Bayesian Networks

Li Xuanguang, A0154735B

1 Introduction

Bayesian networks are a type of probabilistic graphical model with a set of random variables that has a possible mutual causal relationship. The network is a Directed Acyclic Graph (DAGs), with nodes representing variables, edges representing the casual relationship between the nodes, and each node having a conditional probability distribution [6]. Bayesian networks are usually used for calculating the probability of some variable given a complex situation with many inter-related variables which maybe causal in nature. When there is uncertainty, complexity, and casuality in the model, Bayesian network will be very useful [5].

In this report, we will explore the theoretical concepts required to understand Bayesian networks, followed by some examples of building Bayesian networks, then finishing by covering the Hidden Markov Model (HMM) and the Viterbi Algorithm. We assume that the reader is familiar with probability theory.

2 Fundamentals of Bayesian Networks

In this section, we introduce the concept of d-Separation, followed by model building.

2.1 Casual Networks and d-Separation

A casual network is a *directed graph* with a set of variables and directed links (arcs) between these variables. If there is an arc from variables A to B, then B is the child of A, and A is the parent of B. When the state of a variable is known, it is *instantiated*. This now allows us to define an important concept.

Definition 1. *(d-Separation)* [6] Two variables A and B are *d-separated* in a casual network if, for all paths between A and B, there is another variable X such that either

i. the connection is serial (Figure 2.1.1a) or diverging (Figure 2.1.1b) and X is instantiated; or
ii. the connection is converging (Figure 2.1.1c), and neither X nor any of X's descendants are instantiated.

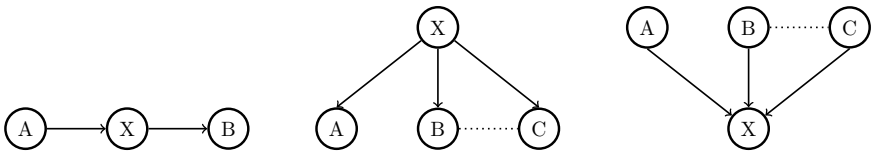

Figure 2.1.1: Serial connection, diverging connection, and converging connection diagram

We can also define the Markov blank of a variable, which has the property that when the variable is instantiated, the node is separated from the rest of the network.

Definition 2. [6] The *Markov blanket* of a variable A is the set consisting of the parents and children of A, as well as variables sharing a child with A.

Relations between variables can be defined quantitatively with the use of Bayesian networks. Note that the edges in Bayesian networks does not imply causality, but when building the model, we ensure that the model's d-separation properties correspond to the real world's conditional independence properties.

Definition 3. [6] A *Bayesian network* consists of:

i. a set of variables and set of directed edges between variables;
ii. each variable having finite set of mutually exclusive states;
iii. the entire graph is a Directed Acrylic Graph (DAG), which means there is no feedback cycles;
iv. each variable A with parents $B_1, \ldots, B_n$ and conditional probability $P(A \mid B_1, \ldots, B_n)$.

Recall the general chain rule in probability theory:
Given a set of variables $\alpha = \{A_1, \ldots, A_n\}$, for any probability distribution $P(\alpha)$ we have

$$P(\alpha) = P(A_n|A_1, \ldots, A_{n-1})P(A_{n-1}|A_1, \ldots, A_{n-2}) \cdots P(A_2|A_1)P(A_1). \tag{1}$$

Adapting this for Bayesian networks, we have the following chain rule [6]:
Assume the Bayesian network contain the set of variables $\alpha = \{A_1, \ldots, A_n\}$. The network has a unique joint probability distribution $P(\alpha)$ given by the product of all conditional probability tables of the network:

$$P(\alpha) = \prod_{i=1}^{n} P(A_i|\mathrm{pt}(A_i)) \tag{2}$$

where $\mathrm{pt}(A_i)$ are parents of A_i, and $P(\alpha)$ reflects the properties of the Bayesian network.
This yields a compact representation of the joint probability distribution of Bayesian networks. We now have enough background to build the Bayesian models.

2.2 Bayesian Model Building

We can consider the following framework [6] for building a Bayesian network model:

i. Form the hypothesis events, which are the events not directly observable (or observable only at an unacceptable cost). These events are then grouped into sets of mutually exclusive and exhaustive events to form the *hypothesis variable*.
ii. Identify the informational channels which may provide a certainty estimate on the hypothesis variables, and group these information into *information variables*.
iii. Establish the directed links for the Bayesian network.

2.2.1 Hybrid Bayesian Models

Bayesian models with both discrete and continuous variables are *Hybrid Models*. We would require additional constraints [6] in order to perform exact probability updating on the networks:

i. Each continuous variable must be assigned a (linear) conditional Gaussian distribution. For each configuration $\mathbf{c}$ of discrete parents, the variance $\sigma^2_{\mathbf{c}}$ is a constant, and the mean $\mu_{\mathbf{c}}$ is a linear function of the continuous parents $Y_1, \ldots, Y_m$.

ii. No discrete variable have continuous parents.

If the continuous variable does not have any parents, then it is assigned an unconditional normal distribution. If the above two conditions cannot be met, the continuous variable may be transformed into variables with a finite number of states for modelling.

2.2.2 Hidden Markov Models

Note that Bayesian models may assume the Markov property (where the past has no influence on the future) due to d-separation property. Hence, this is suitable for modelling data with the time variable. *Hidden Markov Models* are strictly repetitive models with the Markov property, as explained in the next section.

3 Hidden Markov Models

The aim of Hidden Markov Models (HMM) is to determine an observed sequence by the underlying process which is unobservable. The states of the HMM cannot be directly estimated from the observed data. By the Markov property, the next state relies only on the current state and transition probability between states. The classic algorithms to solve the HMM are as follows [8]:

i. Viterbi Algorithm: for finding the most likely sequence

ii. Forward-Backward Algorithm: for probabilistic inference and sampling from the distribution on hidden variables given observed variables

iii. Baum-Welch Algorithm: estimates the parameters of the HMM (initial distribution, transition matrix, and emission distributions) using the forward-back algorithm and Expectation Maximisation.

3.1 Structure of the Hidden Markov Model

We define the mathematical structure of HMM. Note that an observed sequence on the HMM is related to the underlying Markov process, which means the number of observable states may differ from that of hidden states. Hence, HMM must comprise of two sets of states and three sets of probabilities [4]:

i. *Hidden States* $y_{t,i}$: of the system (time t, state value i), described by a Markov process

ii. *Observable States* x_t: (at time t) of the process that are visible (i.e., measurable)

iii. *Initial Probability Distribution* π_0: containing probability that the model is in one of the hidden states at the initial time. Note that $\pi_{0i} \in \pi_0$ such that $\pi_{0i} = p(y_{0,i}) = p(y_0 = i)$

iv. *State Transition Matrix A*: containing transition probabilities $a_{i,j}$ of one hidden state to another hidden state (or itself), given the previous hidden state. Note that $a_{i,j} = p(y_t = i | y_{t-1} = j) = p(y_{t,i} | y_{t-1,j})$

v. *Emission Probability Matrix B*: containing probability $b_{t,i}$ that the particular measurable state can be observed, given that the model is in one of the hidden states. Note that $b_{t,i} = p(x_t | y_t = i) = p(x_t | y_{t,i})$

A simple hidden state sequence diagram can be visualised as follows (we omit the state value i for simplicity):

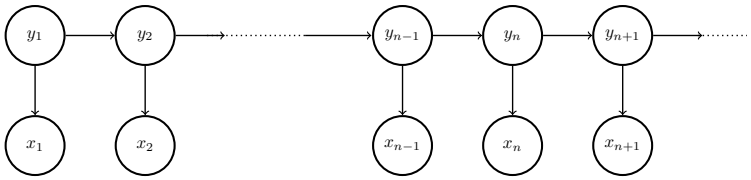

Figure 3.1.1: A simple HMM with only one hidden state per time frame

By the Markov property and Equation (2), we can show that the total probability of the model is

$$p(y_{1:n}, x_{1:n}) = p(y_1)p(x_1|y_1)\prod_{t=2}^{n} p(y_t|y_{t-1})p(x_t|y_t) \tag{3}$$

where $y_{1:n} = (y_1, y_2, \ldots, y_n)$ is the hidden states, and $x_{1:n} = (x_1, x_2, \ldots, x_n)$ is the sequence of observations. For notation, let M be the number of hidden state values, N be the horizon of the HMM.
Then $p(y_{t,m}) = p(y_t = m), m \in M$ is a possible state, and $n \in N$ is the timestamp.

3.2 Viterbi Algorithm

The Viterbi Algorithm aims to find the most likely sequence $y^*_{1:n} = \{y^*_1, \ldots, y^*_n\}$ with maximal probability given $x_{1:n} = (x_1, x_2, \ldots, x_n)$. Formulated, this is finding $y^*_{1:n} \in \arg\max_{y_{1:n}} p(y_{1:n}|x_{1:n})$, or equivalently,

$$y^*_{1:n} \in \arg\max_{y_{1:n}} p(y_{1:n}, x_{1:n}) \tag{4}$$

as $p(x_{1:n})$ is constant with respect to $y_{1:n}$.

3.2.1 Intuition for the Algorithm

Note that Viterbi algorithm is a dynamic programming algorithm fulfiling the following conditions:

i. Optimal substructure property: the optimal path solution $y^*_1, \ldots, y^*_i, \ldots, y^*_n$ contains the optimal substructure solution $y^*_1, \ldots, y^*_i$.
ii. Overlapping subsolutions property: if both y^*_i and y^*_j are in the optimal path solution, where $j > i$, then the calculation of the probability for being in state y^*_i is part of calculations for being in state y^*_j.

We also assume that the HMM has limited horizon, i.e., the last state $n < \infty$, and the HMM is of first-order.
We have the following intuition for working out the algorithm (with time complexity of only $O(M^2N)$):

1. Calculate the probability of reaching each state once for each time step
2. Then memoise this probability in a Dynamic Programming table (in 2 Trellis, $\delta_m(n)$ and $\zeta_m(n)$):
   i. The trellis has size $(M + 2) \times (N + 2)$, with states M as rows, the timestep N as columns
   ii. Each cell of $\delta_m(n)$ records the probability of most likely path that ends in state $y_{m,n}$ at time n, while $\zeta_m(n)$ is a backpointer:

$$\delta_m(n) = \max_{1 \le i \le M} [\delta_i(n-1)a_{i,m}b_{n,m}], \quad \text{where } a_{i,m} \in A, b_{n,m} \in B \tag{5}$$

$$\zeta_m(n) = \arg\max_{1 \le i \le M} [\delta_i(n-1)a_{i,m}b_{n,m}], \text{ where } a_{i,m} \in A, b_{n,m} \in B \tag{6}$$

where A and B is as defined in Section 3.1

iii. Probability is calculated by maximising over the optimal paths of going to y_j for each y_i.

3.2.2 The Algorithm (Pseudocode)

Definition 4. The *Viterbi algorithm* has input: Initial Probability Distribution π_0, State Transition Matrix A, and Emission Probability Matrix B and for horizon $N < \infty$, and perform the following steps:

1. Initialise the variables:

$$\delta_m(1) = \pi_{0m} b_{1,m} \quad 1 \leq m \leq M \tag{7}$$

$$\zeta_m(1) = 0 \qquad\quad 1 \leq m \leq M \tag{8}$$

2. Use recursion and memoisation:

$$\delta_m(n) = \max_{1 \leq i \leq M} [\delta_i(n-1) a_{i,m} b_{n,m}] \quad 1 \leq m \leq M, 1 \leq n \leq N \tag{9}$$

$$\zeta_m(n) = \arg\max_{1 \leq i \leq M} [\delta_i(n-1) a_{i,m} b_{n,m}] \quad 1 \leq m \leq M, 1 \leq n \leq N \tag{10}$$

3. Terminate the program

$$\text{Highest probability achieved is: } prob* = \max_{1 \leq i \leq M} \delta_i(n)$$

$$\text{The optimal path is: } y^*_{1:n} \quad = \arg\max_{1 \leq i \leq M} \delta_i(n)$$

As the algorithm computes products of probabilities, this may result in arithmetic overflow/underflow issues. Hence, we may use logarithms in the equations to deal with the issue (as logarithm is order-preserving).

3.3 Forward-Backward Algorithm

The Forward-Backward algorithm aims to find the likelihood that a sequence is generated by the HMM [2]:

i. $p(x_{1:n})$: the probability of $x_{1:n}$ given the HMM

ii. $p(x_{i:j})$: the probability of a substring of $x_{1:n}$, given the HMM

iii. $p(y_i = m | x_{1:n})$: the posterior probability that the ith state is m, given the sequence $x_{1:n}$

3.3.1 Intuition for the Algorithm

Note that for $p(y_{1:n}|x_{1:n}) = \frac{p(x_{1:n}, y_{1:n})}{p(x_{1:n})}$, the normalisation constant is $p(x_{1:n})$. To compute $p(x_{1:n})$ efficiently, we try to find recursive formulas which we can apply dynamic programming to.

The algorithm consists of two parts:

i. Forward Algorithm: sum over $y_1, y_2, \ldots, y_n$ in that order, and derive the recursion for computing $p(x_{1:j}, y_j)$ for each $y_j = m$, $1 \leq m \leq M$ and each $1 \leq j \leq N$

ii. Backward Algorithm: sum over $y_N, y_{N-1}, \ldots, y_1$ in that order, and derive the recursion for computing $p(x_{j+1,N}|y_j)$ for each $y_j = m$, $1 \leq m \leq M$ and each $1 \leq j \leq N$

The forward and backward algorithm both have a time complexity of $O(M^2 N)$

3.3.2 The Algorithm (Pseudocode)

The inputs for both the forward and backward algorithm are same as that of Viterbi Algorithm.
The algorithm takes as input the Initial Probability Distribution π_0, the State Transition Matrix A, and the Emission Probability Matrix B and for the time horizon $N < \infty$.

Definition 5. For the *Forward Algorithm*:

1. For each $y_1 = m$, $m \in M$, compute $\sigma_1(y_1) = p(y_1)p(x_1|y_1)$
2. For each $j = 2, \ldots, N$ and for each $y_j = m$, $m \in M$, compute

$$\sigma_j(y_j) = \sum_{y_{j-1}} \sigma_{j-1}(y_{j-1})p(y_j|y_{j-1})p(x_j|y_j) \tag{11}$$

3. Terminate the program to find $p(x_{1:N}) = \sum_{y_N} \sigma_N(y_N)$

Note that $\sigma_j(y_j) = \sum_{y_{1:j-1}} p(x_{1:j}|y_{1:j}) = p(x_{1:j}, y_j)$, which can be used to compute many useful results, such as predicting the next values x_{j+1} given $x_{1:j}$:

$$\begin{aligned} p(x_{j+1}) \propto p(x_{1:j}, x_{j+1}) &= \sum_{y_j, y_{j+1}} p(x_{1:j}, x_{j+1}, y_j, y_{j+1}) \\ &= \sum_{y_j, y_{j+1}} p(x_{1:j}, y_j)p(y_{j+1}|y_j)p(x_{j+1}|x_j) \end{aligned}$$

Definition 6. For the *Backward Algorithm*:

1. For each $y_n = m$, $m \in M$, compute $\delta_N(y_N) = 1$
2. For each $j = N-1, N-2, \ldots, 1$ and for each $y_j = m$, $m \in M$, compute

$$\delta_j(y_j) = \sum_{y_{j+1}} p(y_{j+1}|y_j)p(x_{j+1}|y_{j+1})\delta_{j+1}(y_{j+1}) \tag{12}$$

3. Terminate the program to find $p(x_{1:N}) = \sum_{y_j} p(y_j)p(x_j|y_j)\delta_j(y_j)$

Note that $\delta_j(y_j) = \sum_{y_{j+1:N}} p(x_{j+1:N}, y_{j+1:N}|y_j) = p(x_{j+1:N}|y_j)$.

3.4 Baum-Welch Algorithm

With the background of Forward-Backward Algorithm, we can now understand the Baum-Welch Algorithm. The algorithm provides a way to measure the HMM parameters of the Initial Probability Distribution π_0, the State Transition Matrix A, and the Emission Probability Matrix B.
This algorithm iterates the forward-backward algorithm until convergence, and is a special case of the class of Expectation-Maximisation (EM) Algorithms.

3.4.1 Background on Expectation-Maximisation Algorithms

The aim of EM algorithms is to find a Maximum Likelihood Estimate (MLE) or Maximum a os- teriori (MAP) estimate for models with hidden or unobserved variables. As models with hidden variables is often complicated and multimodal, computation for maximisation is difficult with no guarantee of an optimal

solution (even with EM algorithms).
However, if there are observed variables along with hidden variables in the model (hence having 'complete data') that is modelled as exponential family, then the optimisation is computationally efficient.

A brief summary of the EM algorithm is as follows [8]:
Model is $(X, Y) \sim p_\theta(x_{1:n}, y_{1:n})$, where $x_{1:n} = (x_1, \ldots, x_n)$ is observed data, $y_{1:n} = (y_1, \ldots, y_n)$ is hidden variable, and $p_\theta(x_{1:n}, y_{1:n})$ is an exponential family.
The goal is to find $\theta_{MLE} \in \arg\max_\theta p_{\theta(x_{1:n})}$ where $p_{\theta(x_{1:n})} = \sum_y p_\theta(x_{1:n}, y_{1:n})$.

1. Initialise the variable: θ_1 with randomisation
2. While convergence is not met,

    i. Expectation-Step: compute

$$Q(\theta, \theta_k) = \mathbb{E}_{\theta_k}(\log p_\theta(X, Y)|X = x_{1:n}) = \sum_y (\log p_\theta(x_{1:n}, y_{1:n}))p_{\theta_k}(y_{1:n}|x_{1:n})$$

    ii. Maximisation-Step: solve for $\theta_{k+1} \in \arg\max_\theta Q(\theta, \theta_k)$

We can now adapt this algorithm for HMMs (hence forming the Baum-Welch Algorithm).

3.4.2 The Algorithm (Pseudocode)

The Baum-Welch Algorithm aims to find the HMM parameters of Initial Probability Distribution π_0, the State Transition Matrix A, and the Emission Probability Matrix B. There are a few phases for this algorithm: the initial phase, the expectation step, and the maximisation step.

From Definition 5 (the Forward Algorithm) and Definition 6 (the Backawrd Algorithm), we have

$$\sigma_j(y_j) = \sum_{y_{1:j-1}} p(x_{1:j}|y_{1:j}) = p(x_{1:j}, y_j) \tag{13}$$

$$\delta_j(y_j) = \sum_{y_{j+1:N}} p(x_{j+1:N}, y_{j+1:N}|y_j) = p(x_{j+1:N} \tag{14}$$

With these equations, we can easily determine the probability distribution of the hidden variable at any time j given the entire sequence of observed data:

$$p(y_j|x_{1:N}) = \frac{\sigma_j(y_j)\delta_j(y_j)}{p(x_{1:N})} \tag{15}$$

We also define the below functions to be used in the maximisation step:

$$\gamma_j = p(y_j|x_{1:N}) = \frac{\sigma_j(y_j)\delta_j(y_j)}{\sum_{y_j} \sigma_j(y_j)\delta_j(y_j)}$$

$$\beta_j = p(y_j, y_{j-1}|x_{1:N}) = \frac{\sigma_j(y_j)\delta_{j-1}(y_{j-1})p(y_{j-1}|y_j)p(x_{j-1}|y_{j-1})}{\sum_{y_j} \sigma_j(y_j)\delta_{j-1}(y_{j-1})p(y_{j-1}|y_j)p(x_{j-1}|y_{j-1})}$$

Now we are ready to show the Baum-Welch Algorithm:

1. Initial Phase: randomise the values of the parameter matrices π_0, A, and B.
2. While values are not converged, repeat the steps:
   i. Expectation-Step: Compute σ and β with the forward-backward algorithm with the current state values of π_0, A, and B.
   ii. Maximisation-Step: For all $n \in N$, update π_0, A, and B with the formulas below:

$$\pi^*_m = \gamma_0$$
$$a^*_{ij} = p(y_n = j | y_{n-1} = i) = \frac{\sum_n \beta(y_n = j, y_{n-1} = i)}{\sum_n \gamma(y_{n-1} = i)}$$
$$b^*_{ij} = p(x_n = j | y_n = i) = \frac{\sum_n \gamma(y_n = i) \times 1_{y_n = j}}{\sum_n \gamma(y_n = i)}$$

4 Extensions of Hidden Markov Models

In this section, an outline of more advanced HMMs is introduced to show how the basic HMM can be extended to solve more complex mathematical problems.

4.1 Infinite State Hidden Markov Model

The infinite state HMM (iHMM) is designed to work around limitations of the standard HMM [3] such as:

i. Specification of model structure in advance. In real scenarios, it is unreasonable to assume a priori believe that the data was generated from a limited number of discrete states.
ii. We may not know the form the state transition matrix takes on.
iii. Over/underfitting due to maximum likelihood methods not considering the complexity of the model.

The iHMM model brings many applications to fields requiring usage of Bayesian nonparametrics [10].

4.2 Hierarchical Hidden Markov Model

The Hierarchical Hidden Markov Model (HHMM) is an extension of the standard HMM, where each state is a self-contained HHMM. If a state in the HHMM is activated, it will activate its own underlying HHMM, and the process is repeated until the production state that behaves like a standard HMM state [1].
HHMM is used together with Dynamic Bayesian Networks (DBNs), which are Bayesian Networks with variables that may link to itself at a future timestate [7].
We dive deeper into an application of HHMM in quantitative finance, specifically for High Frequency Trading on Foreign Exchange [9] in the appendix.

References

[1] Johanna Appel. Hierarchical hidden markov models, Mar 2022.

[2] Serafim Batzoglou. Lecture 6: Hidden markov models continued.

[3] Matthew Beal, Zoubin Ghahramani, and Carl Rasmussen. The infinite hidden markov model. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[4] Ramaprasad Bhar and Shigeyuki Hamori. *Hidden Markov Models: Applications to Financial Economics*. Springer, 2004.

[5] Finn B. Jensen. *An introduction to Bayesian Networks*. UCL Press, 1996.

[6] Finn B. Jensen and Thomas Graven-Nielsen. *Bayesian networks and decision graphs*. Springer, 2007.

[7] Bayes-Server Learning-Center. Dynamic bayesian networks - an introduction.

[8] Jeffrey W. Miller. Hidden markov models, 2016.

[9] Javier Sandoval and German Hernandez. Learning of natural trading strategies on foreign exchange high-frequency market data using dynamic bayesian networks. *Machine Learning and Data Mining in Pattern Recognition*, pages 408 – 421, 2014.

[10] Ioannis Sgouralis and Steve Presse. An introduction to infinite hmms for single-molecule data analysis. *Biophys J.*, 112(10), May 2023.

Appendix

A Research Paper Summary

We summarise a research paper as part of the project requirements.

A.1 Learning of Natural Trading Strategies on Foreign Exchange High-Frequency Market Data Using Dynamic Bayesian Networks [9]

The paper makes use of Dynamic Bayesian Networks (DBN) to model the Forex market as a complex non-deterministic dynamic system. The aim of such models is to predict prices.

A.1.1 Development of Price Prediction Models

DBN was first applied with the standard HMMs. Referencing to Hassan, a 4-hidden state system was used, with current day's open, high, low and closing prices used, with aim to predict next day's closing price.
Extending the model requires usage of HHMM to model different levels of time hierarchy. Jangmin's model uses 5-hidden states of strong / weak bear, random walk, weak / strong bull, with each state having second-level abstract states, which called the third-level (production) state for output emission (Section 4.2).
Tayal and Wisebourt also adapted the HMM to high-frequency data with a 2-hidden state model with runs and reversal, with each state having second-level state that define the emitted zig-zag observations. This high-frequency model was considerably affected by market liquidity, with less liquid asset performing worse.

A.1.2 Representation of Model

The author uses HHMM, with states encoded by vector $\mathbf{Q_t} = \{Q_t^1, \ldots, Q_t^D\}$, where D is the number of network levels. The DBN representation is a set of indicator variables $F_t^d, d = \{1, \ldots, D\}$ that determines if the HMM at level d at time t has finished. The model uses the following states, with $D = 2$:

i. $Q^1(1) = \{\text{Market State 1}\}$
ii. $Q^1(2) = \{\text{Market State 2}\}$
iii. $Q^2(1), Q^2(4) = \{\text{Negative Feature Producer of Market State 1 and 2}\}$
iv. $Q^2(2), Q^2(3) = \{\text{Positive Feature Producer of Market State 1 and 2}\}$

This allows the author to create the DBN and HHMM as follows:

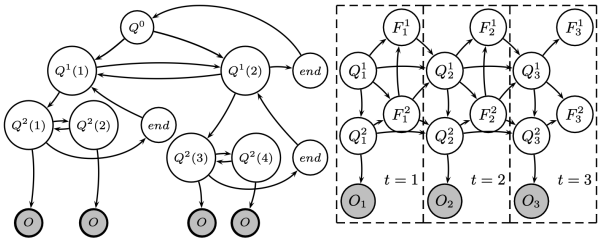

Figure A.1.1: The DBN on the left, and HMM on the right [9]

A.1.3 Construction of Observed Variables

The author constructs a feature vector to capture information from transaction data and order book.
The author performs the following operations on transaction data:

1. Create a timeseries of local extrema from the transaction data
2. Identify the zig-zag process by taking difference between n-adjacent local extrema

The order book data is transformed to supplement the data:

1. Create the volume-weighted average limit order book price (VWAP) (which captures the order book's depth changes between transaction zig-zag points)
2. Calculate the imbalances in the order book by taking differences in the VWAP spreads

Thereafter, the author creates a discrete feature vector with three elements:

1. Zig-zag pattern type (capture min-max of transaction price)
2. Transaction local extrema data (capture transaction price momentum)
3. Order book dynamics with imbalance data (capture increasing/decreasing liquidity in order book)

The HHMM is able to model an expert trader's knowledge of evolution of the markets, as the market will entire a regime, and then positive / negative features are produced; there is guarantee that the positive and negative feature will alternate.

A.1.4 The Dataset

Three months (2 Feb 2012 to 31 May 2012) of tick-by-tick information from limit order book, and transactions of USD/COP spot rate is used. Data was extracted from the Set-FX data vendor, with over 43,000 transactions and over 658,000 book order updates of first 10 orders in each side of the book.

A.1.5 Methodology, Results, and Conclusion

Feature vector series is divided into training (14 days), validation (8 days), and backtest (38 days) data.
The trading strategy used is a momentum based, i.e. long position in uptrend market, and short position in downturn market.

The author run the EM algorithm 50 times to maximise likelihood given observed data over the training set. From the results, some runs of the trading strategies are profitable in training, but when tested on validation data, fails to generalise in risk/return performance.
The author has also run the Forward-Looking Viterbi inference for benchmarking, which indicates that when future information is used, the performance greatly exceeds the non-Forward-Looking Viterbi algorithm (that do not use future information).

The author then concludes that the model outperforms passive strategies of buy-and-hold and sell-and-hold, and that the market-regime classification ability of the model is significant.

A.1.6 Comments and Critique of the Research Paper

The author has chosen a rather illiquid (USD-COP) forex pair to conduct research on, rather than a more liquid (EUR-USD) forex pair. The data chosen significantly impacts the performance of the research, as the

author has already highlighted (in Section A.1.1) that other researchers have findings that high-frequency models are considerably affected by market liquidity. Yet, the author tries to adapt their own variation of Tayal and Wisebourt's HMM (that is a 2-hidden state model that has runs and reversal) to this dataset (which focuses on intraday forex trading).

In addition, the domain of application is an issue. Forex market modelling is much more complicated due to possible (direct or indirect) government intervention such as buying/selling foreign reserves to prop/lower their own currency, or raising interest rates on government bonds (such as what the US is doing right now in 2022) may impact the trend and regime more than simply 'market momentum'. Perhaps the US equity market will be a better domain to apply the model with due to simpler market dynamics (as there are less levers of action that may influence the momentum).

Lastly, the author uses a forward-looking model that looks at future data as the 'best performance benchmark'. This is a poor benchmark to use, as the author is essentially doing a 'look-forward' bias on the dataset, which leads to very skewed (and in this case extremely good) performance on the benchmark model. All the author's tested model also fails to take into account the trading costs, hence making the model appear better in performance than it should be.

As a conclusion, the paper may not be a good candidate for actually implementing a HMM-based trading strategy for live trading.