Hidden Markov Models

Elliot Pickens Bata-Orgil Batjargal

January 17, 2020

Abstract

Following in the footsteps of many quantitative funds, in this paper we demonstrate how Hidden Markov Models can be used to predict future states of the financial market. Using a Hidden Markov Model, we analyzed Goldman Sachs Group's closing price from Jan 1, 2016, through Dec 31, 2018. We trained our model on the most recent 875 trading days and then tested our model by predicting the states associated with 220 holdout days not used for the training process. First, we defined three possible market states (stagnant, bull, or bear). Then we used the forward-backward algorithm to learn the parameters of a Hidden Markov Model starting from a set of approximate initial probability, transition, and emission matrices.

1 Background

Hidden Markov Models (HMM) are probabilistic models based on simple two assumptions. The first assumption and namesake of the model is the Markov assumption that only the present state is relevant when predicting the future. In other words, in a sequence of states $q_1, q_2, ..., q_{i-1}$, the next state(q_i) is only dependent on the preceding state (q_{i-1}) [3]. We can abstract the assumption with,

$$\textbf{Markov Assumption: } P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}). \tag{1}$$

The second assumption is output independence, which states that each observation is solely a result of the current un-observable state of the system. That is to say that each piece of data we possess is assumed to only occur as a result of the contemporary state of the system. For example, if we were trying to understand the the previous summer's weather patterns using some form of indirect data (like the amount of ice cream a friend consumed each day) the state of the weather on a given day would only be dependent on the level of ice cream consumption on that same day. Formally, output independence can be written as:

$$\textbf{Output Independence: } P(o_i|q_1...q_i...q_T, o_1...o_i...o_T) = P(o_i|q_i). \tag{2}$$

HHMs build upon more conventional Markov models using of *hidden* (latent) variables and recursive algorithms. With these latent variables, we can learn about the probabilities associated with a sequence of observed states. When implemented, HMMs ultimately take the form of matrices representing transition probabilities from state i to state j (a_{ij}), observation probabilities (b_o) for each state with the help of sequences of observations (o_t), initial probability distribution(π), and states (q_t) that caused the observations. These matrices are learned though a training process that tries to select the transition matrix (A), observation probability matrix (B), and initial probability distribution (π) that best explain the observations.

1.1 Markov Models

With the Markov assumption, we can construct Markov models using the following components: a set of states ($Q = q_1q_2...q_N$), a transition probability matrix ($A = a_{11}a_{12}...a_{n1}...a_{mn}$), and an initial probability distribution ($\pi = \pi_1, \pi_2, ..., \pi_N$) [3].

1.1.1 An Example Markov Model

In this section, we will explore how the Markov assumption can be used to define a model based on the example Markov chain shown in Figure 1.

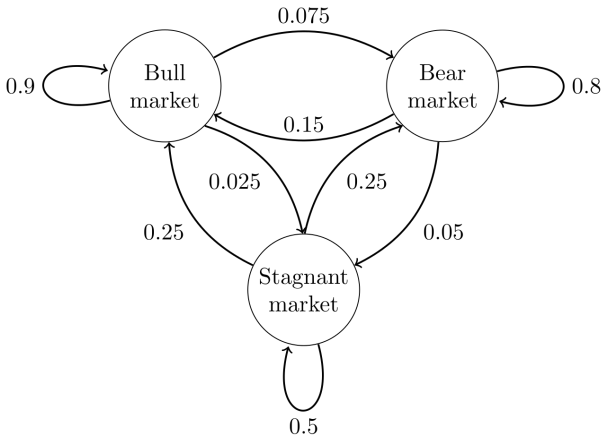

Figure 1: Graphical Representation of a Transmission Matrix for Market States where arrows stands for the probability to transition from one state to another[4]

For Figure 1, we can define a transition matrix P, where each state's transition probabilities are defined in a row vector. For example, $P(1,1)$ represents the probability of transitioning to bull, $P(1,2)$ to bear, and $P(1,3)$ to stagnant markets.

$$P = \begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}. \tag{3}$$

If we represent each state as a row vector x, with the relation $x^{(n+1)} = x^{(n)}P$. Then the state in $(n+3)$ is

$$\begin{aligned} x^{(n+3)} &= x^{(n+2)}P = \left(x^{(n+1)}P\right)P \\ &= x^{(n+1)}P^2 = \left(x^{(n)}P\right)P^2 \\ &= x^{(n)}P^3 \end{aligned}. \tag{4}$$

For example, if the system is in state 2 (bear) at time n, then at time $n+3$ the updated distribution is:

$$\begin{aligned} x^{(n+3)} &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}^3 \\ &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0.7745 & 0.17875 & 0.04675 \\ 0.3575 & 0.56825 & 0.07425 \\ 0.4675 & 0.37125 & 0.16125 \end{bmatrix} \\ &= \begin{bmatrix} 0.3575 & 0.56825 & 0.07425 \end{bmatrix}. \end{aligned} \tag{5}$$

From the above, we can see that the state is most likely to still be in the bear market with 0.57 probability at time $n+3$[4].

1.2 Hidden Markov Models Foundations

Markov models can be quite useful (as can be seen in section 2.1.1). However, they are handy when we can fully observe a sequence of events, and their causes are also a *hidden* sequence of events that we want

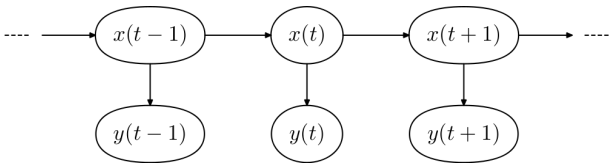

Figure 2: An example state emission sequence where x is the states that caused y observations[3]

to investigate.

Building on the foundations elements of Markov models (a set of states ($Q = q_1 q_2 ... q_N$), a transition probability matrix ($A = a_{11} a_{12} ... a_{n1} ... a_{mn}$), and an initial probability distribution ($\pi = \pi_1, \pi_2, ..., \pi_N$)), HMM make use of a sequences of observations ($O = o_1 o_2 ... o_T$) and observation probabilities ($B = b_i(o_t)$) to answer questions about the *hidden* events.

These five parts of the HMM are then glued together with the Markov assumption and an assumption of output independence. With them, we can now begin to think about how we might be able to discover *hidden* states that caused our sequence of observations. That discovery process (the essential task of an HMM) is covered in the following sections:

- **Likelihood**: Determine the likelihood $P(O|\lambda)$ of a set of observations O, given an HMM $\lambda = (A, B)$
- **Decoding:** Discover the most probable sequence of hidden states Q given O & λ
- **Learning:** Learn the parameters A & B of λ given O & Q

1.2.1 Likelihood: The Forward Algorithm

The forward algorithm is used to find the likelihood that a sequence of observations occurred for a given HMM λ. For example, if we do not know which hidden states were present and we want to find the probability that the observations $o_1, o_2, ..., o_n$ occurred we need to use the forward algorithm. That is to say, that in its most basic form the forward algorithm is used to find the likelihood of a given state happening at a given time.

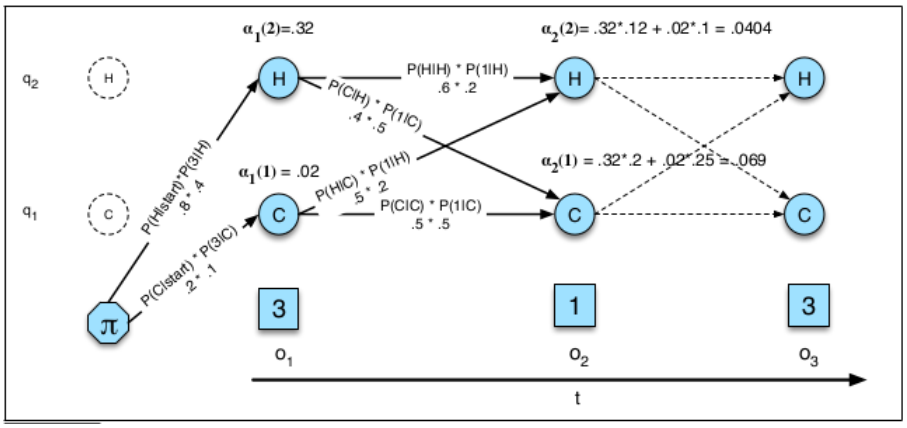

Figure 3: Forward Algorithm Diagram[3]

In order to find the likelihood of our observations O we need to first be able to find the probability that j is the t^{th} state following the first t observations and our HMM λ. We can find that probability value (denoted $\alpha_t(j)$) following all of the paths that could lead to state j and totalling their probabilities. This probability is commonly written as:

$$\alpha_t(j) = P(o_1, o_2, ..., o_t, q_t = j|\lambda) \tag{6}$$

To calculate $\alpha_t(j)$ the forward algorithm takes a pseudo-recursive approach (using dynamic programming) by summing over all paths that lead to the state j. This can be computed as

$$\alpha_t(j) = \sum_{i=1} \alpha_{t-1}(i) a_{ij} b_i(o_t). \tag{7}$$

1.2.2 Decoding: The Viterbi Algorithm

During the decoding process, we want to find the most probable sequence of hidden states given a set of observations and our HMM λ. For example, say we want to know whether a single day (in a series of days) was hot or cold based on the number of ice cream cones a friend of ours consumed on each day in the series. With the Viterbi algorithm, we can find the most likely set of hidden states (heat levels) given our observations (ice cream cones eaten).

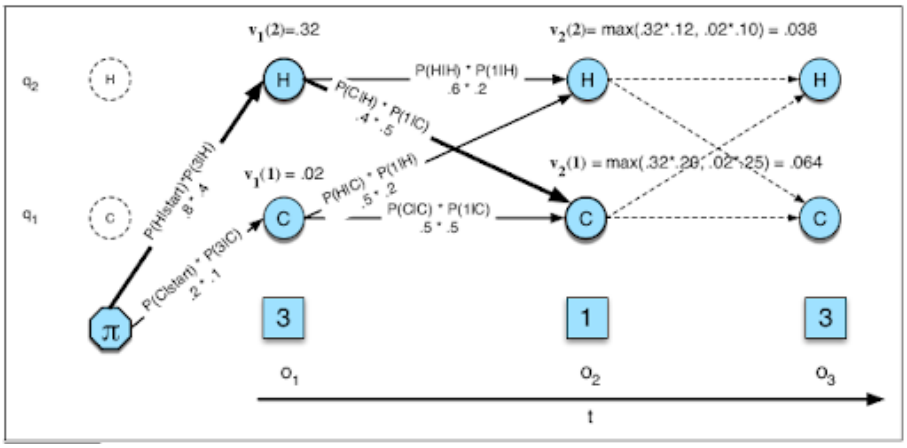

Figure 4: Viterbi Algorithm Diagram[3]

Given that we already have the forward algorithm it is tempting to consider simply applying it to all possible hidden state sequences and select the most probable one. The difficulty with such an approach is that its branching nature causes its computational complexity to grow exponentially with the number of observations. The Viterbi Algorithm drastically reduces the number of computations required by just calculating the maximum probability at each fork in the graph, as can be seen in Figure 4. The value associated with each state j at time t can be expressed as follows (by assuming we select the most probable state at each junction)

$$v_t(j) = \max_i v_{t-1}(i) a_{ij} b_i(o_t). \tag{8}$$

Like the forward algorithm, the Viterbi algorithm is a dynamic programming algorithm that recursively fills out its cells. Thus, at each step we have already calculated the probability of landing in each possible state during the previous time step and we can compute the probability of a state q_t as

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \tag{9}$$

1.2.3 Learning: The Forward-Backward Algorithm

During the learning phase our goal is to find the transition (A) and observations (B) matrices that best maximize the probability $p(O|\lambda)$. To maximize this value we will essentially formulate the problem as

maximum likelihood estimation. Such an MLE calculation is, however, very difficult to solve directly. To avoid the analytic difficulty of the problem we will use a form of the expectation maximization algorithm.

To setup this version of the EM algorithm we will first define transition probability a_{ij} (1) as the expected number of transitions from state i to state j *divided* by (2) expected number of transitions from state i. Also, let's define observation probability $b_j(v_k)$ (3) as the expected number of passes through state j and observing symbol v_k *divided* by the expected number of passes through state j (4).

Since the forward-backward algorithm is a case of the EM algorithm it consists of two steps: the expectation step where we calculate the expectation values (1), (2), (3), and (4) and the maximization step where we calculate a_{ij} and $b_j(v_k)$.

function FORWARD-BACKWARD(*observations* of len T, *output vocabulary* V, *hidden state set* Q) **returns** *HMM=(A,B)*

initialize A and B
iterate until convergence
E-step

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} \quad \forall\, t \text{ and } j$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} \quad \forall\, t,\ i, \text{ and } j$$

M-step

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

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

return A, B

Figure 5: Forward-Backward Pseudo Algorithm[3]

In the expectation state we calculate the probability of being in state i at time t and state j at time $t+1$. Then we find the probability of being at each state j at each time t. So we calculate the backward probability that tells us if the current state will be j at time $t+1$ and the likelihood of observing the remainder of the sequence up to time T. We also calculate a forward probability (using the Forward algorithm) to find the likelihood of being in the state i at time t (as we can see in Figure 6). In other words, Forward-backward probability, given the transition and observation matrices, is the probability of observing both future observations and past observations up to a time t (where i and j occurred consecutively).

$$P(forward-backward)_t(i,j) = \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)[3] \tag{10}$$

The forward-backward probability can be expressed as $P(X,Y|Z)$, where Z denotes the transition and observation matrices we are learning, X is observation where state i and state j were consequent, and Y is set of observations. We are interested in $P(X|Y,Z)$ because we want to know the likelihood of a transition from state i to state j. Following the laws of the probability, we can write

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)}[3] \tag{11}$$

Considering $P(Y|Z)$ is the probability of seeing the observations given the matrices being learned it is product of the forward and backward probabilities (equaling to $\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)$). So, we can calculate expected number of transitions from state i to state j (1) as

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}.[3] \tag{12}$$

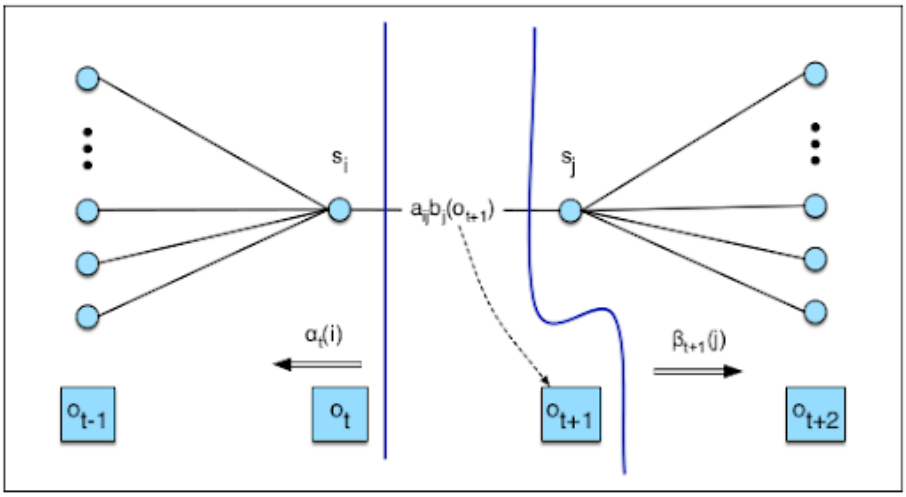

Figure 6: Forward-Backward Algorithm Diagram[3]

Furthermore, the probability of being in state j at time t can be expressed as

$$\gamma_t(j) = P(q_t = j|O,\lambda) = \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \tag{13}$$

Since $P(q_t = j, O|\lambda)$ is a product of the forward and backward probabilities, we can write

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}.[3] \tag{14}$$

Now with the variables needed to find a_{ij} and $b_j(v_k)$ defined we can solve for the remaining variables in the forward-backward algorithm. The expected number of transitions from state i (2) is $\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)$. The expected number of passes through state j and observing symbol v_k (3) is $\sum_{t=1 s.t. O_t = v_k}^{T} \gamma_t(j)$. Finally, the expected number of times in state j (4) is $\sum_{t=1}^{T} \gamma_t(j)$. In Figure 5, we can see how all of these come together synchronously to learn a_{ij} and $b_j(v_k)$.

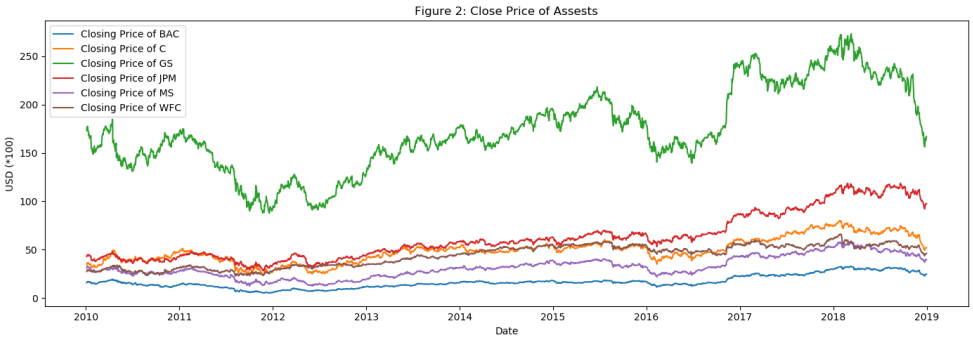

Figure 7: Asset Closing Prices

2 Methods

2.1 Set-up

We plan to analyze a single financial asset over a significant time frame (three years of daily data), using the daily closing prices for the asset we wish to analyze and a basket of related assets. With three years of data, we have over 1000+ observations (with 5-10 emissions (data points) per observation). Overall our data should be a large $\sim 1000 \times 10$ matrix. It should be noted, however, that for this project, we will only use a single feature in our data matrix to build our model (leaving us with a $\sim 1000 \times 1$ matrix). We chose to use a single feature data matrix to keep things in line with the theory we laid out in the previous section (in which we had only a single hidden state influencing each emission), but it is possible to expand the work done in the previous sections in order to create HMMs with multiple emissions per hidden state (i.e., $P(o_i|q_i) \rightarrow P(o_{i1}, o_{i2}, ..., o_{in}|q_i)$)[2].

Using the `investpy` Python package, we wrote a simple script that pulls publicly available stock market data from the web. With our script in hand we retrieved nine years worth of price data for 6 major publicly traded banks that we expect may be related in some way, since pulling the data of other major financial institutions could improve our model of a given bank's stock price)[1].

Having pulled all necessary data, we formatted the data and created extra features we thought might be helpful using Python's `pandas`[2]. `pandas` also allowed us to export our data into a simple CSV file that can be read in both Python and Matlab.

2.2 Procedure

When working with financial time series, we can imagine using an *HMM* to predict which state $s \in \{bear, bull, stagnant\}$ an asset is going to move into based on its current state, or predicting the current state underlying the market.

In order to use such an HMM, we first had to train it using our data. To do this, we extracted a daily closing price for Goldman Sachs and then split the series into the train, test, and validation sets using the `pandas` package[3]. We then converted the data in each of the sets into sequences of observations suitable for an HMM. In our case, each observation was either a 1 (if the difference between the current day's price and previous day's was > 0) and 0 otherwise[1].

We then applied the forward-backward algorithm to our training set to learn our A, B, and π. For this project, we used Python's `hmms` package to learn and use our HMM[4]. As was previously stated in

[1]The six banks we pulled data for are: Goldman Sachs, the Bank of America, Citigroup, J.P. Morgan, Morgan Stanley, and Wells Fargo

[2]We created three extra features for each stock (fractional change, fractional low, and fractional high)

[3]We chose to use the Goldman Sachs closing price because it is the most variable of the six series we pulled

[4]The `hmms` is fast and easy to use package, but almost completely lacks documentation and appears to have lost support so we would not generally recommend using it for HMM tasks. For those interested in using HMMs in Python, we would suggest

section 1.5, the forward-backward algorithm tends to become stuck in a local optimum and fails to learn a truly effective HMM. There are several ways to avoid this issue and find a useful model, but we chose to take a somewhat rough guess and check approach for this project. More specifically, we ran the forward-backward algorithm a number of times on our training set and selected the model that gave us the best results using the forward algorithm (the HMM that gave the highest probability of our observations occurs).

Figure 8: Predicted States for the Training Set (GS Closing Price)

Finally, we used our model to predict the states underlying our data using the Viterbi algorithm. We also predicted the sequence of underlying states that produced the test and validation sets using our HMM and the Viterbi algorithm to see how well our model performed on data not seen during the training process.

3 Results

Figure 9: Predicted States for the Validation Set (GS Closing Price)

The model we trained using over two years of daily trading data appears to approximate the state sequence with reasonably good accuracy, but it has several flaws. These flaws are not, however, unexpected. As mentioned in section 2.2, we trained our HMM to find the states $\{bear, bull, stagnant\}$ using only a single sequence of observations that were constrained to only two emission levels $\{0, 1\}$. Given these constraints, we cannot expect to have a particularly refined model, but we believe our results are promising and could easily be improved with further model tuning and increased complexity (i.e., the addition of extra states, emission levels, or observation sequences).

taking a look at the `hmmlearn` package

3.1 A Quick Note on Matlab

We used Python for a significant portion of this project because the library support for Python allowed us to "semi-manually" implement an HMM without having to write all necessary functions from scratch. However, we could have more easily done this project in Matlab using the HMM functions built into Matlab's *Machine Learning and Statistics* library. We only chose to use Python to build our stock price model, because we wanted to tinker with the functions in a way that is not possible using Matlab's locked down libraries.

3.2 Discussion/Further Applications

Throughout this paper, we have discussed how HMM can be used to find a sequence of hidden states given a set of observations. While that is the fundamental task of HMM, it should be noted that once we have our A, B, and π we can also use our model to make predictions. Predictions can be made by using our transition and transmission matrices to find the most probable next step in our sequence, given our current state.

We can also use the information gained by our HMM to inform other models that may have a better predictive capacity. For example, we could feed a sequence of predicted of states into another statistical learning algorithm as a series of one encoded variables, which the algorithm could use to (hopefully) better approximate the trends in our data[5].

Future prediction is, however, a secondary task of HMM and generally a secondary concern to those using HMM[5]. The most successful contemporary applications of HMM can be found in speech recognition and bioinformatics[3][6]. In both bioinformatics and speech recognition, many tasks require the discovery of a sequence of states, rendering them well suited to HMM. In speech recognition, for example, we can imagine a series of audio clips being defined as our observations and the words that produced them being the underlying states.

4 Conclusion

The hidden Markov model is a powerful tool in time series analysis, especially when we believe there may be an underlying pattern (or series of states) affecting our series.

With three reasonably simple algorithms (forward, Viterbi, and forward-backward), we can convert a basic Markov model into an HMM capable of finding hidden states. The forward algorithm allows us to evaluate sequences of observations to check either the likelihood of their occurrence or evaluate how well a given HMM is fit to a sequence of observations. The Viterbi algorithm gives us the ability to predict which states underpin our observations by selecting the most likely set of states given an HMM λ. Finally, the forward-backward algorithm lets us fit an HMM to a sequence of observations by repeatedly updating our model's parameters via a particular case of the expectation-maximization algorithm that employees forward and backward probabilities.

The efficacy of HMM can be seen in our simple example where we fit an HMM to stock market data in order to find the hidden market states. With just a simple model consisting of three possible states and two possible observation values, we were able to build a capable (if flawed model). Our constrained example shows the power held by a relatively un-tuned collection of three matrices.

References

[1] Danish A. Alvi. *Application of Probabilistic Graphical Models in Forecasting Crude Oil Price*. Papers 1804.10869. arXiv.org, Apr. 2018. URL: `https://ideas.repec.org/p/arx/papers/1804.10869.html`.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] At least as of December 2019

[3] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. ISBN: 9780131873216 0131873210. URL: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

[4] S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*. London: Springer-Verlag, 1993. URL: /brokenurl#probability.ca/MT.

[5] Nguyet Nguyen. "Hidden Markov Model for Stock Trading." In: *International Journal of Financial Studies* 6.2 (2018), pp. 1–17. URL: https://EconPapers.repec.org/RePEc:gam:jijfss:v:6:y:2018:i:2:p:36-:d:138097.

[6] Maria Servitja Robert. "A First Study on Hidden Markov Models and one Application in Speech Recognition." In: (2016), pp. 1–51.