_Toc25604264)

[Likelihood Calculation: 6](#_Toc25604265)

[Results: 6](#_Toc25604266)

[Appendix 7](#_Toc25604267)

[Task 1a: 7](#_Toc25604268)

[Task 1b: 8](#_Toc25604269)

[Task 2: 15](#_Toc25604270)

# Task 1a: Bayes Rule and the Product Rule

This task required the implementation of the Bayes rule to calculate the conditional probability P(d|t), given the probabilities P(d), P(t|d) and P(¬t|¬d). Bayes rule is based in the product rule, which itself is derived from the rule for conditional probability (1a), rearranged to the product rule (1b):

***(1a) ->***

***(1b) ->***

*As the formulae for joint probability is defined as (1c), we can also conclude (1d) must be accepted. Thus (1e) is a valid reorganisation of (1b). This means (1f) is valid and leads to (1g) defined as* ***Bayes Rule****.*

***(1c) ->***

***(1d) ->***

***(1e) ->***

***(1f) ->***

***(1g) ->***

The implementation of Bayes rule is fairly straightforward, as it only requires simple substitution into the equation. To make use of Bayes rule, the probabilities; P(d), p(t) and p(t|d) are required; of these, P(d) and P(t|d) are given. P(t) is calculated as the sum of the product of all scenarios where T = true. This consists of P(t|d) and P(t|¬d). P(t|¬d) is calculated as 1-P(¬t|¬d); as such P(t) = P(d)P(t|d) + P(¬d)P(t|¬d). Substituting the values into Bayes rule allows the calculation of the required result.

On passing the values ***P(d)=0.0001***, ***P(t|d)=0.99***, and ***P(¬t|¬d)=0.95***, into the function the probability ***P(d|t)*** is calculated as: **0.00197628458498024**

# Task 1b: Bayesian Networks and Inference

## Bayes Net:

Bayes nets are defined as “…a directed graph in which each node is annotated with quantitative probability information.” (Russell and Norvig, 2014, pg. 520). The structure of connections in the network define the conditional relationship between nodes, where each node represents a random variable and a connection from node A to node B states the value of node B is in some way dependent on the outcome from node A. This is implemented as a list of Nodes created in topological order, where each Node contains a name, and a list of parents by minimum.

## CPT Generation:

Conditional Probability Tables (CPT) are tables which represent the different probability distributions at a node, given the evidence defined from its parent nodes. For a network containing only Boolean random variables, any node will have a table with a size of 2parent\_count to define the size of its CPT.

The calculation of a CPT table via data is split into 2 categories. For a node with 0 parents, the CPT will be calculated as (2a) while if a node has 1 or more parents, the CPT will be calculated as (2b). Both of which can be defined as the count where the query and evidence is consistent, divided by the count where the evidence alone is consistent. Against each of these counts, are a sampling value which is included to ensure no illegal values are generated within the division.

***(2a) ->***

***(2b) ->***

These are implemented by the function [**BAYESIAN\_NETWORK::CPT** ]. It takes an input of a dataset, the columns of interest within said dataset and the sequence to count the number of occurrences. It calls the function [**BAYESIAN\_NETWORK::counter**], which returns the total number of occurrences across a set of rows given a sequence. This is repeated again using the body of the 2 lists, factors this into the equation and returns the result.

## Inference (Exact and Approximate):

Inference is the process of deriving new axioms from an existing knowledge base with the requirement that the new axioms must come from probabilistic reasoning and not out of thin air.

There are many forms of inference which work within Bayesian networks, but they all lie under one of 2 categories, exact inference and approximate inference. Exact inference makes use of predefined rules about the setup of a network and the relationships between nodes, such as conditional probability rules, Bayes rule, etc. Approximate inference makes likely estimations based on limited knowledge about certain aspects predefined within the network. Both have their benefits and drawbacks in different situations. Deciding on the correct method of inference depends on the setup of the Bayes net.

There are a number of considerations when deciding on the most appropriate method of inference, the main 2 are the size of the network, and the network complexity. Approximate inference such as rejection and likelihood sampling scale linearly with the number of nodes, so increasing the amount of nodes in the network does not greatly impact its performance, on the contrary, exact inference methods scale exponentially as for every additional hidden node added to the network, the total number of possible samples doubles.

The complexity also has a deeper impact on the performance and the choice of algorithm is not so simple, when deciding on how to process multiply connected networks, the use of clustering to join split paths can reduce the complexity of the network and the processing required from O(n2) to O(n).

## [Exact] Inference by Enumeration:

Inference by enumeration (2d) defines the joint probability distribution of a set of evidence is equal to the product of each nodes in the network given that evidence. Inference by enumeration essentially iterates through each possible set of node values, calculates a product based on the CPTs at each node, and takes the sum of these products.

***(2d) ->***

This is implemented under the function [ **INFERENCE::enumeration** ]. The function takes an input of some evidence, and a query. It then proceeds to go through every possible sample, factoring in the evidence into the sample. For each sample, a probability of the query is calculated, then the sum of each of these probabilities is returned. The product is calculated by taking the conditional probability retrieved for each node given the evidence defined within the sample.

## [Approximate] Inference by Rejection Sampling:

The basis for this is in generating prior samples. Prior sampling is done by looping through each node in the network generating an event based on the prior probability. This sample is then compared against the evidence, with any samples where the evidence does not match, being removed. The final distribution is calculated by normalizing the count *N* of the full probability *P(A|B)* against the evidence alone *P(B)*.

The complexity is stacked against the total number of nodes in the network, allowing for a linear increase in processing for the more nodes added. Unfortunately, for smaller networks, this sampling may actually take more processing time then the exact inference methods. To get accurate numbers, a significant number of samples must be generated as so many are rejected, with the number rejected increasing exponentially as more evidence values are added; for this network, a total sample count of around 500 gives reasonable values compared with the exact values returned from enumeration.

This is implemented by simply looping through the total number of samples. For each node within the sample, the event is calculated; if the node value is defined within the evidence, and the evidence does not match, the sample is rejected. For each completed sample, a counter is incremented. If the query value also matches the value defined in the sample a second counter is incremented. Once all samples have been generated, the distribution is calculated as the query counter divided by the total counter.

## [Approximate] Inference by Likelihood Sampling:

The biggest issue with rejection sampling is the sheer amount of wasted processing as samples are generated which don’t match the evidence. Likelihood sampling aims to fix this by, literally fixing the state to be consistent with the evidence. For each sample generated, the product of the costs to fix each state is calculated based on the states of the parents of the evidence. The final probability is calculated as the sum across all the cost products where the query is met, divided by the sum across all cost products.

The accuracy of Likelihood Sampling is impacted on by the amount of evidence but the location of the evidence within the network also has an impact, where if evidence is defined relatively close to the base of the network, the accuracy will decrease as there is more randomness piling on before the evidence definition is satisfied.

## Results:

Running the 3 implementations of inference returns the following results, where the time is normalised relative to enumeration:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Enumeration** | **Rejection Sampling (500)** | **Likelihood Sampling (500)** |
| **< P(SM|CO,FA), P(¬SM|CO,FA)>** | *<0.8342, 0.1658>* | *<0.8571, 0.1429>* | *<0.8195, 0.1805>* |
| **Distance from Target** | *0* | *0.0229* | *0.0147* |
| **Time to Run** | *x1* | *x1.7038* | *x1.8120* |

As we can see here, the inference calculated for rejection sampling is the worst, being twice the distance from the exact value compared with the likelihood distribution. And given the time being .1 times slower, a gain of nearly 64% is clearly superior.

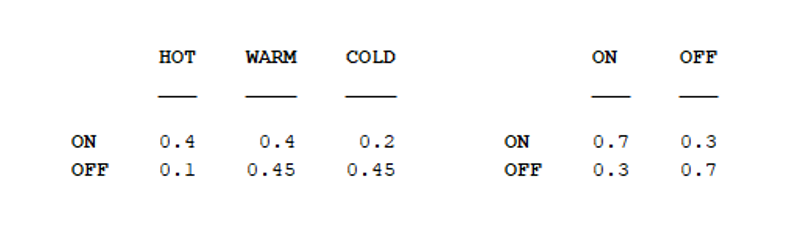
# Task 2: Markov Models

## Difference between States and Observations:

Networks mimicking the real world are made up of 2 types of nodes, node which exhibit an observable effect, and those which are the underlying cause of an observable effect. When modelling probabilistic reasoning over time, these are classified into observations and states. Within Markov systems, observations and states are modelled as 2 sets of conditional probability tables where.

For the given task, the transition model is created as a 2\*2 matrix where one diagonal is 0.7, and the other diagonal is 0.3, so P(Xt | Xt-1) is passed into the matrix to calculate the probability distribution of a change in state; where Xt defines the state at the current time-step and Xt-1 defines the state at the previous time-step. The sensor model is built for the task as a 3\*2 matrix, which define the probability P(Et | Xt), where Et defines the observed state.

Figure . (a): Sensor Model P(Et | Xt) (b): Transition Model P(Xt-1|Xt)



(a)

(b)

## Forward Algorithm:

The forward algorithm derives from a recursive conditional probability chain combined with the Bayes rule and the sensor Markov Assumption. With the aim at each time-step to generate the probability of a transition, given the observation occurring, and the state at the previous time-step P(Xt | Xt-1, et).

For a recursive system such as the Markov model being developed, the aim is to return a conditional probability of a state given all evidence since the last known state, this can be framed as a function *f* which takes the latest evidence, and the probability distribution of the old state given the historical sensor values. This can be represented as the equation (3a).

***(3a) ->***

By rearranging the contents of the function, we can redefine it stating the Probability of X given the historical evidence and the evidence at time t+1 (3b).

***>>***

***(3b) ->***

*To make the calculations simpler, and enable the elimination of some factors, we first apply the Bayes rule to the equation (3b), this allows us to replace the conditional probability of the state given the evidence, to the normalized product of the probability of the evidence given the state and past values of e, and the state given the past values of e. This results in equation (3c).*

***>>***

***(3c) ->***

*The sensor Markov Assumption is then applied here which aims to remove the reliance on the historical evidence, stating the recursive estimation of the state is valid unless proven otherwise. By applying this assumption, we are able to completely remove the reliance on historical values of the evidence and state, the Markov Assumption is applied, removing the historical elements, leaving the equation as (3d).*

***>>***

***(3d) ->***

*Equation (3d) can be modified further into a state where the components can be generated solely from the transition and sensor model. When substituting some arbitrary values in as t, and t+1, we can see that the probability distribution of the state given the evidence can be calculated as the joint probability of the transition model and the sensor model across all values for the previous estimation of the state. Thus this calculation can be applied to the equation (3e), replacing the final component in (3d).*

***>>***

***>>***

***>>***

***(3e) ->***

Equation (3e) is implemented in code by looping through each of the timesteps and recursively saving the normalised value of the independent hidden states, using the evidence from the previous step. The initial probability distribution of the state of the heater is defined as equiprobable before the recursion starts.

## Likelihood Calculation:

The likelihood calculation is an almost identical calculation to the forward model (3e), with the only change being the normalisation at each timestep is removed creating equation (3f). This normalisation is removed to return the overall probability given some time t.

***(3f) ->***

The likelihood of a sequence occurring, can be calculated once the likelihood at each hidden variable is found. This calculation makes use of equation (3g), which defines the likelihood of a sequence of evidence (*L*) from time 1 to time t, as the sum of the likelihood distribution for the hidden variable x at time t (3h).

***(3h) ->***

The implementation of this is the same as the forward model, but with the normalization constant removed, and the final element in the list summed over.

## Results:

The script returns the result of [**e(1:t): 0.0043996**], when called with input of [**Calculate\_Liklihood(["COLD","WARM","HOT","WARM","COLD"])**].

# Appendix

## Task 1a:



## Task 1b:

### Main Script:



### Bayesian Network Setup:



### CPT Functions:





### Inference by Enumeration:



### Inference by Rejection Sampling



### Inference by Likelihood Sampling



## Task 2:

### Main Script:

