Hidden Markov Chain

Aziz Lohourignon

Winter 2025

Outline

- Reminder: Markov chain
- Hidden Markov Model
- Estimating parameters
- Application 1: Coin tossing game
- Application 2: Detecting bearish and bullish markets in financial time series using HMM

Markov Chain I

Definition:

- A **Markov chain** is a stochastic process that represents a sequence of events or states, where the future state depends only on the present state and not on past states.

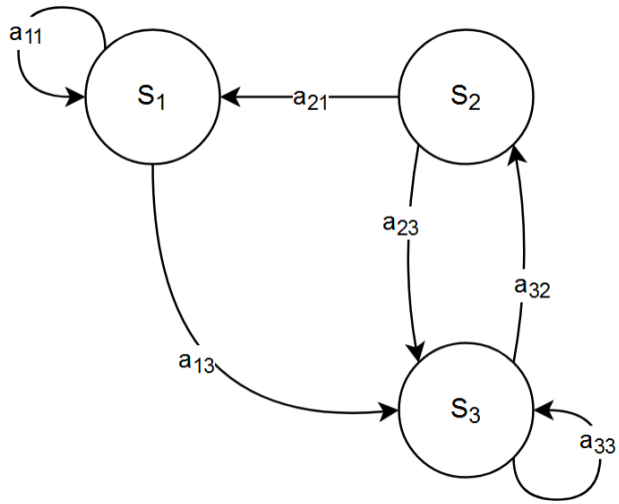
Let $S = \{S_1, S_2, \ldots, S_n\}$ be the set of all possible states, and let $X = \{X_k \mid X_k \in S,\ k = 1, \ldots, T\}$ be the time series of states.
The **Markov property** states that for any $k \geq 0$ and states $X_0, X_1, \ldots, X_k$, we have:

$$P(X_{k+1} = S_j \mid X_k = S_i, X_{k-1}, \ldots, X_0) = P(X_{k+1} = S_j \mid X_k = S_i). \quad (1)$$

Remark: In other words, the probability of transitioning to state S_j at time step $k+1$ depends only on the current state S_i at time step k, and not on any previous states.

Markov Chain II

- Formally, a first-order Markov chain is defined by the set of states S and a transition probability matrix $A = [a_{ij}]$, where a_{ij} represents the probability of transitioning from state S_i to state S_j in one step.



The elements of the transition matrix must satisfy the following stochastic constraints:

- $0 \leq a_{ij} \leq 1, \quad \forall i, j$
- $\sum_{j=1}^{n} a_{ij} = 1, \quad \forall i.$

Hidden Markov Model (HMM)

- The model presented earlier implicitly assumes that each state corresponds to an observable (physical) event.
- However, many real-world scenarios involve underlying states that influence observations but are not directly observable.
- This limitation led to the development of **Hidden Markov Models (HMMs)**, which extend the basic Markov chain model by introducing hidden or unobservable states that affect the observed data.

Example:

- Consider a coin-flipping game where the player has two coins: one biased and one fair.
- We do not know which coin is used for each flip, but we observe the outcomes of the flips.
- The hidden state represents the coin being used, while the observable state corresponds to the result of the flip.

Elements of HMM I

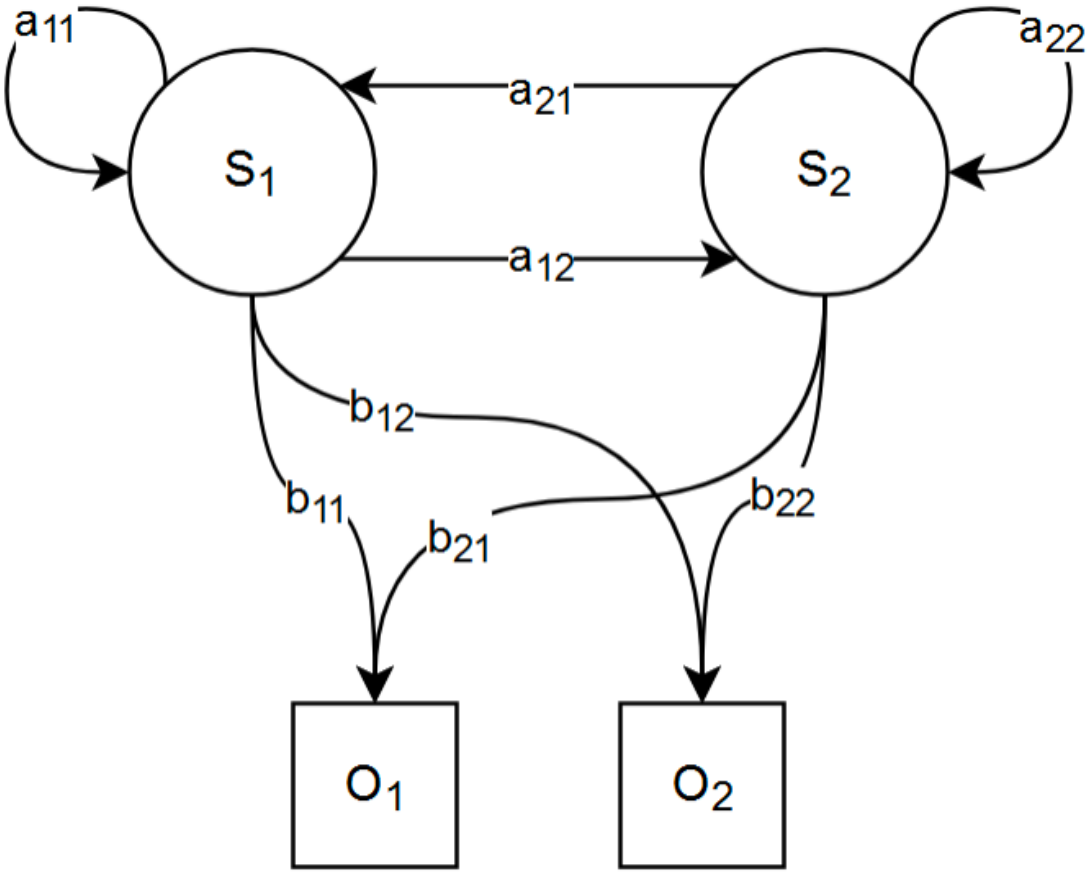
- A **Hidden Markov Model (HMM)** consists of a hidden state process that follows a Markov chain, where each state generates an observation based on a probability distribution that depends only on the current state.
- Let $O = \{O_t \mid O_t \in O, t = 1, \ldots, T\}$ be the observed sequence, where O is the set of possible observations. The hidden states evolve according to a Markov process, and the probability of an observation $o \in O$ being emitted from state S_i is given by the emission probability function $b_i(o)$.
- The emission probability matrix is denoted as $B = \{b_{jk} = b_j(o_k)\}$.

Elements of HMM II

Key Properties:

- The model has a finite number of states, denoted as N.
- At each time step t, the system transitions to a new state based on a transition probability distribution, which depends only on the previous state (Markov property).
- After each transition, an observation is generated according to a fixed probability distribution associated with the current state.

Elements of HMM III



Notation

We formally define the notation for a discrete observation Hidden Markov Model (HMM):

Model Parameters:

- T : Length of the observation sequence (total number of time steps).
- N : Number of states.
- M : Number of possible outcomes.
- $Q = \{q_1, q_2, \ldots, q_N\}$: Finite set of states.
- $V = \{v_1, v_2, \ldots, v_M\}$: Finite set of possible outcomes.

Probability Distributions:

- $A = \{a_{ij}\}$, where $a_{ij} = P(q_j \mid q_i)$: State transition probability matrix.
- $B = \{b_j(k)\}$, where $b_j(k) = P(v_k \mid q_j)$: Outcome probability distribution for state q_j.
- $\pi = \{\pi_i\}$, where $\pi_i = P(q_i \text{ at } t = 1)$: Initial state distribution.

Compact Notation:

$$\lambda = (A, B, \pi)$$

Estimating Parameters – The Three Fundamental Problems of HMMs

Problem 1: Evaluation (Likelihood Computation) Given an observation sequence $O = (O_1, O_2, \ldots, O_T)$ and a model $\lambda = (A, B, \pi)$, how do we compute $P(O \mid \lambda)$, the probability of the observation sequence given the model?

Problem 2: Decoding (State Sequence Inference) Given an observation sequence $O = (O_1, O_2, \ldots, O_T)$, how do we determine the optimal state sequence $Q = (q_1, q_2, \ldots, q_T)$ that best explains the observations according to some meaningful criterion?

Problem 3: Learning (Parameter Optimization) How do we adjust the model parameters $\lambda = (A, B, \pi)$ to maximize $P(O \mid \lambda)$, i.e., how do we train the HMM to best fit the observed data?

Solutions to the three HMM Problems

Problem 1: Evaluation I

We aim to compute the probability of the observation sequence O given the model $\lambda = (A, B, \pi)$. The most straightforward approach is to enumerate all possible state sequences of length T.

- For a fixed state sequence $Q = (q_1, q_2, \ldots, q_T)$, the probability of the observation sequence given O and λ is:

$$P(O \mid Q, \lambda) = b_{i_1}(o_1) b_{i_2}(o_2) \cdots b_{i_T}(o_T).$$

- The probability of such a state sequence I, on the other hand, is given by:

$$P(Q \mid \lambda) = \pi_{i_1} a_{i_1 i_2} a_{i_2 i_3} \cdots a_{i_{T-1} i_T}.$$

- The joint probability of O and Q, i.e., the probability that both occur simultaneously, is given by:

$$P(O, Q \mid \lambda) = P(O \mid Q, \lambda) P(Q \mid \lambda).$$

Problem 1: Evaluation II

- To compute $P(O \mid \lambda)$, we sum over all possible state sequences:

$$P(O \mid \lambda) = \sum_{Q} P(O \mid Q, \lambda) P(Q \mid \lambda).$$

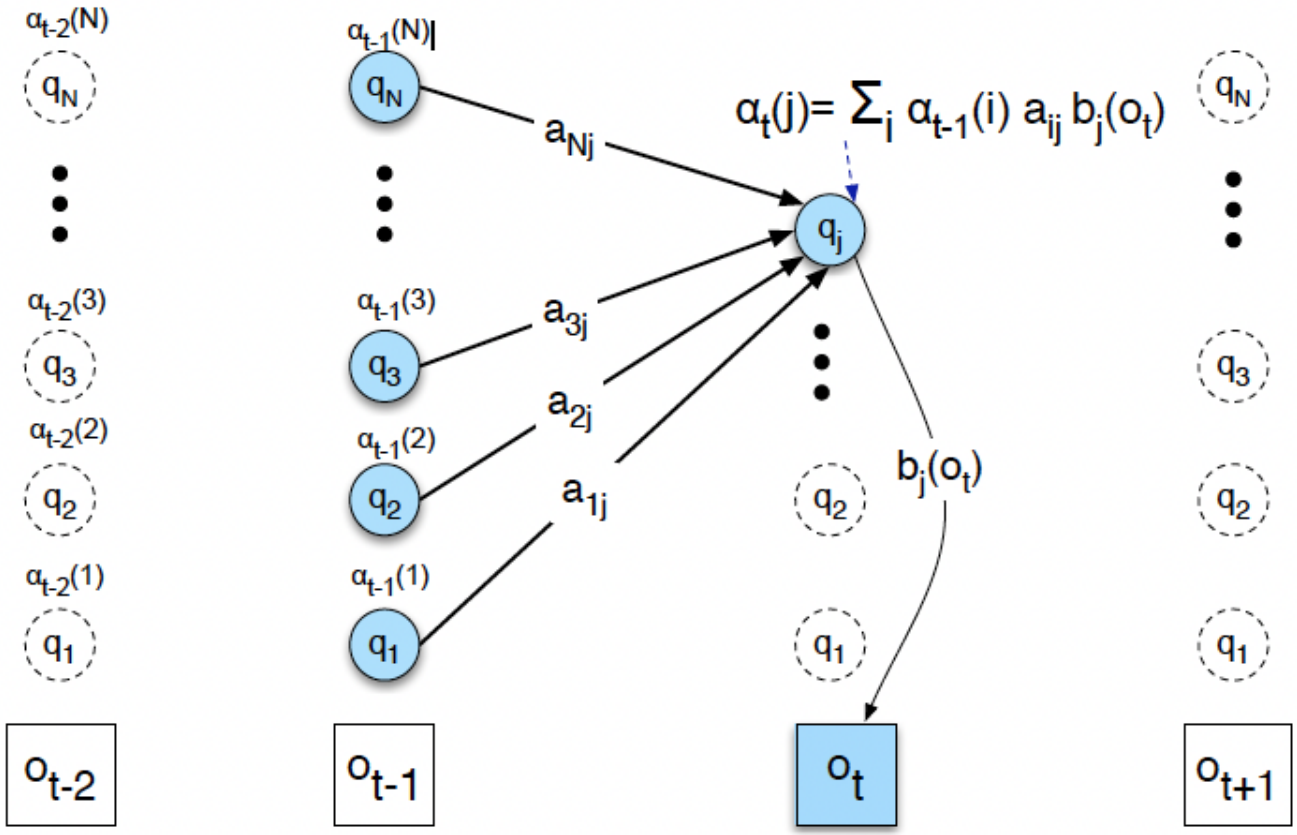
- Since this direct computation has an exponential complexity of $O(N^2 T)$, we use a more efficient approach in practice: the **Forward Algorithm**.

Problem 1 : Forward Algorithm

- The Forward Algorithm is a type of dynamic programming algorithm.
- Each cell of the Forward Algorithm table, $\alpha_t(j)$, represents the probability of being in state j after observing the first t observations, given the model λ.
- The value of each cell $\alpha_t(j)$ is computed by summing over the probabilities of all possible paths leading to this state. Formally, each cell expresses the following probability:

$$P(o_1, o_2, \ldots, o_t, q_t = j \mid \lambda).$$

Forward Algorithm



Speech and Language Processing. Daniel Jurafsky James H. Martin.
Copyright © 2024.

Forward Algorithm: Step by Step

1. **Initialization:**

$$\alpha_1(j) = \pi_j b_j(o_1), \quad 1 \leq j \leq N.$$

2. **Recursion:**

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t), \quad 1 \leq j \leq N, \quad 1 \leq t \leq T.$$

3. **Termination:**

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Problem 2 : Decoding the State Sequence

- There are several possible approaches to solving Problem 2, which involves finding the optimal state sequence associated with a given observation sequence.
- One possible optimality criterion is to select the states i_t that are individually most likely. This approach maximizes the expected number of correctly identified individual states.
- However, this criterion and its corresponding solution may present some issues. When certain transitions are disallowed (i.e., $a_{ij} = 0$ for some i and j), the resulting state sequence may, in fact, be impossible.
- This method determines the most likely state at each time step without considering the overall structure of the table, the temporal dependencies between states, or the total length of the observation sequence.

Problem 2 : The Viterbi Algorithm

- Like the Forward Algorithm, the Viterbi Algorithm is a type of dynamic programming algorithm that utilizes a dynamic programming table.
- The algorithm iteratively computes the most likely path to each state at each time step, taking into account both the current observation and the probabilities of the previous states.
- Instead of selecting the states that are individually most likely, the algorithm tracks the most probable path through the state sequence.
- The Viterbi Algorithm is structurally identical to the Forward Algorithm, except that instead of summing probabilities, it selects the maximum probability at each step.

Viterbi Algorithm: Step by Step I

1. **Initialization:**

$$\text{Viterbi}_1(j) = \pi_j b_j(o_1), \quad 1 \leq j \leq N$$

$$\psi_1(j) = 0, \quad 1 \leq j \leq N$$

2. **Recursion:**

$$\text{Viterbi}_t(j) = \max_{1 \leq i \leq N} \text{Viterbi}_{t-1}(i) a_{ij} b_j(o_t), \quad 1 \leq j \leq N, \quad 1 < t \leq T$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} \text{Viterbi}_{t-1}(i) a_{ij} b_j(o_t), \quad 1 \leq j \leq N, \quad 1 < t \leq T$$

3. **Termination:**

$$P^* = \max_{1 \leq i \leq N} \text{Viterbi}_T(i)$$

$$q_{T^*} = \arg \max_{1 \leq i \leq N} \text{Viterbi}_T(i)$$

Viterbi Algorithm: Step by Step II

- $\text{Viterbi}_t(j)$ represents the probability that the HMM is in state j after observing the first t observations and passing through the most probable state sequence $q_1, \ldots, q_{t-1}$, given the model λ.

$$\text{Viterbi}_t(j) = \max_{q_1,\ldots,q_{t-1}} P(q_1, \ldots, q_{t-1}, o_1, o_2, \ldots, o_t, q_t = j \mid \lambda)$$

- The state-observation likelihood of the observation symbol o_t, given the current state j, is represented by $\psi_t(j)$.
- The best score is given by:

$$P^* = \max_{1 \leq i \leq N} \text{Viterbi}_T(i)$$

- The starting point of the backtrace is determined by:

$$q_{T^*} = \arg \max_{1 \leq i \leq N} \text{Viterbi}_T(i)$$

Problem 3: Parameter Estimation

- The solution to this problem is typically addressed using the forward-backward algorithm, also known as the Baum-Welch algorithm.
- The Baum-Welch algorithm is a specific implementation of the Expectation-Maximization (EM) algorithm, tailored for Hidden Markov Models (HMMs). It iteratively performs three main steps: the forward step, the backward step, and the update step.
- In the forward step, the algorithm computes the probability of being in a specific state at each time step, given the observed sequence up to that point.
- In the backward step, the algorithm computes the probability of observing the remaining part of the sequence from a given state at each time step. This step complements the forward procedure by calculating the backward variable, denoted as $\beta_k(i)$, which represents the probability of being in state q_i at time k and observing the remaining sequence from time $k+1$ to the end.

Baum-Welch algorithm : Backward Step

- The backward probability $\beta_t(i)$ represents the probability of observing the sequence from time $t+1$ to the end, given that we are in state i at time t (and given the automaton λ):

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, \ldots, o_T \mid q_t = i; \lambda)$$

- It is computed inductively in a similar manner to the forward algorithm.

Backward Step - Step by Step

1. **Initialization:**

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

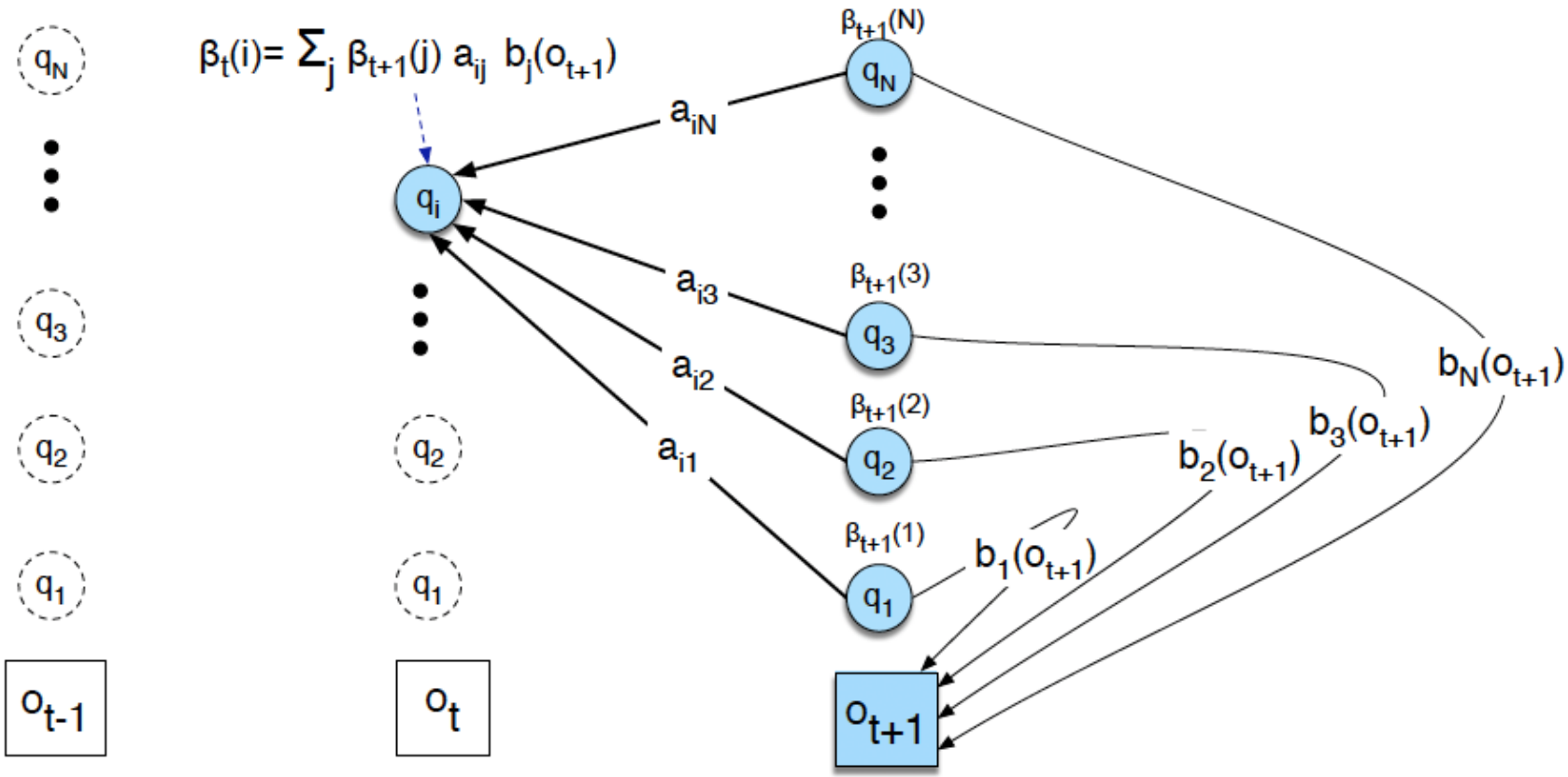
2. **Recursion:**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, \quad 1 \leq t < T$$

3. **Termination:**

$$P(O \mid \lambda) = \sum_{j=1}^{N} \pi_j b_j(o_1) \beta_1(j)$$

Backward algorithm



Speech and Language Processing. Daniel Jurafsky James H. Martin. Copyright © 2024.

Baum-Welch Algorithm

- Let's begin by examining how to estimate $\hat{a}_{ij}$ using a variant of simple maximum likelihood estimation:

$$\hat{a}_{ij} = \frac{\text{Expected number of transitions from state } i \text{ to state } j}{\text{Expected number of transitions from state } i}$$

- How do we compute the numerator?
- Assume we have an estimate of the probability that a given transition $i \to j$ occurs at a particular time t in the observation sequence. If we knew this probability for each time t, we could sum over all time steps to estimate the total count for the transition $i \to j$.

Baum-Welch Algorithm

More formally, let's define $\xi_t(i,j)$ as the probability of being in state i at time t and state j at time $t+1$, given the observation sequence and the model:

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O; \lambda)$$

To compute ξ_t, we first compute a probability similar to ξ_t, but with the addition of the observation probability. Note the difference in the conditioning of the observation O:

$$\tilde{\xi}_t(i,j) = P(q_t = i, q_{t+1} = j, O \mid \lambda)$$

Baum-Welch Algorithm: Update of Matrix A I

The quantity $\tilde{\xi}_t(i,j)$ is given by:

$$\tilde{\xi}_t(i,j) = \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$$

To compute ξ_t from $\tilde{\xi}_t$, we apply the laws of probability and divide by $P(O \mid \lambda)$, since

$$P(X \mid Y, Z) = \frac{P(X, Y \mid Z)}{P(Y \mid Z)}$$

The probability of the observation given the model is simply the forward probability of the entire sequence (or equivalently, the backward probability of the entire sequence):

$$P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i) = \sum_{j=1}^{N} \pi_j b_j(o_1) \beta_1(j)$$

Update of Matrix A II

Therefore, the final equation for $\xi_t(i,j)$ is:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O \mid \lambda)}$$

The expected number of transitions from state i to state j is then the sum over all t of $\xi_t(i,j)$. For our estimate of a_{ij}, we need one more component: the total expected number of transitions from state i. This can be obtained by summing over all transitions out of state i. Here's the final formula for $\hat{a}_{ij}$:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)}$$

Baum-Welch Algorithm: Update of Matrix B I

We also need a formula for recomputing the observation probability. This is the probability of observing a given symbol v_k from the observation vocabulary V, given a state j:

$$\hat{b}_j(v_k) = \frac{\text{Expected number of times in state } j \text{ and observing symbol } v_k}{\text{Expected number of times in state } j}$$

For this, we need to know the probability of being in state j at time t, which we will denote as $\gamma_t(j)$:

$$\gamma_t(j) = P(q_t = j \mid O; \lambda)$$

Update of Matrix B II

Once again, we will compute this by including the observation sequence in the probability:

$$\gamma_t(j) = \frac{P(q_t = j, O \mid \lambda)}{P(O \mid \lambda)}$$

The numerator is simply the product of the forward and backward probabilities:

$$\gamma_t(j) = \alpha_t(j)\beta_t(j)$$

We are now ready to compute $\hat{b}_j(v_k)$. For the numerator, we sum $\gamma_t(j)$ for all time steps t in which the observation o_t is the symbol v_k that we are interested in. For the denominator, we sum $\gamma_t(j)$ over all time steps t. The result is the percentage of the times we were in state j and saw symbol v_k. The notation $\sum_{t=1}^{T}\{o_t = v_k\}$ means "sum over all t for which the observation at time t was v_k":

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^{T}\{o_t = v_k\}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$$

Forward-Backward Algorithm

We now have methods to re-estimate the transition matrix A and observation matrix B from an observation sequence O, assuming an initial estimate of A and B. These re-estimations form the core of the iterative forward-backward algorithm.
The algorithm starts with an initial estimate of the HMM parameters $\lambda = (A, B, \pi)$ and iterates as follows:

- Compute the expected state occupancy counts $\gamma_t(j)$ and the expected state transition counts $\xi_t(i, j)$ based on the current estimates of A and B.
- Use $\gamma_t(j)$ and $\xi_t(i, j)$ to update A and B.
- Repeat until convergence.

FORWARD-BACKWARD: Step by step

- Initialize A and B
- Iterate until convergence:
  - **E-step:**

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)} \quad \text{for all } t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \quad \text{for all } t, i, \text{ and } j$$

  - **M-step:**

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N}\xi_t(i,k)}$$

$$\hat{b}_j(v_k) = \frac{\sum_{t=1}^{T}(o_t = v_k)\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$$

- Return A, B

Application 1: Coin Tossing Game

Coin Tossing Game

- In this first application, we consider a coin-tossing game where one coin is biased towards tails and the other towards heads.
- We do not observe the coin directly but only the outcome of each toss. The goal is to determine which coin was used for each toss.
- We then compare our predictions with the actual results to assess the accuracy of our model.

Coin Tossing Game

- We begin our test without any prior assumptions, meaning that we initialize the transition matrix, emission matrix, and initial probability vector at 0.5.
- Under this setup, we achieve a correct prediction rate of 51.85
- However, by incorporating prior knowledge—such as the fact that once a coin is used, it is more likely to be used again in the next toss, and that one coin is biased towards heads while the other is biased towards tails—the model improves its accuracy.
- With this adjustment, the model achieves a correct prediction rate of 61.48

Coin Tossing Game

- This result highlights the significant impact of the initial guess on the model's performance.
- To improve the results, we can use a method that tests the model with different initial values and selects the one with the highest probability.

Application 2: Detecting bearish and bullish markets in financial time series using HMM

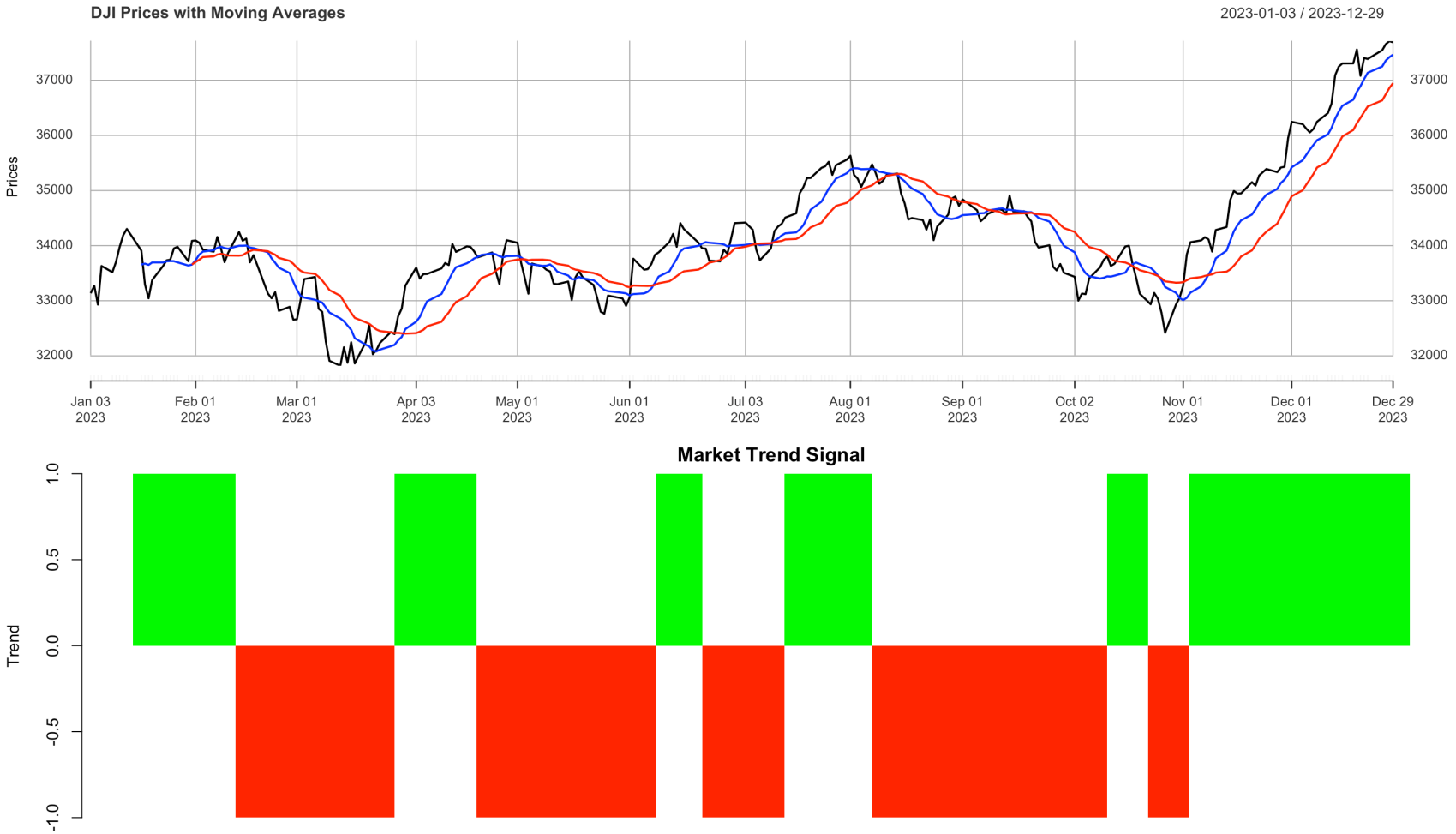
Trend detecting

- The objective of this implementation is to determine whether the market is in an uptrend or downtrend. Additionally, given an uptrend, we aim to estimate the probability of observing a negative return.
- We compare the effectiveness of a Hidden Markov Model (HMM) versus a moving average crossover strategy in generating trading signals.

Trend detecting: Data and Methodology

- We use daily price data of the Dow Jones Index from 01-01-2023 to 01-01-2024.
- Since we work in discrete time, we define a binary flag indicating whether the price increased (u) or decreased (d) during a trading session.
- This flag serves as the input for our model.

Trend detecting



Trend detecting

- We observe that the model consistently detects regime changes before the moving average.
- One might therefore consider using the model as a trading signal. However, it is important to keep in mind that the model was trained on this specific dataset, which explains its high accuracy.
- To validate the reliability of this model, backtesting should be conducted on a test set using a model trained on a separate training set.

Conclusion

- Hidden Markov Models (HMM) are powerful tools for modeling time series with latent states, as they can effectively capture regime changes and shifts in data patterns.
- One of the strengths of HMMs is their ability to work with discrete or continuous emission probabilities. In the case of continuous emissions, these models can capture correlations and complex dependencies within the data, which makes them particularly useful for more nuanced predictions.
- For a better model, one important consideration is increasing the number of observations, as more data typically leads to more reliable parameter estimation and better generalization to unseen data.

Conclusion

- Furthermore, to improve the performance of an HMM, factors such as selecting the right number of states, improving the feature selection, and addressing issues like overfitting should be carefully considered.
- Ultimately, a successful HMM model should balance model complexity with the available data, ensuring that it captures the underlying patterns without overfitting or underfitting.

References

- Jurafsky, D., Martin, J. H. (2024). Speech and Language Processing.
- Oelschläger, L., Adam, T. (2024). Detecting Bearish and Bullish Markets in Financial Time Series Using Hierarchical Hidden Markov Models.
- Rabiner, L. R., Juang, B. H. (1986). An Introduction to Hidden Markov Models.